Exploring Data Preprocessing by Dry Bean Classification

By Law Kiu Tsz (57812965), Tam Ho Fung (55773779), Nam Ho Sing (56681717)

Abstract

In 2020 Koklu and Ozkan used the dimension of beans to do a bean type classification [1]. Since then, numerous researchers have made significant contributions to further enhance this classification task. This project continues the research done by Koklu and Ozkan, aiming to explore the data preprocessing to achieve a better result. Feature selection, normalization, noise, and oversampling has been applied to the data set, and the best classification was done by using Z-score normalization, CFS feature selection, SMOTE and RF classifier. By achieving a PRC AUC of 0.99, this project also showed an enhancement compared to the previous research.

1. Introduction

This project continued the research started by Koklu and Ozkan [1] in 2020 classifying different types of dry beans from computer images. Different types of dry beans were captured by computer vision system, and the dimensions of the beans were obtained by image processing. Classification task was done based on the dimension features to identify different types of beans. Several studies conducted on this dataset have yielded positive outcomes, although the exploration of data preprocessing was not sufficiently thorough. Therefore, this project aims to further investigate and analyze additional data preprocessing methods.

2. Literature Review

2.1. Related Studies

The original data set was from the research started by Koklu and Ozkan [1] in 2020 building a classification system for identifying distinct types of dry beans. They built a computer vision system which captured images of the beans of 7 different species, from image processing the dimension features of the beans were obtained. The type of beans was then classified by using the dimension features. In this research support vector machine (SVM), random forest (RF) and k-nearest neighbors (KNN) were used for classification.

The research was then continued by M. Dogan et al. [2] focusing on the deep learning techniques in classification. Compared to the data set of Koklu and Ozkan consists of 7 different types of beans, M. Dogan et al. prepared a new set of data which consists of 14 types of beans, and the classification task was done by convolutional neural network (CNN) and extreme learning machine (ELM). The team applied a pre-trained GoogLeNet as CNN model, and further optimize it by ELM techniques.

M. S. Khan et al. [3] tried to address the class imbalance problem in the data set from the research of Koklu and Ozkan. They introduced Adaptive Synthetic (ADASYN) algorithm to oversample the minority class. The oversampled data was then passed to classifiers including RF, SVM, KNN, logistic regression (LR), Decision Tree (DT), Naïve Bayes (NB), XGBoost (XGB) and Multilayer perceptron (MLP) for classification.

J. C. Macuácua, J. A. Centeno, and C. Amisse [4] tried a different approach to solve te class imbalance problem in the data set of Koklu and Ozkan. They applied the Synthetic Minority Over-sampling Technique (SMOTE) technique, which is also an algorithm to oversample the minority class. They also applied principal component analysis (PCA) to the original data as well. For classification RF, SVM, KNN were used.

2.2. Project Initiatives

From the related studies there was already much research focusing on the classification methods. Different classifiers have been applied to the data set, for both traditional classification methods and deep learning algorithms. To further enhance the classification models, the research direction might be heading towards the cutting edge of deep learning.

Traditional Classifiers: LR, KNN, DT, RF, SVM, NB, XGB, MLP

Deep Learning: MLP, CNN, ELM

However, there was still space for preprocessing improvements. Most of the experiments done by previous researchers were using the original data, and SMOTE, ASASYNC and PCA were the only techniques applied to data preprocessing. Therefore, our research aimed at having a further exploration towards various data preprocessing methods, and the objective was achieving a better classification performance through data preprocessing.

3. Data set

3.1. Source of Data

The data set was “Dry Bean Dataset Classification” obtained from Kaggle [5]. It can be accessed publicly through the Kaggle website.

3.2. Class Labels

The data set consisted of 7 classes which were the different types of beans. The class distribution was imbalanced. The DERMASON was the majority class, and the BOMBAY was the minority class.

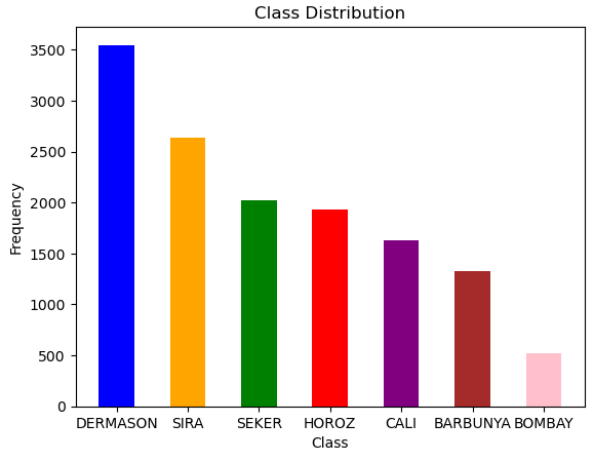

Fig. 1. Class distribution of the data set

Class	Count
DERMASON	3546
SIRA	2636
SEKER	2027
HOROZ	1928
CALI	1630
BARBUNYA	1322
BOMBAY	522
Total	13611

Table 1. Visualized class distribution

3.3. Dimension Features

Koklu and Ozkan applied image processing from the pictures of the beans to obtain the dimension features [1]. For example, area, perimeter and convex area, were converted into numerical features. The figures below showed the dimensions captured from the beans, as well as the distribution of the features.

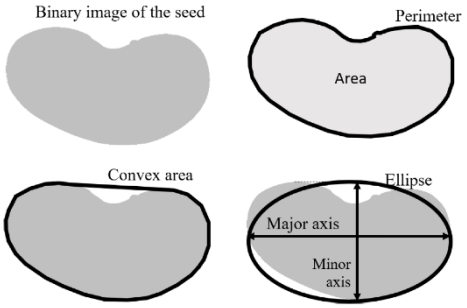

Fig. 2. Basic spatial features computed from the binary image by M. Dogan et al. [2]

No	Feature	Mean	Std	Min	25%	50%	75%	Max
1	Area	53048.28455	29324.09572	20420	36328	44652	61332	254616
2	Perimeter	855.283459	214.289696	524.736	703.5235	794.941	977.213	1985.37
3	MajorAxisLength	320.141867	85.694186	183.601165	253.303633	296.883367	376.495012	738.860154
4	MinorAxisLength	202.270714	44.970091	122.512653	175.84817	192.431733	217.031741	460.198497
5	AspectRation	1.583242	0.246678	1.024868	1.432307	1.551124	1.707109	2.430306
6	Eccentricity	0.750895	0.092002	0.218951	0.715928	0.764441	0.810466	0.911423
7	ConvexArea	53768.20021	29774.91582	20684	36714.5	45178	62294	263261
8	EquivDiameter	253.06422	59.17712	161.243764	215.068003	238.438026	279.446467	569.374358
9	Extent	0.749733	0.049086	0.555315	0.718634	0.759859	0.786851	0.866195
10	Solidity	0.987143	0.00466	0.919246	0.98567	0.988283	0.990013	0.994677
11	Roundness	0.873282	0.05952	0.489618	0.832096	0.883157	0.916869	0.990685
12	Compactness	0.799864	0.061713	0.640577	0.762469	0.801277	0.83427	0.987303
13	ShapeFactor1	0.006564	0.001128	0.002778	0.0059	0.006645	0.007271	0.010451
14	ShapeFactor2	0.001716	0.000596	0.000564	0.001154	0.001694	0.00217	0.003665
15	ShapeFactor3	0.64359	0.098996	0.410339	0.581359	0.642044	0.696006	0.974767
16	ShapeFactor4	0.995063	0.004366	0.947687	0.993703	0.996386	0.997883	0.999733

Table 2. Distribution of features

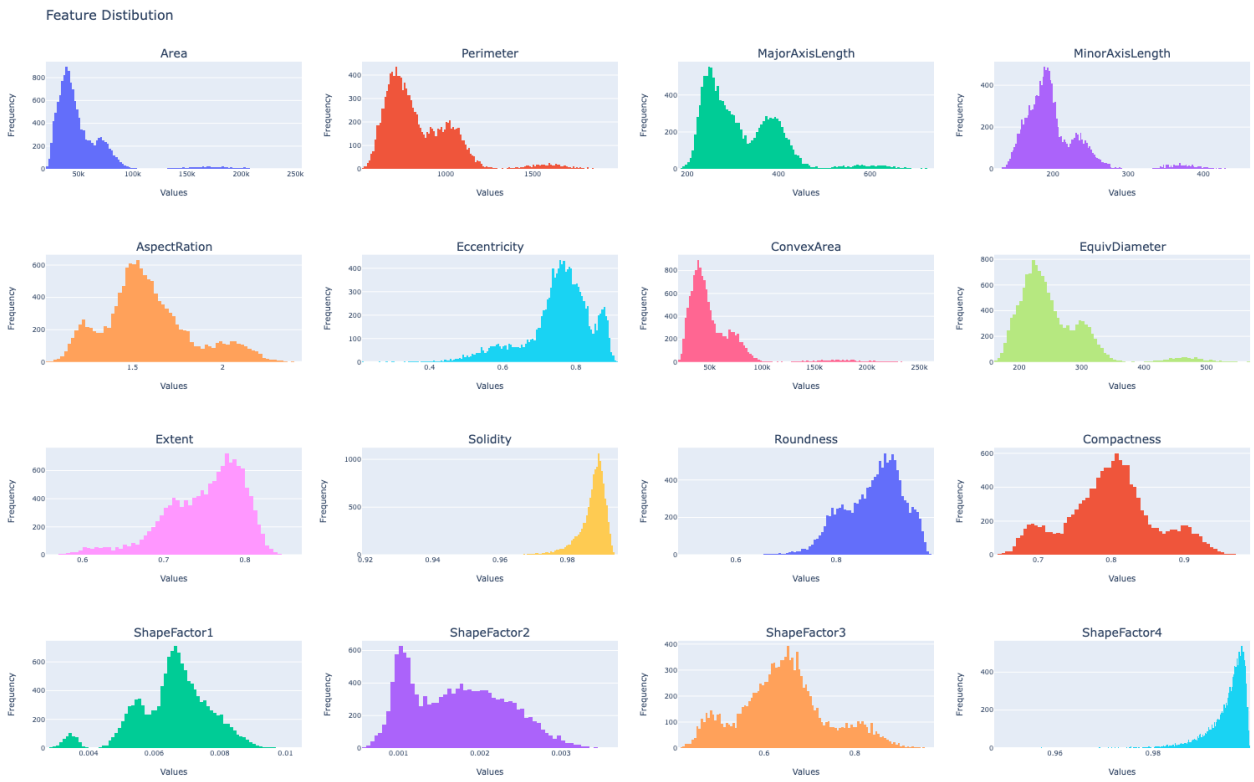

Fig. 3. Visualized distribution of features

4. Methodology

4.1 Data Preprocessing

The project aimed at exploring more on the data preprocessing on the dry bean data set. Various preprocessing techniques have been applied.

4.1.1. Feature Selection

From the previous research there were no feature selection techniques. Therefore, it is worth exploring the possibility of feature selection for enhancing the classification performance.

Correlation-based Feature Selection Subset Evaluation (CFSSubsetEval)

This algorithm was proposed by Hall [6] aimed to address the problem of feature selection. His research tried to study a new feature selection algorithm based on the correlation of features to the classes, by assuming that good features should be highly correlated to the classes. His research resulted in the CFSSubsetEval which was a correlation-based feature selector implemented in WEKA, a popular open-source software suite for machine learning and data mining. This algorithm will be applied to the data set for selecting features highly correlated to the classes using WEKA.

Information Gain

Information gain is the reduction in entropy produced from partitioning a set with a feature, which can be calculated by:

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i log_2 p_i$$

The calculated information gain will be between 0 and 1. The higher the value means the feature is more informative. It can be used as a metric to measure the quality of the feature. In this project the InfoGainAttributeEval from WEKA was used for evaluating the features. Features with low ratings were eliminated from the data set.

4.1.2. Normalization

Normalization is essential when applying different algorithms. It is because in a dataset, there will be a different data range for each attribute. Thus, to equalize the range for each attribute while keeping the data in the same scale, different normalization methods have been used [8]. For instance, Min-Max and Z-Score normalization.

Min-Max Normalization

Using Min-Max normalization, a linear transformation can be performed on the data using the following formula:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

For every feature, the minimum and maximum values will be transformed to 0 and 1 respectively and the rest will be a numeric value between 0 and 1. Thus, the comparisons between these data will then be balanced.

Z-Score Normalization

Similarly, the main ideas of Z-Score normalization are also to rescale the data in a comparable scale. The difference between these two normalization methods is that Z-Score will use the mean and standard deviation of the data to perform the normalization as shown in the formula.

$$Z = \frac{(x - \mu)}{\sigma}$$

Therefore, Z-Score handles the outliners better because it uses the standard deviation in the formula, so the individual values carry less weight. But for Min-Max normalization, the outliner itself should be the values used to normalize the data which is the highest or lowest in the dataset. Thus, normalization results in greater impact on the overall normalization rate.

4.1.3. Noise

The idea of introducing noise to the data set came from image processing. In image classification, one of the image processing techniques is to introduce some noise to the images to make the classifier more robust and prevent overfitting.

Gaussian Noise

In this project gaussian noise was introduced to the Z-score normalized data. The gaussian noise was with mean = 0 and standard deviation = 0.1, which was also known as 10% gaussian noise. The below figure shows the distribution of the noise introduced.

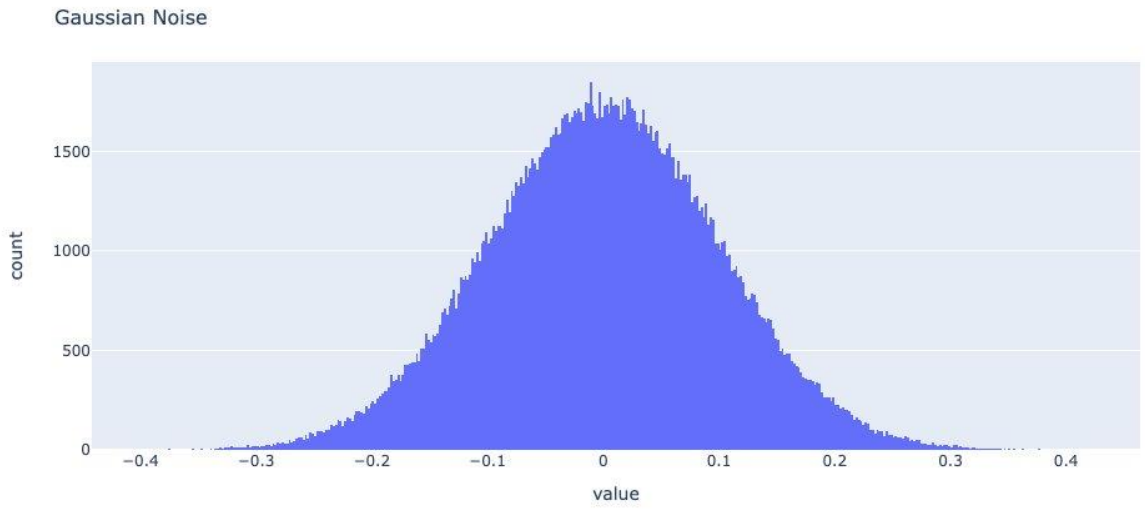

Fig. 4. Distribution of gaussian noise

4.1.4. Class Imbalance

As mentioned in the previous part, the dataset contains an imbalanced dataset which may affect the classifier's accuracy. Particularly for classes that have fewer samples, the classifier may not be able to efficiently learn the decision boundary. Additionally, the classifier may not be able to accurately classify all minority classes in the dataset.

Therefore, techniques like SMOTE will be explored in this project to solve the issue of imbalanced data.

Synthetic Minority Over-sampling Technique (SMOTE)

In SMOTE, the basic idea is to generate new samples for each minority class to solve the problem. By choosing class samples that are in the minority, SMOTE will select samples that are closer in their feature space to the data connecting its k nearest neighbors. New samples will then be generated between the two selected samples and this step will be repeated until the data imbalance is solved [9].

So, the decision region of the minority class can be expanded to the number of nearest neighbors chosen. Overall, this approach effectively enhances the generalizability of the decision region for each minority class.

However, there are some limitations when the SMOTE is used which the generated data may not accurately represent the data pattern. This is because SMOTE assumes that the data on minority classes are closely linked. If the data quality is poor, samples created may not be representative. Therefore, validations are important after SMOTE has been applied.

4.2. Classification

4.2.1. Classification Models

This section first describes the multiple common classifiers used to conduct the experiments. Including Random Forest, AdaBoost, k-Nearest Neighbor and Decision tree (CART). Parameters were passed to the classifier functions to facilitate analysis with a greater depth. Then, it describes the evaluation methods used to analyze the experiment results.

4.2.2. Random Forest Classifier

Random Forest is a widely used and adopted algorithm in machine learning, it can tackle classification and regression problems effectively. In random forest, "Bagging", also known as Bootstrap Aggregation, is used as its ensemble technique. The usage of Bagging means that many models, which are individual decision trees, are created from random samples taken from the dataset with replacements, with each tree being trained independently using its corresponding sample set. Then, the output of all the trees is combined via a majority voting process to form the final output, in which the most predicted outcome of all the trees is selected. Since the very specific rules in each individual tree could be removed in the aggregation process, the combined result would have less variance compared to a single tree, allowing the random forest to offer reduced overfitting and less sensitivity to noise, resulting in a higher predictive accuracy and reliability over regular decision tree algorithm. However, random forest costs higher computational resources meaning that it is best suited for complex tasks with high dimensional data while regular decision tree can be used to handle simple tasks.

Random Forest Experiment Implementation

The experiment was conducted using the "ensemble.RandomForestClassifer" function from the sklearn library, using the original, z-score normalization, min-max normalization, CFS and Info Gain data and Noise included data individually, as well as combination testing with the data processing techniques, is as follows: First, the dataset was split using an 80% to 20% train-test ratio. Since our dataset has many classes, a function that was implemented is called "OneVSRestClassifier" which would, for each class, fit it against all other classes.

The random forest classifier (referred to as RF) is configured to be executed multiple times using different number of individual trees: 10, 50, 100, 200 trees, which was achieved by passing the numbers as parameters in the "GridSearchCV" function, which takes the RF in as its estimator. This was then passed to the OneVSRestClassifer, which was then used to fit the previously generated training dataset. After that, the "predict" function of OneVSRestClassifier which would return the predicted multiclass targets, was called. In addition, the "predict_proba" function was called to get the probability estimate for all classes. Finally, the "classification_report" function was called to display the report and PRC, ROC functions were called to display their visualization respectively.

4.2.3. AdaBoost Classifier (Adaptive Boosting)

AdaBoost is one of the classifiers that had become quite popular for machine learning in recent years, it produces a strong classification model by training iteratively. In AdaBoost, the Boosting technique is used as the ensemble method, in which the process is as follows: First, a training subset is randomly selected from the dataset, and it is used to train the model using the AdaBoost model, the decision tree classifier is used by default. Then, higher weight would be assigned to the incorrectly classified results to facilitate a higher probability of them being used in classification of the next iteration. It also assigns higher weight to the classifier with the higher accuracy in each iteration. Then the Iterations would be repeated. The iteration process will end when a "perfect" fit occurs meaning the training data fits without inaccuracies, or when the maximum number of iterations is reached, this number is specified when preparing the AdaBoost classifier. Finally, after all the iteration has ended, a prediction can be obtained.

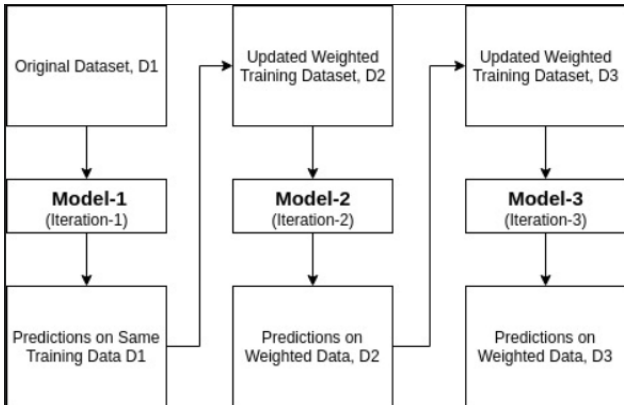

Fig. 5 Iterative process of AdaBoost [6]

AdaBoost Experiment Implementation

The experiment was conducted using the "ensemble.AdaBoostClassifer" function from sklearn using the original, z-score normalization, min-max normalization, CFS and Info Gain data and Noise included data individually, as well as combination testing with the data processing techniques, is as follows: First, an instance of AdaBoost classifier function was created. Then, much like the process in the Random Forest experiment, "OneVSRestClassifier" and "GridSearchCV" were used to support multi class processing and to facilitate finding the best result of en_stimators. The n_estimator parameter in this case is the maximum number of iterations the AdaBoost will run for and is at which point where the boosting is terminated, which the values passed in are 10, 20, 50 and 100 as maximum iterations. After setting up the AdaBoost with the 2 functions explained above, the "fit" function was called with the training data to start the AdaBoost process. Then, the "predict" and "predict_proba" functions were called with the testing data, which would again allow the code segment to print out a classification report in the notebook that visualize important information like precision and recall, as well as the best parameters for n_estimators, and PRC, ROC functions were called to display their visualization respectively.

4.2.4. k-nearest Neighbour Classifier (KNN)

The k-nearest Neighbors Classifier is another popular classifier used in machine learning, unlike Random Forest or AdaBoost, KNN uses proximity to group individual data points, it works under an assumption that similar points can be found near one another. KNN is simpler to use as it only needs the value ok k, which is the number of nearest neighbors needed for prediction, and the distance metric but Euclidean Distance is usually chosen for KNN. KNN calculates the individual distance between the target data point and the other data points, the distance will then be used to calculate which points are the target point's neighbors. Once KNN figured out which points are the target's neighbors, classification and regression can proceed. The figure below shows a simple process of how KNN determines neighbor points.

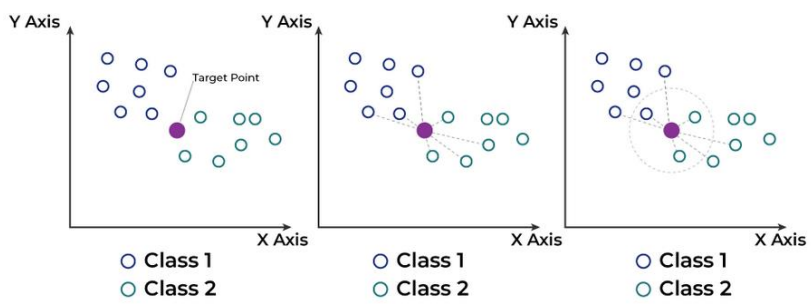

Fig. 6. KNN Neighboring Process [7]

In which the 1st step on the left shows a dataset before the neighboring process, then in the 2nd step in the center shows that KNN considers the distance between the target point (in purple) and the other data points. Finally in step 3 on the right, the 3 closest points within the circle are considered to be neighbors of the target data point. In a classification problem, majority voting is used to determine the class label of the target data point, meaning that the most common class among the neighbors would be the class of the target data point. In the case of the figure, the target would be "voted" as class 2. In a regression problem, an average of the nearest neighbors is calculated, determining the target's class label. While KNN is relatively simple and easy to use, it does have drawbacks such as high computation costs and time costs but most importantly it is prone to overfitting, which is reflected in the experiment.

KNN Experiment Implementation

The experiment was conducted using the "KNeighborClassifier" provided by sklearn, much like the Random Forest and AdaBoost experiments, "OneVSRestClassifier" and "GridSearchCV" are also used to facilitate multiclass processing. The KNN experiment was conducted using the original, z-score normalization, min-max normalization, CFS and Info Gain data and Noise included data individually, as well as combination testing with the data processing techniques. First the dataset was split by a 80% to 20% train-test ratio, then the train and test data were individually converted into NumPy, which can divide tasks into subtasks to process parallelly to speedup processing speed and lighten the load on memory since it uses less memory and storage than a python list. After that, the KNeighborClassifier function was set up using OneVSRestClassifier and GridSearchCV in the same fashion as previous experiments. Notably, parameter set named n_neighbors was passed in which represents the value of k, the values used were 2, 4, 8 and 16. Then the "fit" function was called with the training set to train the model. Finally, the code segment generated the classification report, and PRC, ROC functions were called to display their visualization respectively.

4.2.5. Decision Tree

CART is a variation of the Decision Tree algorithm that can handle both Classification and Regression. It includes classification tree and regression tree. Classification tree is used to determine the target's class label but does not work well against continuous target, where regression tree can be used instead. CART operates by building nodes and branches which represent class labels and decision points respectively. In classification, CART uses the Gini Impurity to determine splits, with lower Gini Impurity representing higher purity of the subset. CART also utilizes the pruning technique to avoid overfitting which is the processing of removing nodes that matter or contribute little to improving the model's accuracy. One popular pruning technique is information gain pruning which calculates Info Gain for each node and removing the nodes with low gain.

Decision Tree Experiment Implementation

The experiment was conducted using the "DecisionTreeClassifier" provided by sklearn, which uses CART algorithm by default. Again, "OneVSRestClassifier" and "GridSearchCV" are also used to facilitate multiclass processing. Data would once again include the original, z-score normalization, min-max normalization, CFS and Info Gain data and Noise included data individually, as well as combination testing with the data processing techniques. The data was split with an 80% to 20% train test ratio, with the training and testing sets converted to NumPy to lower the cost of computational power and storage. The setup for the decision tree classifier function is relatively simple, notably, Gini Impurity and Entropy were used as the criterion. The training data was then passed to the "fit" function and finally the code segment generated the classification report, and PRC, ROC functions were called to display their visualization respectively.

4.3. Performance Evaluation

The performance of a classifier can be evaluated by reviewing its ability to perform correct predictions. This can be evaluated using the rate of correct predictions made among all predictions which the following was used to achieve such purpose.

The Precision Recall Curve (PRC) and Receiver Operating Characteristics (ROC) are visualization methods that shows important values from the results like true positive and true negative rates which would facilitate the evaluation of a classifier's efficiency. Each class is given a unique color in the PRC and ROC for a clear view.

4.3.1 Precision Recall Curve

The following outcomes of the PRC, which is this research's focus, indicate different results:

- high precision and high recall.
    - all predictions are correct.
- high recall & low precision
    - many results
    - but most of its predicted results are incorrect.
- high precision & low recall
    - few results
    - but most of its predicted results are correct.

In which P is precision and R is recall:

$$P = \frac{T_p}{T_p + F_p} \quad R = \frac{T_p}{T_p + F_n}$$

4.3.2 Receiver Operating Characteristic

The outcome of the ROC would indicate performance, since classifier performs better when it is close to the point (0,1) in the ROC, because it means there are no false positives, and all the predictions are true positives.

In the ROC,

- X-axis = False Positive Rate
- Y-axis = True Positive Rate

And TPR (True Positive Rate) and FPR (False Positive Rate) is:

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN}$$

5. Experiment Results

5.1. Preprocessing

Different data preprocessing methods were mentioned to apply on the dry bean data set. In the following, the experiment results of each method will be shown.

5.1.1 Feature Selection

For both feature selection by CFS and infogain were done by WEKA. The CFSSubsetEval from WEKA selected 11 attributes out of 16. For InfoGainAttributeEval features with rank score more than 1 were selected.

CFSSubsetEval		**InfoGainAttributeEval**		
No	**Feature**	**Ranked**	**No**	**Feature**
2	Perimeter	1.524	2	Perimeter
3	MajorAxisLength	1.49	7	ConvexArea
4	MinorAxisLength	1.484	1	Area
5	AspectRation	1.484	8	EquivDiameter
7	ConvexArea	1.437	3	MajorAxisLength
9	Extent	1.376	14	ShapeFactor2
11	Roundness	1.329	13	ShapeFactor1
12	Compactness	1.328	4	MinorAxisLength
13	ShapeFactor1	1.192	15	ShapeFactor3
14	ShapeFactor2	1.192	12	Compactness
16	ShapeFactor4	1.175	5	AspectRation
		1.175	6	Eccentricity
		1.147	11	Roundness
		0.533	16	ShapeFactor4 (eliminated)

0.34	10	Solidity (eliminated)
0.284	9	Extent (eliminated)

Table 3. Result of feature selection

5.1.2 Normalization

In the original data set, the attributes are in different scales. Therefore, due to different scales of the attributes, the effectiveness and importance of the numbers will also be different.

	Area	Perimeter	MajorAxisLength	MinorAxisLength
0	28395	610.291	208.178117	173.888747
1	28734	638.018	200.524796	182.734419
2	29380	624.110	212.826130	175.931143

Fig. 7. Example of attributes in data set that are in different scales

Therefore, normalizations will be performed on the data set to improve the performance of different algorithms by scaling the input features to a common scale without changing the distribution of the data.

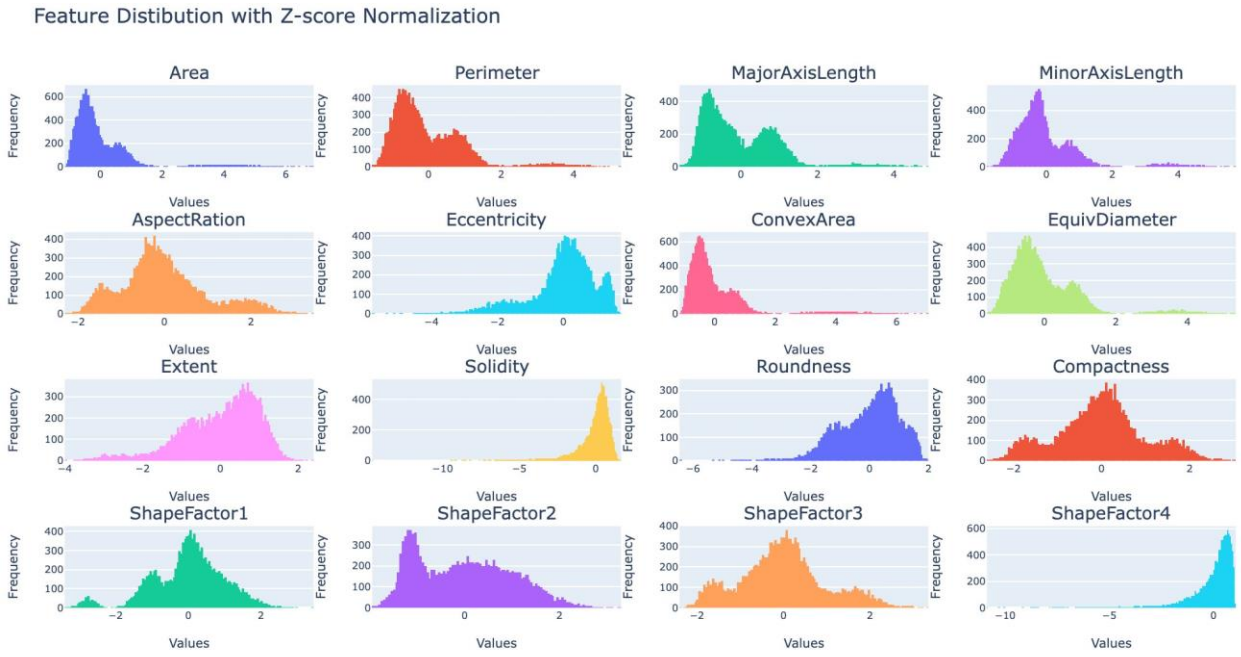

Fig. 8. Visualized distribution of features with Z-Score Normalization

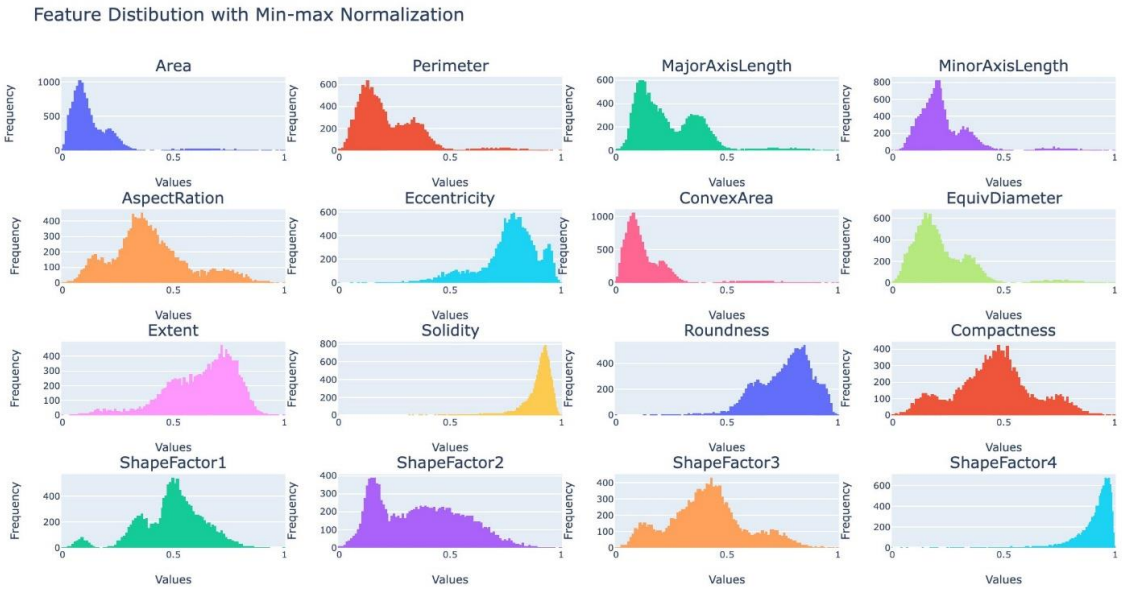

Fig. 9. Visualized distribution of features with Min-Max Normalization

After the Z-score and Min-Max normalization were performed, it is clear to observe that the distribution of the data remains unchanged. The scale of the numbers which is the X-axis of the graph is the only differences of the original data with the normalized one.

5.1.3 Noise

Similarly, after adding 10% gaussian noise to the original data set, the distribution of the data would remain unchanged.

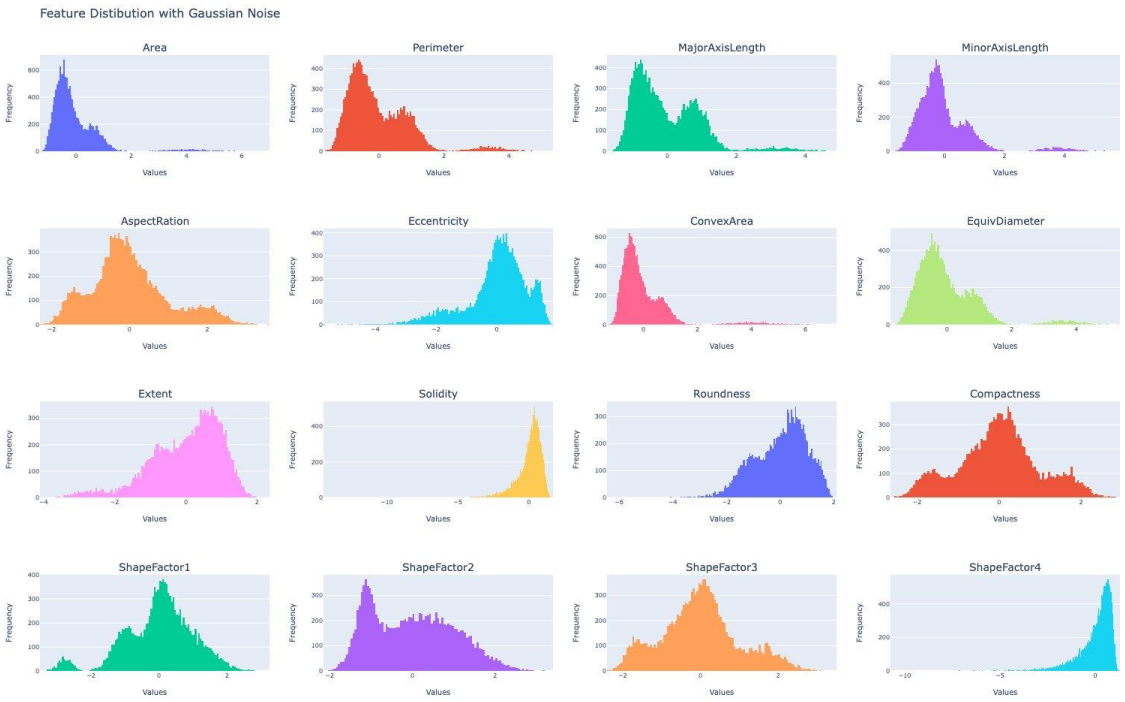

Fig. 10. Visualized distribution of features with Gaussian Noise

5.1.4 Class Imbalance

Originally, the data set is imbalanced that the class “Dermason” has about 3500 samples. Meanwhile, for the class “Bombay”, there are only 500 samples. After performing the SMOTE method, new samples were generated. From figure 10, it is obvious that the class distributions are now even.

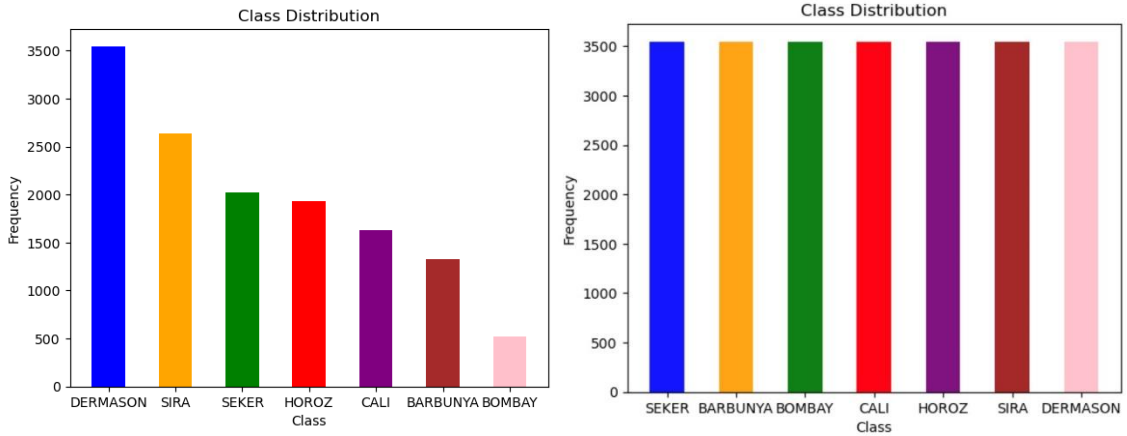

Fig. 10. Comparison of class distribution of the original data set (left) and data set with SMOTE (right)

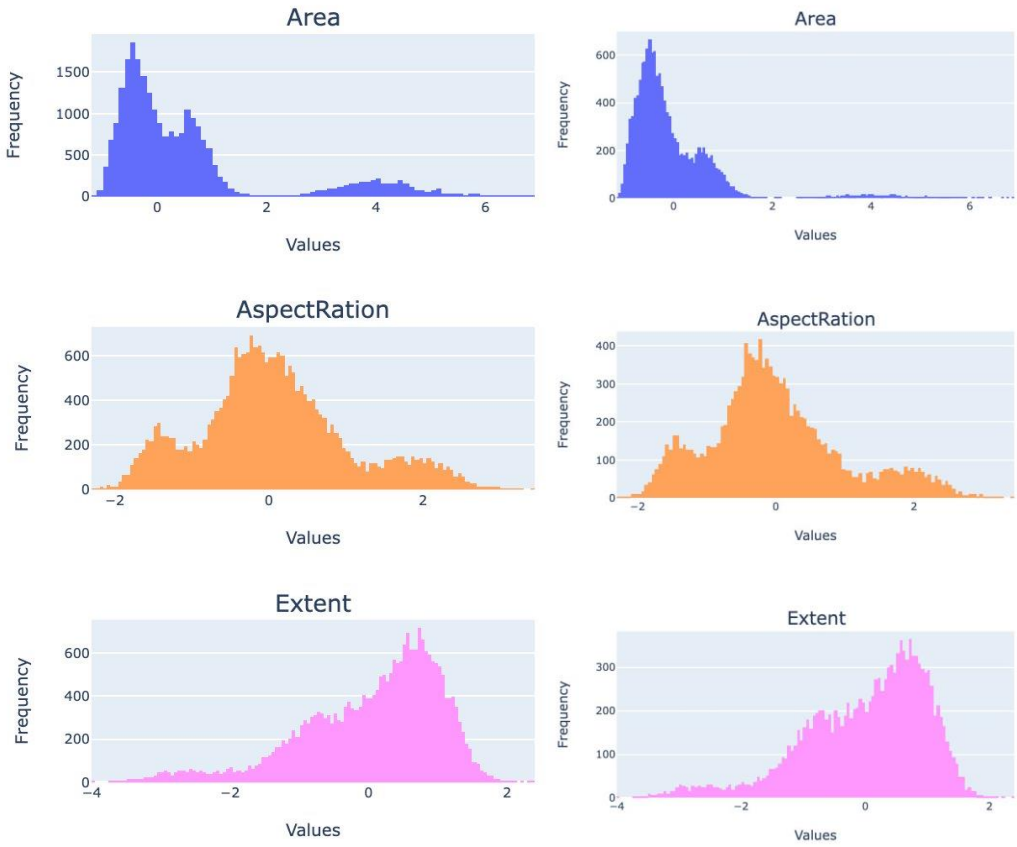

Fig. 11. Comparison of feature distributions of the data set with Z-score Normalization (left) and Z-Score Normalization and SMOTE with training data (right)

5.2. Cross-validation and Fine tuning

The following section aims to find out the best parameters of different classifiers through cross validation and different fine tuning methods.

5.2.1 Cross Validation

Validations are important as the performance of the classifiers handling the unseen data needs to be estimated. Thus, the true performance of the classifiers can then be estimated. A way to accomplish this is to allow the data set itself to indicate it by using cross validation and data spliting.

The first approch used in the project is cross validation, which divide the data into different number of folds. Then, most of the folds will be used for training, but one of the folds data wil be used as validation [10]. Thus, it helps to identify potential issues like overfitting and able to provides a robust estimate of the model's performance. For example, the cross validation accuracy can be used to evaluate the value of K in the kNN algorithms automatically using loop. In the project, the numbers of folds were set to 10.

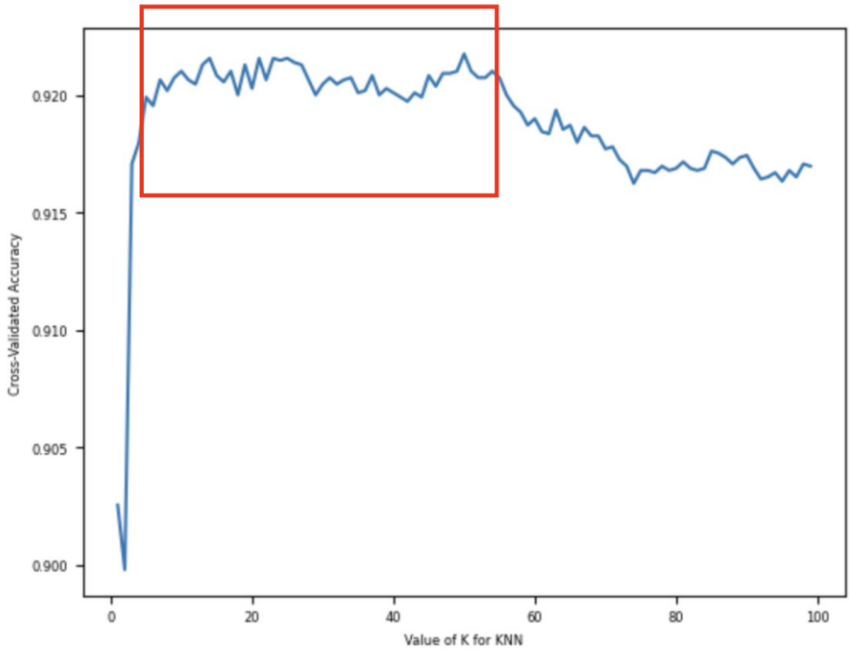

Fig. 12. Visualized idea of how to fine tuning by cross validation accuracy

Similarly, for the train and test split method, it splits the dataset into training and testing data so as to simulate the unseen data. In the project, the ratio of training and testing data is 8:2.

5.2.2 Fine Tuning

Meanwhile, fine-tuning referring to optimizing a model by adjusting its parameters or hyperparameters so that its performance can be improved. After fine-tuning, the model should be able to have better accuracy and efficiency.

To perform fine-tuning, the GridSearchCV function in sklearn was used which combines grid search and cross-validation techniques. This function is designed to search for the a parameter combinations that yield the highest cross-validation accuracy. To conduct the grid search, a dictionary containing the parameter names and the values to be explored is created. Figure 13 provides an example of such a dictionary used for performing a grid search for the kNN classifier.

```
{"n_neighbors": [2,4,8,16]},
```

Fig. 13. Dictionary used for performing a grid search for the kNN classifier

Besides, One-vs-Rest Classifier were also used in the project which is a common approach used in multi-class classification problems. In this approach, instead of directly classifying to multiple classes, a binary classifier was trained for each class. Therefore, in each classifier, one class will be considered as the positive class and all the other classes as the negative class. In short, the binary classifier is trained to distinguish instances of that class from all other classes. This approach actually allows us to use the binary classifiers to handle multi-class classification problems without modifying them.

Finally, after the fine tuning, the best parameter can be obtained through sklearn instance as shown in figure 14. Threfore, this can ensure the classifier performs good without overfitting or underfitting.

```
=== Best Parameter ===
Classifier for class BARBUNYA: {'n_estimators': 100}
Classifier for class BOMBAY: {'n_estimators': 10}
Classifier for class CALI: {'n_estimators': 200}
Classifier for class DERMASON: {'n_estimators': 200}
Classifier for class HOROZ: {'n_estimators': 50}
Classifier for class SEKER: {'n_estimators': 50}
Classifier for class SIRA: {'n_estimators': 200}
```

Fig. 14. The best parameter obtained by grid search for the kNN classifier

5.3. Classifiers Experiments Results

The following section explores the best performing data both individually and within combination testing. From the results, the effectiveness and notable improvements of utilizing data processing techniques are proven and observable. Most importantly, the combination of SMOTE + Z-Score Normalization + Feature Selection surpassed all the results of the individual data testing and other results of combination testing. The full results of all the experiments (including classification report, PRC and ROC) and coding can be found within the notebook.

5.3.1 Random Forest Results

Based on the PRC of the random forest experiment, the individual data that performed the best was the one with Z-score normalization. With a performance of 0.9804, all other individual data sets performed with 0.96 to 0.97.

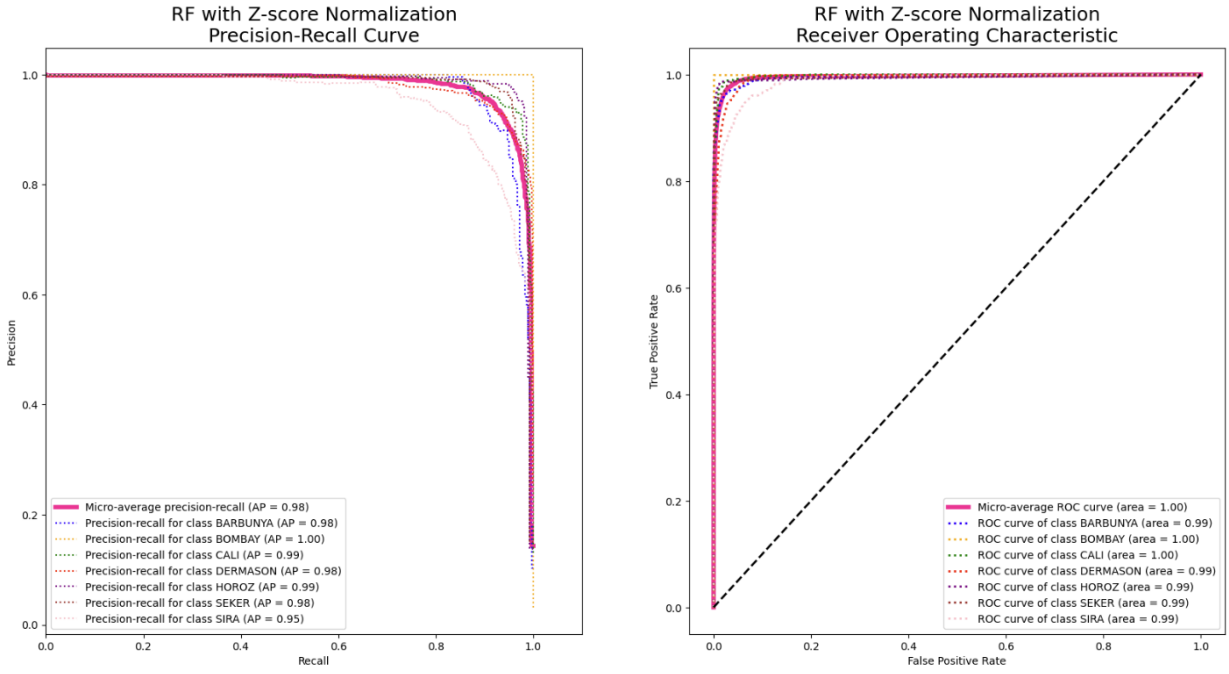

Fig. 15. Graph of PRC and ROC of the random forest classifer with Z-score normalization

However, z-score data is surpassed by one of the combination testing results. The best performance of random forest was observed in the SMOTE + Z-Score + Feature Selection data, with a 0.9893 performance, surpassing the individual Z-Score data set.

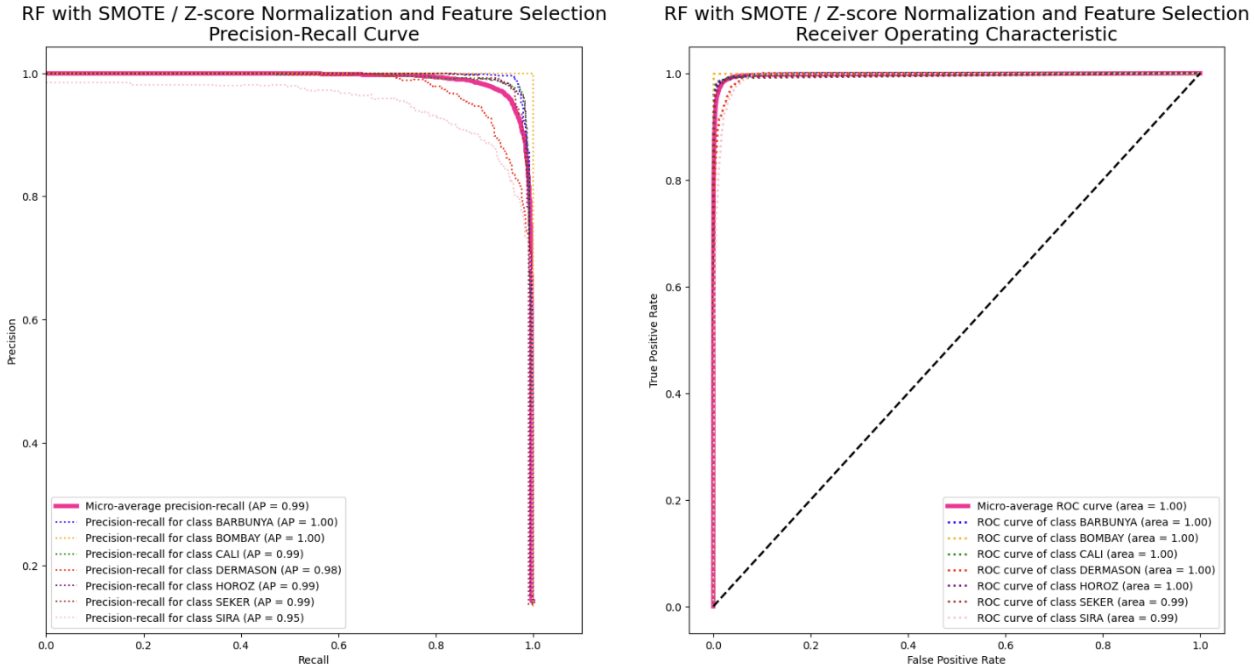

Fig. 16. Graph of PRC and ROC of the random forest classifer with SMOTE, Z-Score normalization and feature selection

The following summerizes the test results of the random forest experiment.

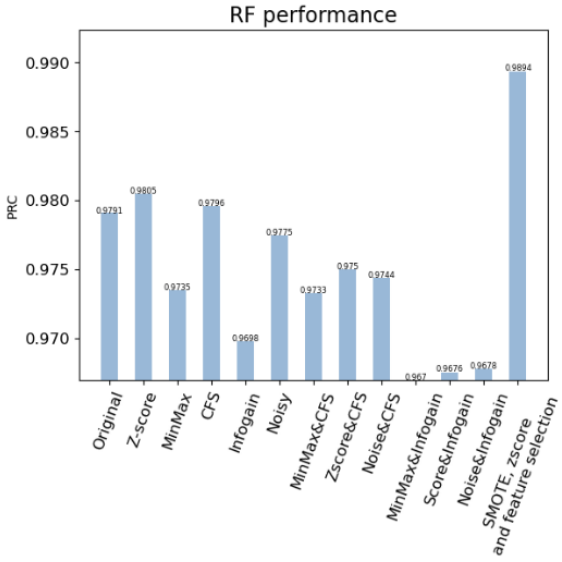

Fig. 17. Performance of random forest under different data preprocessing methods

5.3.2 AdaBoost Results

Based on the PRC of the AdaBoost experiment, the individual data that performed the best was the one with Info Gain. With a performance of 0.9780, all other individual data sets performed with 0.96 to 0.97.

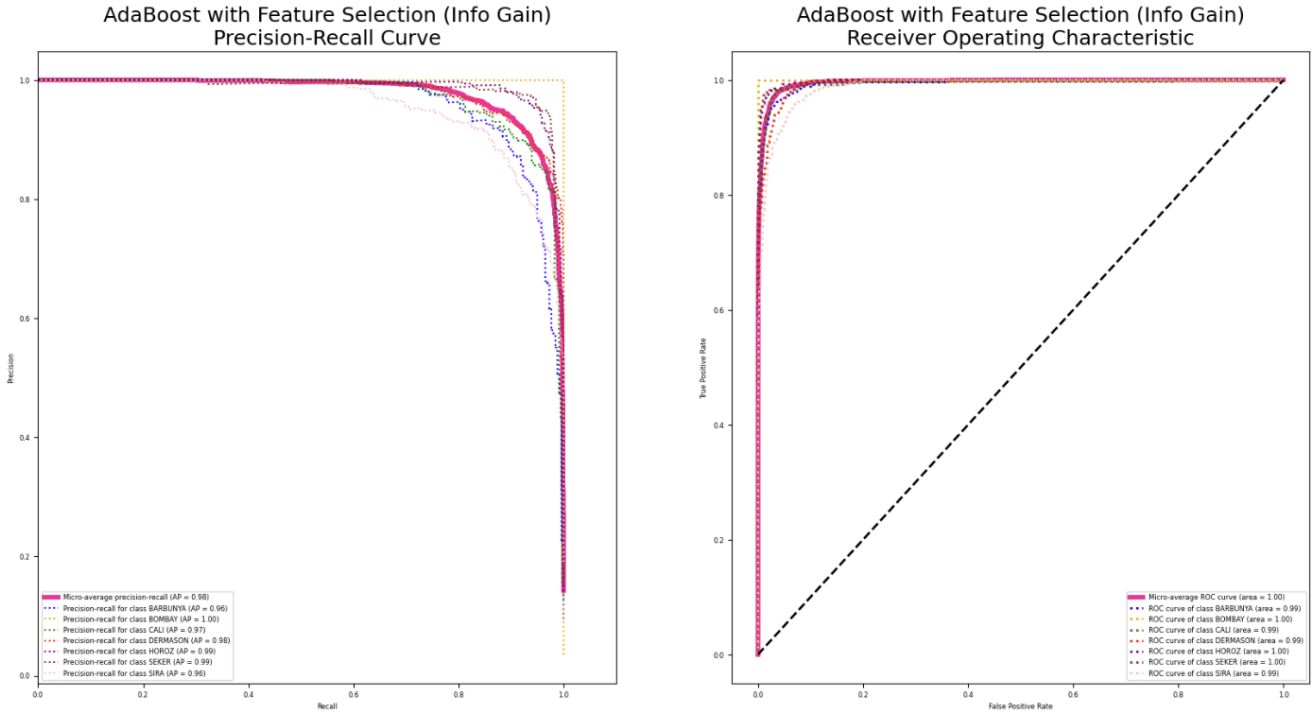

Fig. 18. Graph of PRC and ROC of the AdaBoost with feature selection

However, Info Gain data is surpassed by one of the combination testing results. The best performance of AdaBoost was observed in the SMOTE + Z-Score + Feature Selection data, with a 0.9861 performance, surpassing the individual Info Gain data set, and is only 0.0032 behind that of random forests.

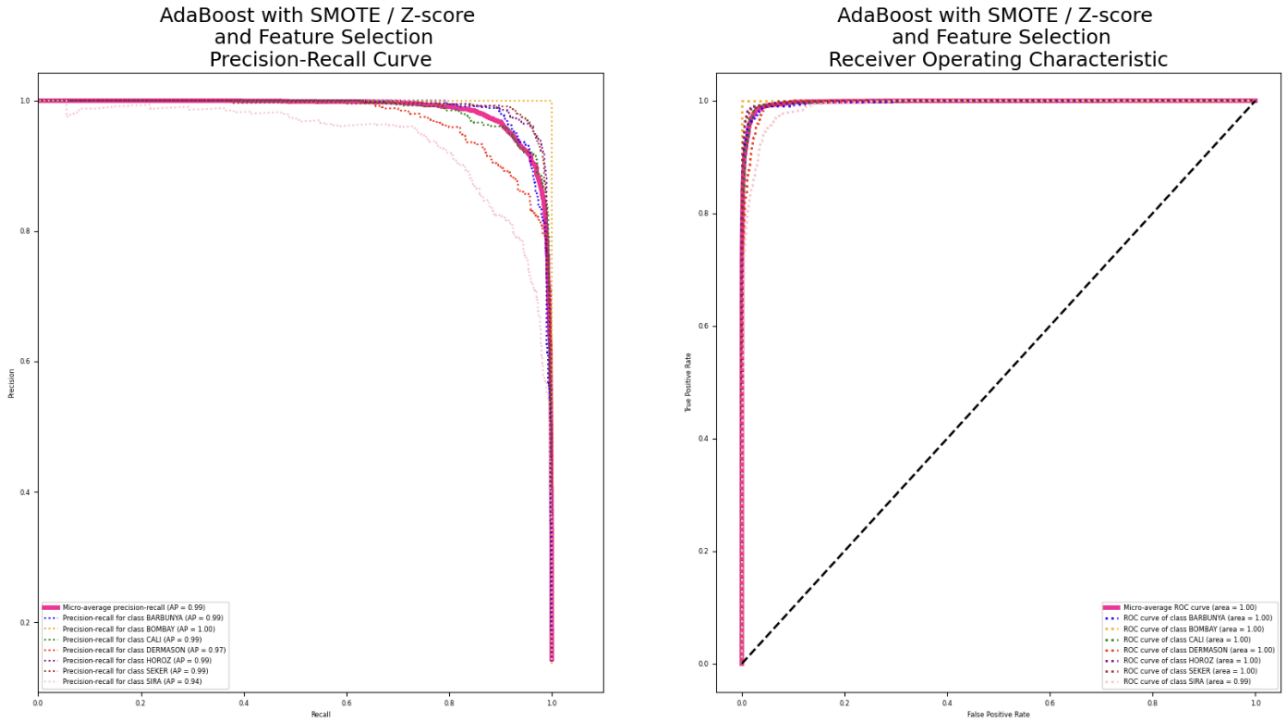

Fig. 19. Graph of PRC and ROC of the AdaBoost with SMOTE, Z-score normalization and feature selection

The following summerizes the test results of the AdaBoost experiment.

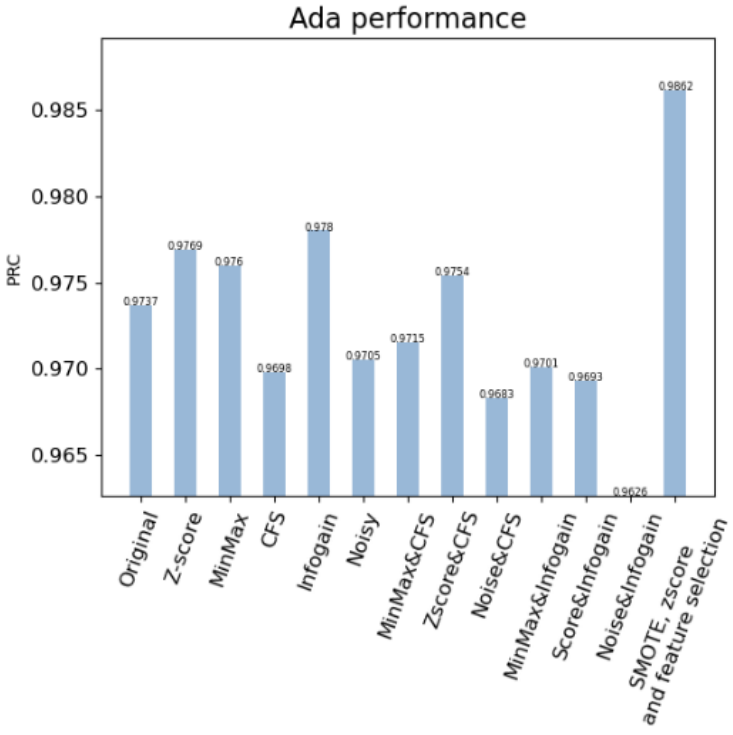

Fig. 20. Performance of AdaBoost under different data preprocessing methods

5.3.3 KNN Results

Based on the PRC of the AdaBoost experiment, the individual data that performed the best was the one with CFS. With a performance of 0.9700, all other individual data sets performed with 0.73 to 0.96.

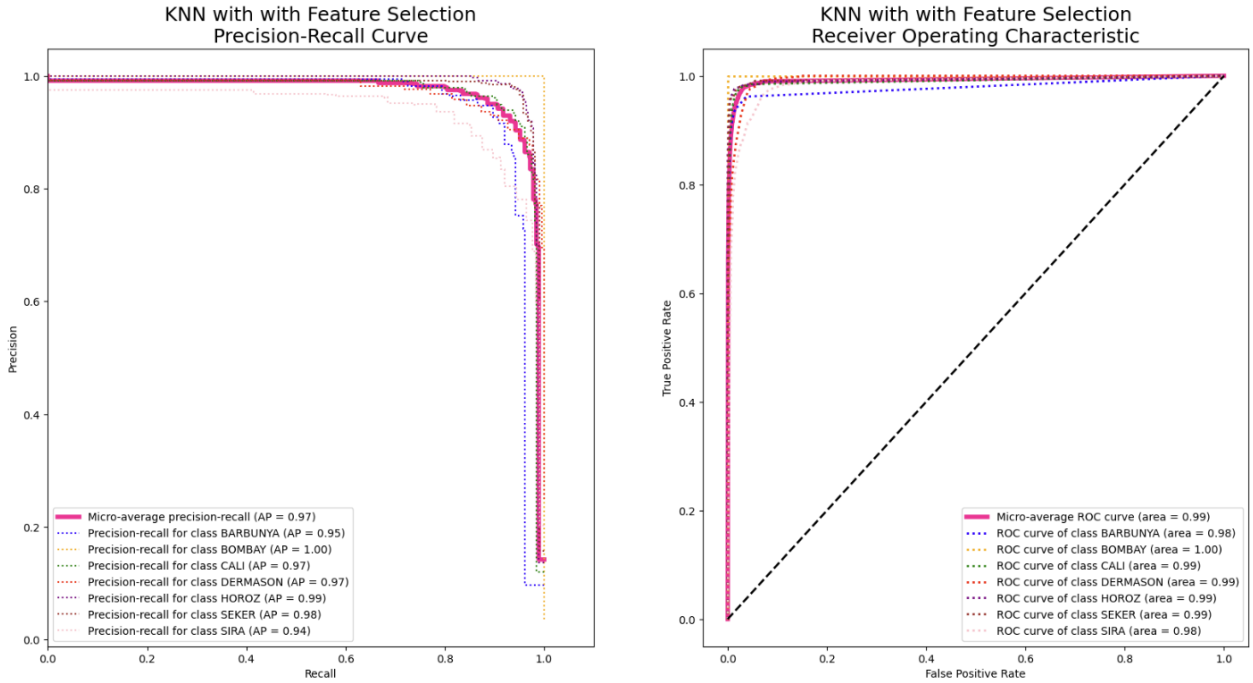

Fig. 21. Graph of PRC and ROC of the kNN with feature selection

Notabally, the original dataset with KNN performed the worst out of all the tests in entire experiment, with only 0.73, reflecting the risk of overfitting without data preprocessing which is also present in other classifiers although not to such a great extent.

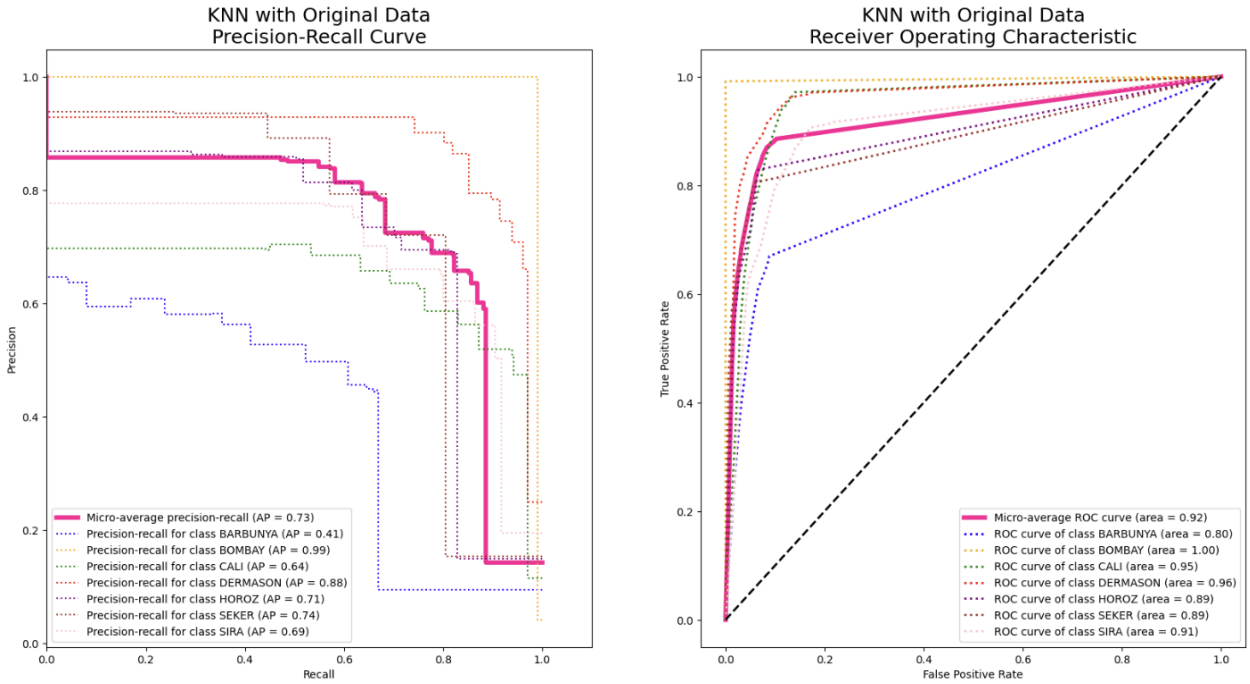

Fig. 22. Graph of PRC and ROC of the kNN original data

However, CFS data is surpassed by one of the combination testing results. The best performance of KNN was observed in the SMOTE + Z-Score + Feature Selection data, with a 0.9861 performance, surpassing the individual CFS data set.

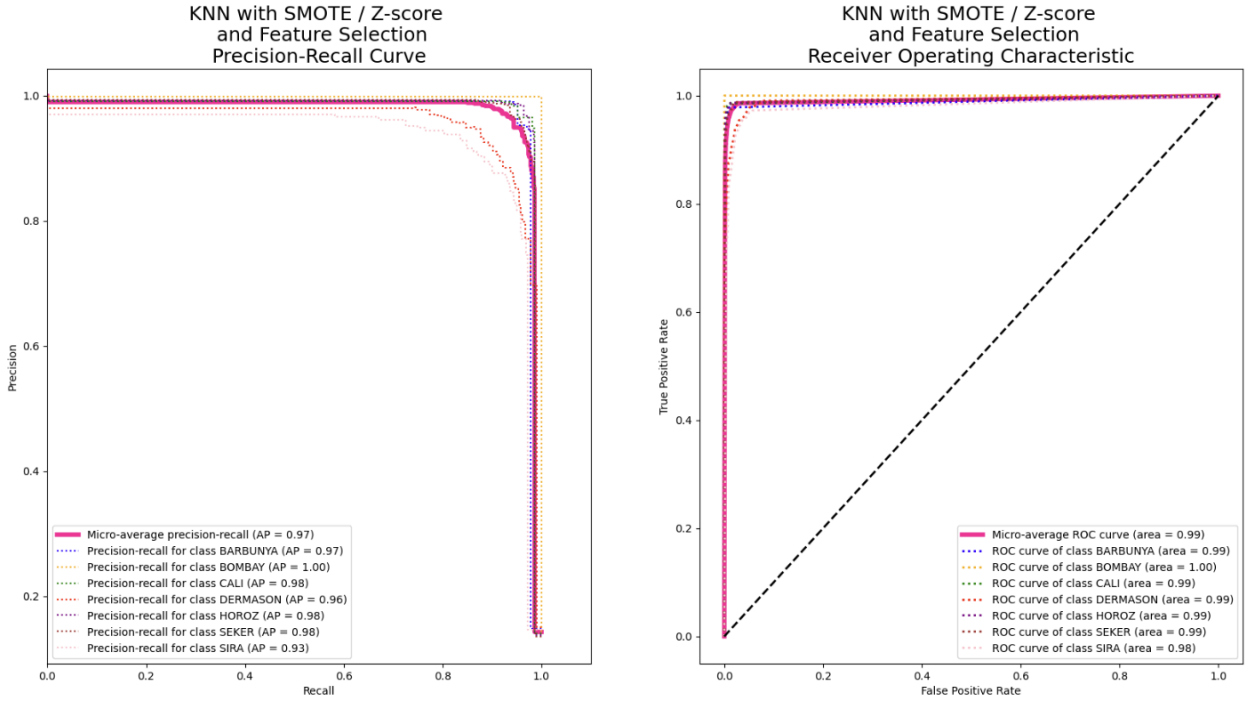

Fig. 23. Graph of PRC and ROC of the kNN with SMOTE, Z-score normalization and feature selection

The following summerizes the test results of the KNNexperiment.

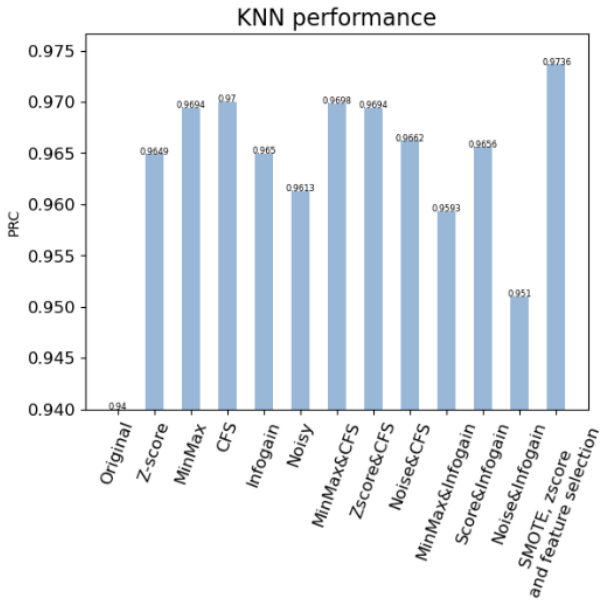

Fig. 24. Performance of kNN under different data preprocessing methods

5.3.4 Decision Tree Results

Based on the PRC of the AdaBoost experiment, the individual data that performed the best was the one with Info Gain, like that of AdaBoosts. With a performance of 0.9682, all other individual data sets performed with 0.93 to 0.961.

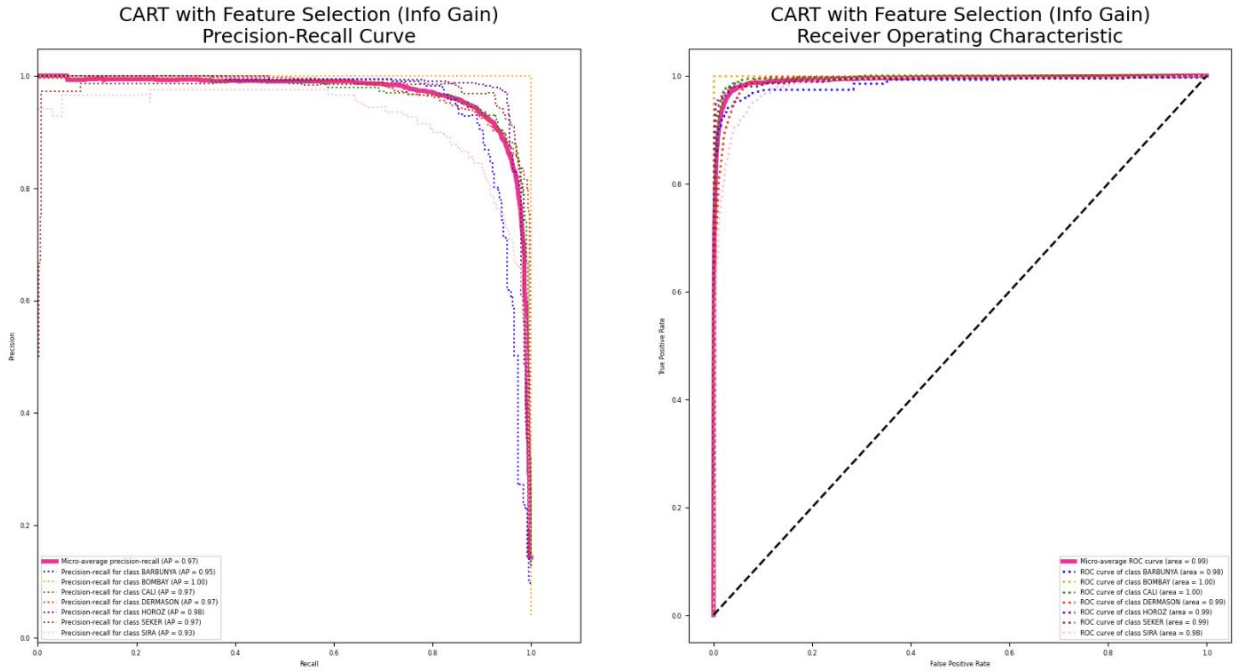

Fig. 25. Graph of PRC and ROC of the CART with feature selection

However, Info Gain data is surpassed by one of the combination testing results. The best performance of Decision Tree was observed in the SMOTE + Z-Score + Feature Selection data, with a 0.9721 performance, surpassing the individual Info Gain data set.

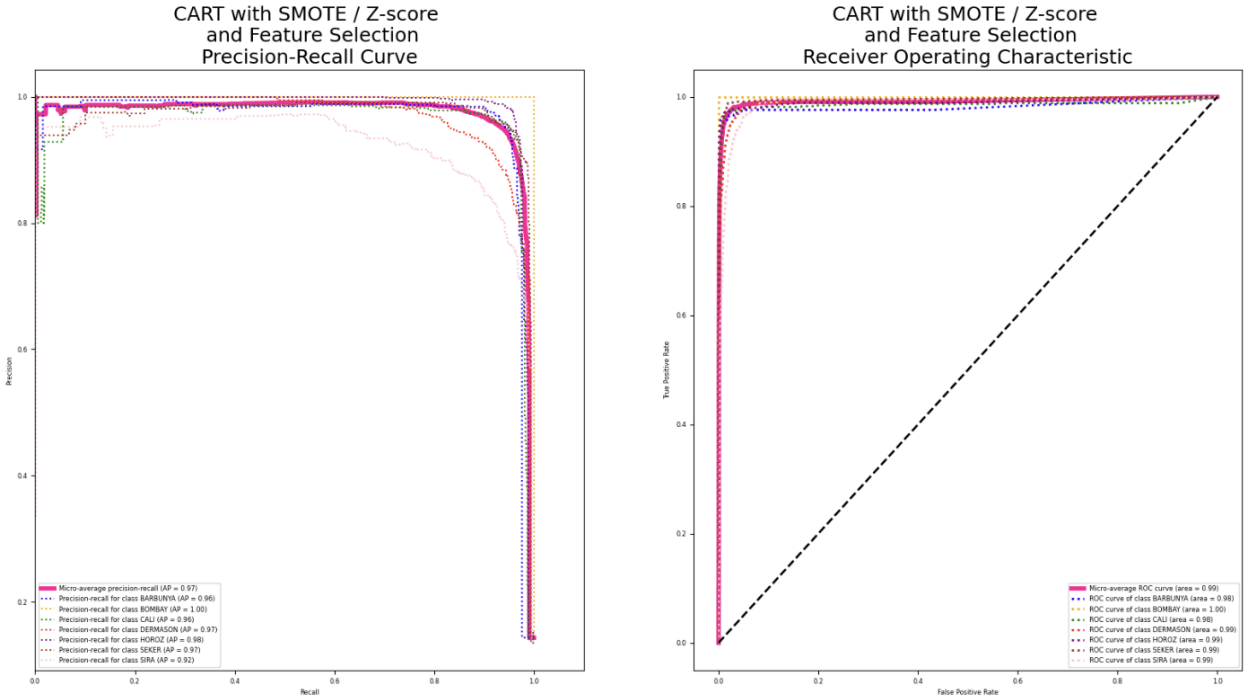

Fig. 26. Graph of PRC and ROC of the CART with SMOTE, Z-score normalization and feature selection

The following summerizes the test results of the Decision Tree(CART) experiment.

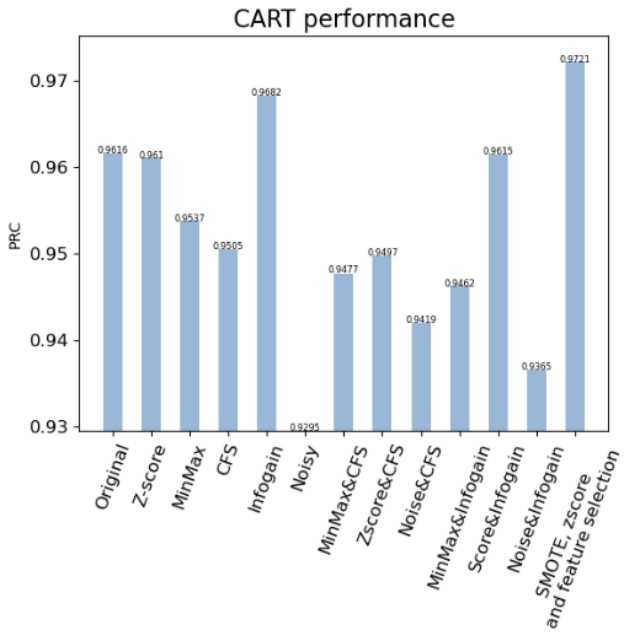

Fig. 27. Performance of CART under different data preprocessing methods

6. Discussion

The below table shows a summarized result of the experiments. The purple color highlighted the best performance of that classifier without SMOTE, while the green color highlighted the performance of that classifier with SMOTE.

	PRC AUC			
	RF	**AdaBoost**	**KNN**	**CART**
Original	0.979144699	0.973678676	0.73434203	0.961590026
Z-score	**0.98048767**	0.976913639	0.964942969	0.961022414
Min-max	0.97350926	0.975952883	0.969369673	0.953725505
CFS	0.979585829	0.969781191	**0.970041488**	0.950480018
Infogain	0.969800689	**0.978043618**	0.964988369	**0.968217017**
Noisy	0.977489563	0.970487715	0.961325432	0.929492055
Min-max & CFS	0.973315627	0.971533892	0.96984866	0.947659283
Z-score & CFS	0.975020048	0.975377604	0.969417381	0.94973238
Noise & CFS	0.974383935	0.968319736	0.966193701	0.941901072
Min-max & Infogain	0.967004399	0.970086957	0.959338112	0.946196496
Score & Infogain	0.967557197	0.969273899	0.965619545	0.961483048
Noise & Infogain	0.967832047	0.962632966	0.950987514	0.936518344
SMOTE, Z-score and feature selection	**0.989352833**	**0.986172399**	**0.973625952**	**0.972122992**

Table 4. Summary of experiment results

6.1. Normalization Performance

For most of the classifier's normalization gave an enhancement to the performance. The Z-score enhanced the RF performance to 0.980 and Adaboost to 0.977, and gradually increased the performance of KNN by 23.2% compared to that using the original data. In general Z-score performed better than the Min-max, as RF, Adaboost and KNN with Z-score were with higher PRC area under curve.

6.2. Feature Selection Performance

Feature selection demonstrated effective performance across various classification tasks. Without SMOTE resolving the class imbalance problem, AdaBoost, KNN and CART classifiers gave the best performance when using feature selection techniques. AdaBoost and CART classifiers performed best when using Infogain, and KNN performed best when using CFS. Classification using RF with CFS also ranked 2nd among the RF results without SMOTE.

6.3. Noise Performance

The experiment using gaussian noise to the classification task did not meet our expectation. All the classifiers the performance dropped when the 10% gaussian noise was introduced to the Z-score normalized dataset. Although it is expected that the noisy data might lead to a better generalization and prevent overfitting, from the result it simply confused the classifiers.

6.4. SMOTE Performance

In general classification with SMOTE performed the best. Our experiment used Z-score normalization, CFS feature selection and SMOTE which the PRC AUC were higher than the other experiments by 1% using

the same classifier. This indicates class imbalance was an important factor towards classification, and SMOTE had solved the problem effectively by oversampling.

6.5. Best Classifier

From our experiment the best classification was done by using Z-score normalization, CFS feature selection and SMOTE in data preprocessing, then using RF for classification.

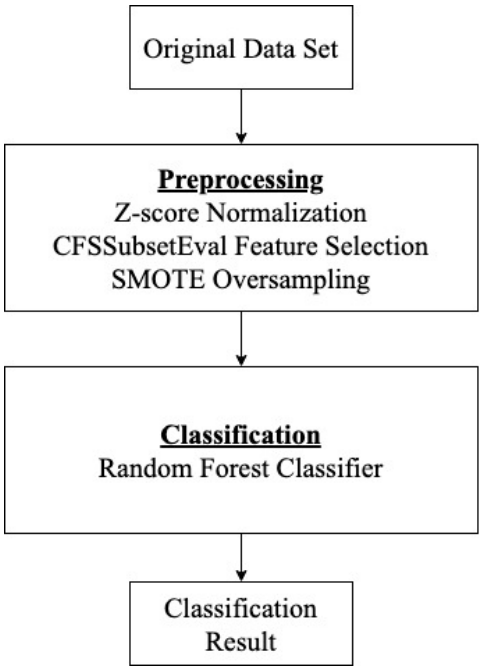

Fig. 28. Pipeline of the best classification result

6.6. Evaluation

Our best classification result was then compared to the best result done by J. C. Macuácua et al. [4] using original data, optimized hyperparameters, SMOTE balancing and KNN classifier.

Our Experiment		**J. C. Macuácua et al.**	
Class	**Precision**	**Class**	**Precision**
BARBUNYA	0.98	BARBUNYA	0.987
BOMBAY	1	BOMBAY	1
CALI	0.98	CALI	0.983
DERMASON	**0.91**	DERMASON	0.889
HOROZ	**0.98**	HOROZ	0.976
SEKER	**0.98**	SEKER	0.978
SIRA	**0.91**	SIRA	0.908

Table 5. Comparison of experiment results

The above table compared the two results by class precision. It showed that our experiment has a slightly improvement, for there were 4 classes out of 7 with higher precision than that done by J. C. Macuácua et al. In their research they examined the PCA instead of normalization, while the best result was acquired by using the original data. This could be one of the explanations our experiment was having a better result.

7. Conclusion

The project continued the research on the dry bean classification by exploring the data preprocessing. It was aimed at enhancing the performance of classification, which was successful with the best classifier using Z-score normalization, CFS feature selection, SMOTE and RF together. The results showed improvement compared to previous research.

8. References

[1] M. Koklu and I. A. Ozkan, "Multiclass classification of dry beans using computer vision and Machine Learning Techniques," *Computers and Electronics in Agriculture*, vol. 174, p. 105507, Jul. 2020. doi:10.1016/j.compag.2020.105507

[2] M. Dogan *et al.*, "Dry bean cultivars classification using Deep CNN features and Salp Swarm algorithm based Extreme Learning Machine," *Computers and Electronics in Agriculture*, vol. 204, p. 107575, Jan. 2023. doi:10.1016/j.compag.2022.107575

[3] M. Salauddin Khan *et al.*, "Comparison of multiclass classification techniques using dry bean dataset," *International Journal of Cognitive Computing in Engineering*, vol. 4, pp. 6–20, Jun. 2023. doi:10.1016/j.ijcce.2023.01.002

[4] J. C. Macuácua, J. A. Centeno, and C. Amisse, "Data mining approach for dry bean seeds classification," *Smart Agricultural Technology*, vol. 5, p. 100240, Oct. 2023. doi:10.1016/j.atech.2023.100240

[5] N. Pourmoradi, "Dry Bean Dataset Classification," Kaggle, https://www.kaggle.com/datasets/nimapourmoradi/dry-bean-dataset-classification/data (accessed Apr. 21, 2024).

[6] "AdaBoost Classifier Algorithms using Python Sklearn Tutorial," *www.datacamp.com*. https://www.datacamp.com/tutorial/adaboost-classifier-python

[7] GeeksforGeeks, "K-Nearest Neighbours - GeeksforGeeks," *GeeksforGeeks*, Nov. 13, 2018. https://www.geeksforgeeks.org/k-nearest-neighbours/

[8] Henderi, H. (2021) 'Comparison of min-max normalization and Z-score normalization in the K-Nearest Neighbor (knn) algorithm to test the accuracy of types of breast cancer', *IJIIS: International Journal of Informatics and Information Systems*, 4(1), pp. 13–20. doi:10.47738/ijiis.v4i1.73.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, Jun. 2002. doi:10.1613/jair.953

[10] Schaffer, C. (1993) 'Selecting a classification method by cross-validation', Machine Learning, 13(1), pp. 135–143. doi:10.1007/bf00993106.

9. Appendix

9.1. Data Set

https://www.kaggle.com/datasets/nimapourmoradi/dry-bean-dataset-classification/data

9.2. Git Repository

https://github.com/tamhofung/dry-bean-classification