UNSW COMPUTING

UNSW DATA9001
Assignment 3

Due: Friday 8nd of August 18:00 Sydney time

Intro

This assignment builds on and extends skills and knowledge acquired in the Computer Science part of DATA9001 (week 8 and 9). Please read each question carefully, answer all of them, report results in a way that is easy to follow. Email any questions or comments to Hao Xue (hao.xue1@unsw.edu.au)

Note:

- Total value: 15 marks (15% of the final mark). Submission is due on **Friday of Week 10 (8 August), 6:00 pm**.
- Please submit your answers to assignment questions in PDF format using the following name: **`A3-z1234567-FirstName-Surname-Report.pdf`**
- Please also submit the Python code that you developed as part of the assignment. The PDF report and code should be submitted in **`A3-z1234567-FirstName-LastName.zip`** file.
- The submission link is on the Moodle site under Assessments Hub.
- **Please sign the Plagiarism Declaration at the bottom of this page. Assignments without signed Plagiarism Declaration will not be accepted.**
- Format: 14 pages max. Do not include a separate title page. At least 11-point font should be used, with adequate margins for comments. Any extra pages will not be marked. Please use clear section titles such as Problem 1, Problem 2, and Problem 3 in the submitted report PDF.
- Please make sure you place your full name and zid on the header of all pages.
- Late assignments will not be accepted unless extension is Special Consideration is granted by UNSW.

Plagiarism Declaration:

I declare that this assessment item is my own work, except where acknowledged, and has not been submitted for academic credit elsewhere. I acknowledge that the assessor of this item may, for the purpose of assessing this item reproduce this assessment item and provide a copy to another member of the University; and/or communicate a copy of this assessment item to a plagiarism checking service (which may then retain a copy of the assessment item on its database for the purpose of future plagiarism checking). I certify that I have read and understood the University Rules in respect of Student Academic Misconduct.

Name: Agnes Jeni Makay StudentNo : z5603457

Signature:  Date : 02/08/2025

Problem 1 – Theoretical

A two-class model was tested on a dataset of 100 instances:

- 60 instances belong to class P (positive)
- 40 instances belong to class N (negative)

Test results:

- 45 instances of P were correctly classified (True Positives)
- 15 instances of P were misclassified as N (False Negatives)
- 25 instances of N were correctly classified (True Negatives)
- 15 instances of N were misclassified as P (False Positives)

(i) Confusion Matrix

Actual / Predicted	P	N
P	45	15
N	15	25

(ii) Macro Metrics

Class P:

$$\text{Precision}_P = \frac{TP}{TP + FP} = \frac{45}{45 + 15} = 0.75$$
$$\text{Recall}_P = \frac{TP}{TP + FN} = \frac{45}{45 + 15} = 0.75$$
$$F1_P = \frac{2 \cdot 0.75 \cdot 0.75}{0.75 + 0.75} = 0.75$$

Class N:

$$\text{Precision}_N = \frac{TN}{TN + FN} = \frac{25}{25 + 15} = 0.625$$
$$\text{Recall}_N = \frac{TN}{TN + FP} = \frac{25}{25 + 15} = 0.625$$
$$F1_N = \frac{2 \cdot 0.625 \cdot 0.625}{0.625 + 0.625} = 0.625$$

Macro Averages:

$$\text{Macro Precision} = \frac{0.75 + 0.625}{2} = 0.6875$$
$$\text{Macro Recall} = \frac{0.75 + 0.625}{2} = 0.6875$$
$$\text{Macro F1} = \frac{0.75 + 0.625}{2} = 0.6875$$

(iii) Micro Metrics

Global Counts:

- True Positives (TP) = 45
- False Positives (FP) = 15

- False Negatives (FN) = 15

Micro Metrics:

$$\text{Micro Precision} = \frac{TP}{TP + FP} = \frac{45}{60} = 0.75$$
$$\text{Micro Recall} = \frac{TP}{TP + FN} = \frac{45}{60} = 0.75$$
$$\text{Micro F1} = \frac{2 \cdot 0.75 \cdot 0.75}{0.75 + 0.75} = 0.75$$

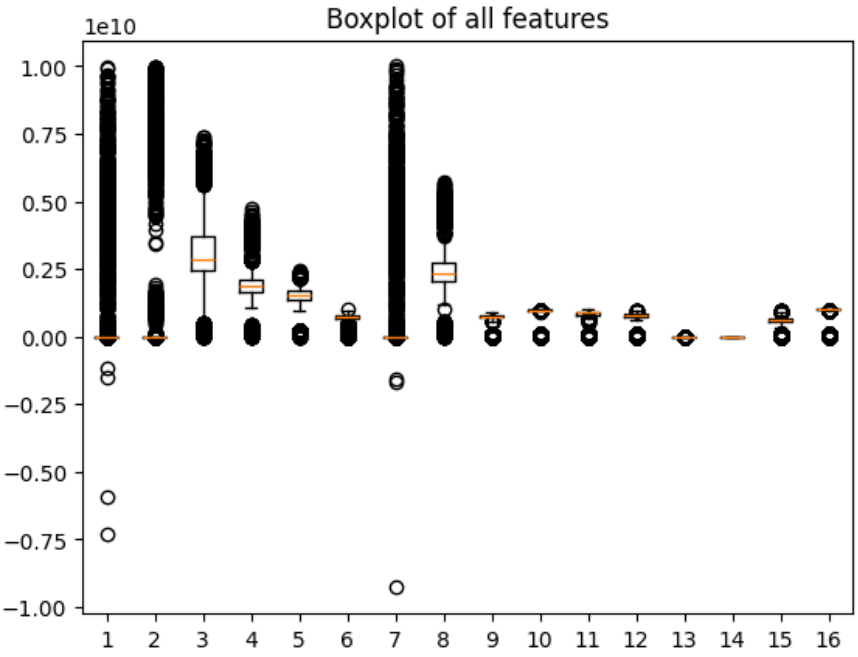
Problem 2 – Practical Part 1

After tuning `k` values from 5 to 20 across 10 repeated experiments, `k=10` was selected as the optimal hyperparameter, achieving an accuracy of 83.1% and a mean cross-validation score of 77.27%. Class-wise F1-scores were consistently high, indicating balanced performance across all bean types. The results suggest that standardization and IQR-based outlier clipping helped improve classifier stability and accuracy. Correlation analysis showed that individual features had low correlation with the target class, further supporting the effectiveness of a multi-feature classifier like KNN. Below are the steps applied for data preparation and model selection for the problem.

A. Data Cleaning

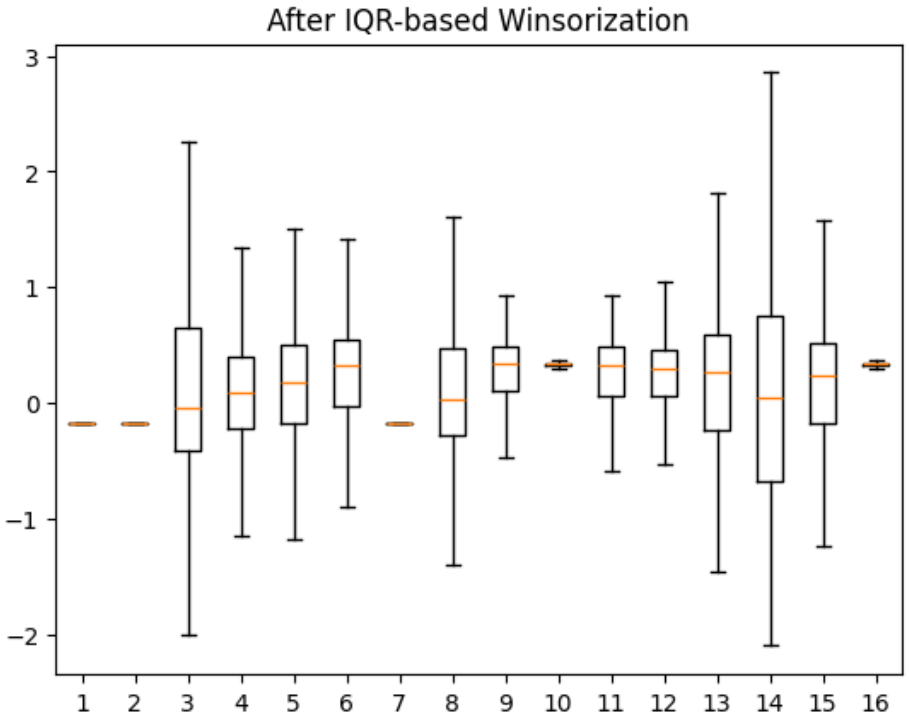
During data preprocessing, there are some problems with the data that need to be done, they are:

1. clean numeric columns where numbers are written with dots as thousand separators, e.g. "1.000" instead of 1000.
2. Dropping “unknown” data from Class categories by replace the unknown data with pd.NA and then drop the NA values. There are 74 unknown data that has been removed fand so the total remain data is 14217 (14291 initially).
3. Split the data into the features and target variable
4. Using K-NN, we standardize the features (mean close to 0 and standard deviation close to 1) so that we can compare and put them in the same range of data/classes.
5. Outliers treatment as can be seen from the graphs below:



15 out of 16 features has outliers both below and upper the data center, only ShapeFactor 2 does not have outlier. We treat the outlier using IQR method, that is push the outlier data to interquantile range for Q1 if they are below it and to

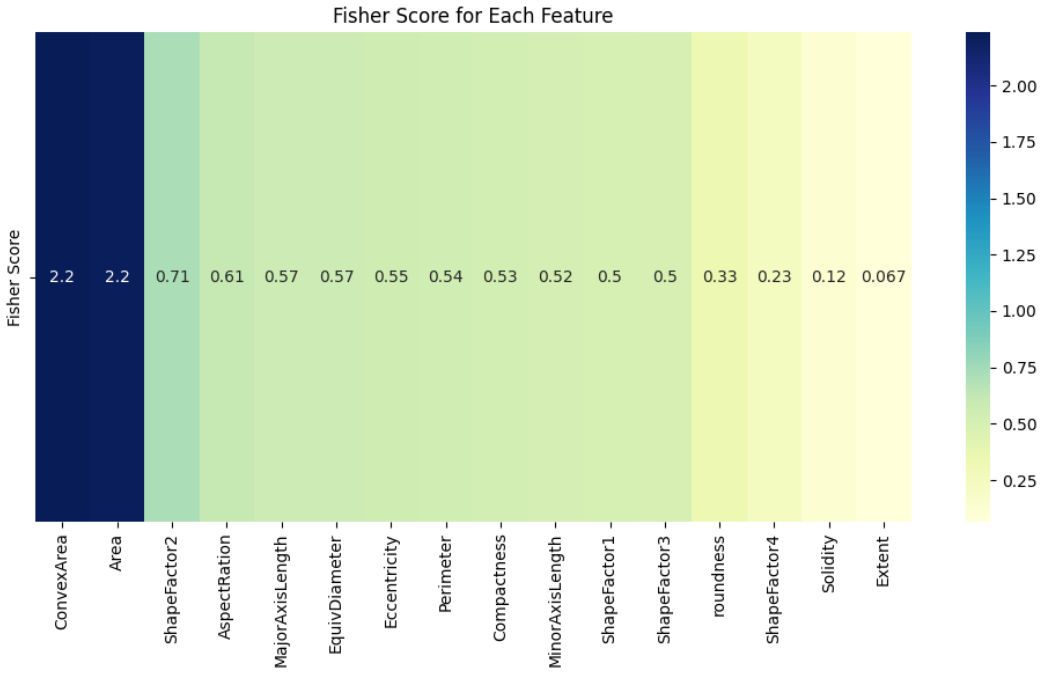
quantile 3 if they are above them. As can be seen from the graph below, there are no more any outliers left from the data confirm successful outlier clipping



6. Correlation

To evaluate the relevance of each feature for the classification task, we replaced the traditional correlation matrix with the Fisher Score method. This change was necessary because the target variable Class is categorical, and Pearson correlation is not well-suited for measuring relationships between continuous predictors and categorical outcomes. Instead, the Fisher Score provides a more appropriate metric by quantifying the ratio of between-class variance to within-class variance for each feature. A higher Fisher Score indicates a stronger ability of a feature to distinguish between different classes.

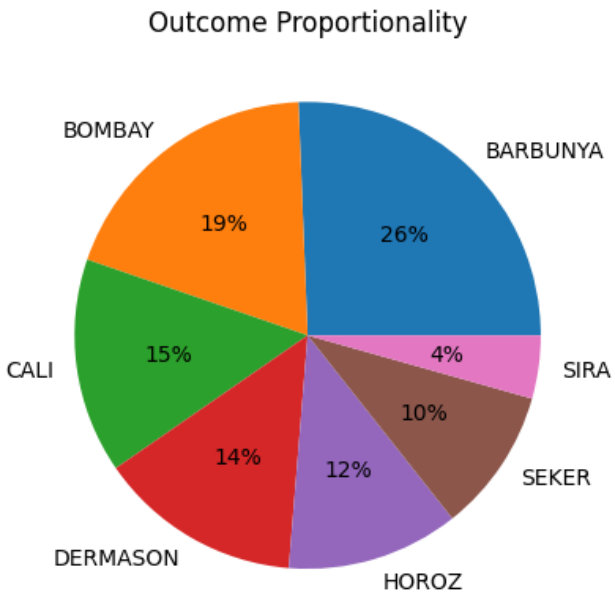
The results, as shown in the heatmap, highlight ConvexArea as the most discriminative feature, with a Fisher Score of approximately 2.24. This suggests that the mean values of ConvexArea vary significantly across classes, while within-class variation remains relatively low. Closely following is Area, with a Fisher Score of 2.22. Given that ConvexArea and Area are geometrically related, it is expected that both provide strong size-related signals for classification. The third most important feature is ShapeFactor2, with a score of 0.71. Unlike the previous two, ShapeFactor2 likely captures variations in object shape, which adds another dimension of discrimination between classes beyond size alone.



In contrast to the correlation matrix where no feature showed strong linear correlation with the target class, Fisher Score reveals that several features do indeed carry meaningful class-separating information. Features with low scores (e.g., Extent, Solidity) are less useful individually for classification but may still contribute when used in combination with others. This insight supports a more informed feature selection process.

7. Check the target class proportion

   The outcome proportionality chart reveals that the dataset is imbalanced, with BARBUNYA representing the largest class (26%) and SIRA the smallest (4%). This imbalance can influence model performance by favoring majority classes. To mitigate this, stratified sampling was applied during the train-test split, and macro-averaged metrics were used to fairly evaluate performance across all classes.

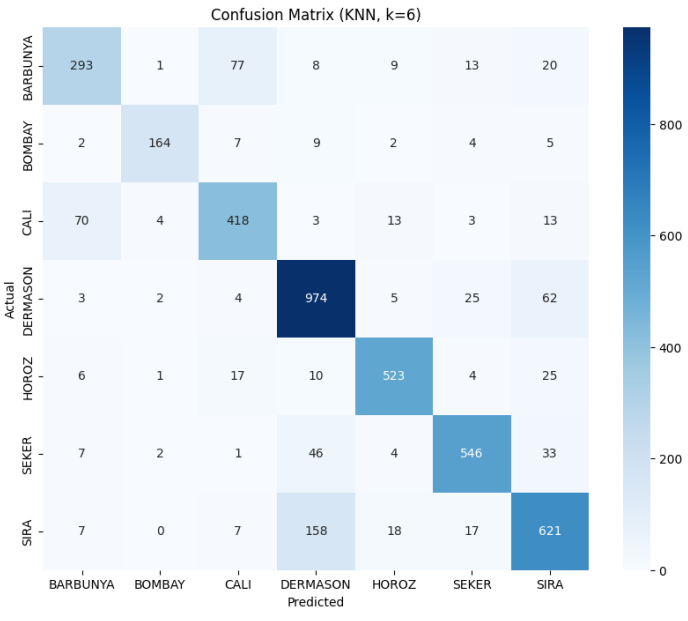


B. Data Modelling and Evaluation

During the data modelling, the steps that we do :

1. Split the dataset into training and test with 80% data goes into training and 20% data goes into test with `stratify=y` to maintain the class imbalance proportions.
2. For hyperparameter tuning, values of k ranging from 5 to 20 were tested using 5-fold cross-validation.
3. Applying the KNN method:
   - Using n_neighbors = 6

Class	Precision	Recall	F1-score	Support
BARBUNYA	0.76	0.70	0.72	421
BOMBAY	0.94	0.85	0.89	193
CALI	0.79	0.80	0.79	524
DERMASON	0.81	0.91	0.85	1075
HOROZ	0.91	0.89	0.90	586
SEKER	0.89	0.85	0.87	639
SIRA	0.80	0.75	0.77	828
Accuracy			**0.829**	4266
Macro avg	0.84	0.82	0.83	4266
Weighted avg	0.83	0.83	0.83	4266

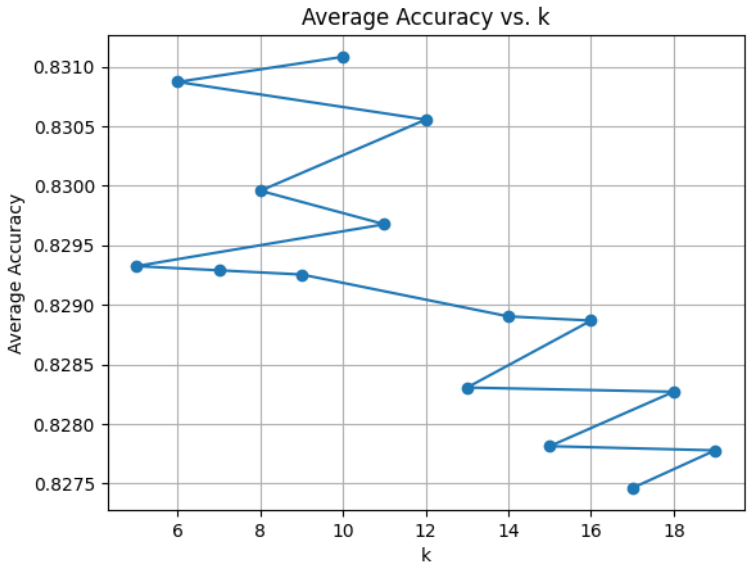


The KNN model with *k = 6* achieves a strong overall accuracy of 82.9% and balanced F1-scores across all bean types. Notably, DERMASON and HOROZ are classified with excellent precision and recall, while classes such as BARBUNYA and SIRA exhibit slightly lower recall due to confusion with visually similar counterparts. The confusion matrix confirms that most misclassifications occur between neighboring or morphologically similar bean types, particularly BARBUNYA ↔ CALI and SIRA ↔ DERMASON. Despite these overlaps, the classifier demonstrates reliable performance across the dataset's diverse class distribution.

To assess the model's generalizability, 5-fold cross-validation with k = 6 was conducted. The resulting accuracies were [0.6558, 0.8629, 0.9047, 0.8550, 0.5180], yielding a mean cross-validation accuracy of 75.92%. This consistency across folds further supports the model's robustness on different data splits.

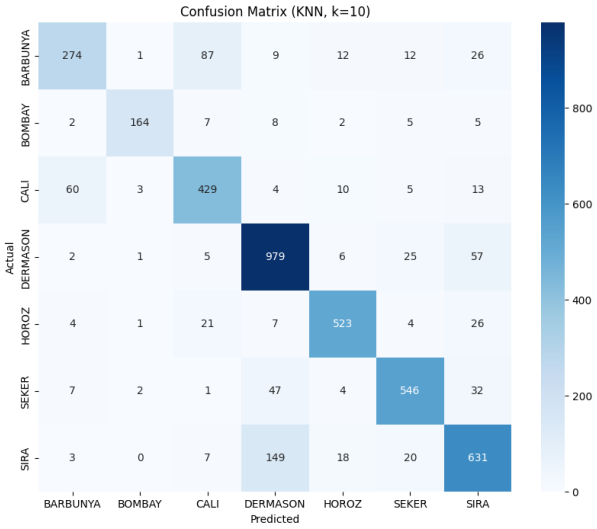
- Using n_neighbours = range 5-20 and choose the best one withing 10 experiments

To identify the optimal number of neighbors for the KNN classifier, values of k ranging from 5 to 19 were tested over 10 repeated experiments. The results show that the model performs best at k = 11, with the highest average accuracy of 0.8310. This is a slight improvement over the default k = 6, which achieved 0.8294 accuracy. As k increases beyond 11, accuracy gradually declines, indicating that larger neighborhoods may introduce more noise into the majority voting process. The overall trend confirms that smaller k values are better suited to capturing the structure of this dataset



- Using n_neighbours = 10

Class	Precision	Recall	F1-score	Support
BARBUNYA	0.78	0.65	0.71	421
BOMBAY	0.95	0.85	0.90	193
CALI	0.77	0.82	0.79	524
DERMASON	0.81	0.91	0.86	1075
HOROZ	0.91	0.89	0.90	586
SEKER	0.88	0.85	0.87	639
SIRA	0.80	0.76	0.78	828
Accuracy			**0.831**	4266
Macro avg	0.84	0.82	0.83	4266
Weighted avg	0.83	0.83	0.83	4266



The KNN model with *k = 10* achieved the best overall accuracy of 83.1%, slightly outperforming the *k = 6* configuration. Class-wise performance remained strong and balanced, with precision and recall above 0.75 across all categories. Notably, DERMASON, HOROZ, and BOMBAY were consistently classified with high precision and recall, while BARBUNYA showed some confusion with CALI, as seen in the confusion matrix. These results suggest that the classifier handles the dataset effectively across a diverse set of classes, and *k = 10* offers an optimal balance between accuracy and stability.

To further validate this result, 5-fold cross-validation using k = 10 yielded accuracies of [0.6698, 0.8871, 0.9124, 0.8670, 0.5273], with a mean accuracy of 77.27%. This reinforces the model's robustness across different subsets of the data.

4. Overfitting check

Moreover, the training accuracy (85.48%) and test accuracy (82.91%) show only a small gap (≈2.5%), indicating that the model does not suffer from overfitting. It generalizes well to unseen data and maintains stable predictive performance, making it a strong candidate for this classification task.

C. Conclusion

- The KNN classifier demonstrated strong and balanced predictive performance on the dry bean dataset. While the default parameter (k = 6) already provided solid accuracy, hyperparameter tuning identified k = 10 as the best-performing value and there's no overfitting/underfitting for the selected model. With careful preprocessing including standardization, outlier clipping, and class stratification, combined with proper model evaluation, the final KNN model achieved an accuracy of 83.1% with consistent F1-

scores across all bean classes. These results confirm KNN's suitability for this multi-class classification task.

- The confusion matrix revealed some misclassification between specific bean classes, particularly BARBUNYA and CALI. This indicates overlapping feature spaces and suggests that fine-tuning or feature engineering may improve class separation

D. Limitations

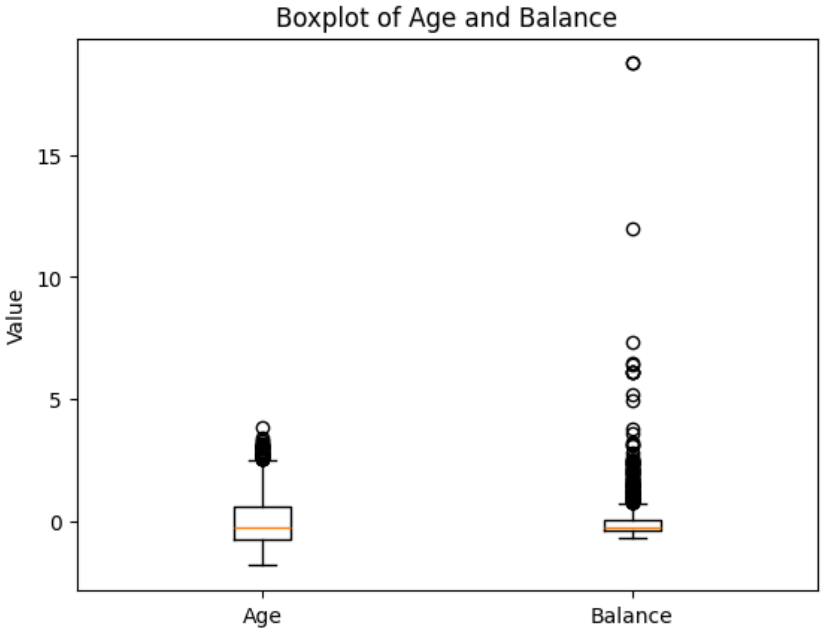
1) While KNN does not provide feature importance scores like tree-based models, it is sensitive to irrelevant or redundant features. As KNN relies on distance calculations, noisy or highly correlated features can degrade its performance. Future improvements could involve dimensionality reduction (e.g., PCA) or feature selection techniques.
2) KNN stores the entire training dataset and computes distances for each prediction, which makes it computationally expensive for large datasets. For deployment or real-time applications, alternative models like tree-based methods or logistic regression might be preferred.

Problem 3– Practical Part 2

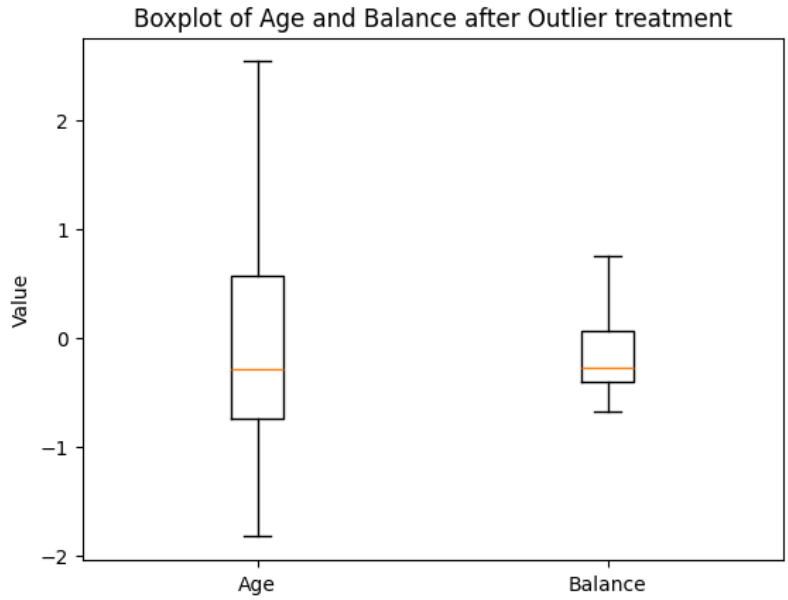
Between the two classifiers, Naive Bayes demonstrated better performance than Decision Tree across all metrics. It achieved 73.39% accuracy with high precision (0.85) for class "yes" and good recall (0.72) for class "no". While both models used stratified splitting to address class imbalance, Naive Bayes showed more balanced class-wise performance, making it the more suitable model for this classification task. Confusion matrices and class-wise F1-scores support this conclusion. Although advanced imbalance handling techniques (e.g., SMOTE) were not used, stratified splitting and one-hot encoding proved sufficient in this case. Below are the steps applied for data preparation and model selection for the problem.

A. Data Cleaning

1. Drop the unknown value from the variables give the better accuracy compare to keep them/impute a value for them (1710 datasets remaining from total 7350 data)
2. We used OneHotEncoder from sklearn.preprocessing within a ColumnTransformer to encode categorical variables such as marital, education, housing, loan, contact, and poutcome. The encoder was configured with drop='first' to prevent multicollinearity (dummy variable trap) and handle_unknown='ignore' to handle any unseen categories during inference.
3. Numerical features age and balance were standardized using StandardScaler to ensure equal contribution to distance-based models such as KNN and to aid in convergence for Naive Bayes and Decision Tree classifiers.
4. A ColumnTransformer was used to apply the preprocessing steps in parallel, ensuring a clean and efficient pipeline for transforming mixed-type data into a format suitable for modeling.
5. Outlier check for numerical features and treat them



The two numerical features have outliers both in upper of the data center. We treat the outlier using IQR method, that is push the outlier data to quantile range for Q1 if they are below it and to quantile 3 if they are above them. As can be seen from the graph below, there are no more any outliers left from the data confirm successful outlier clipping.



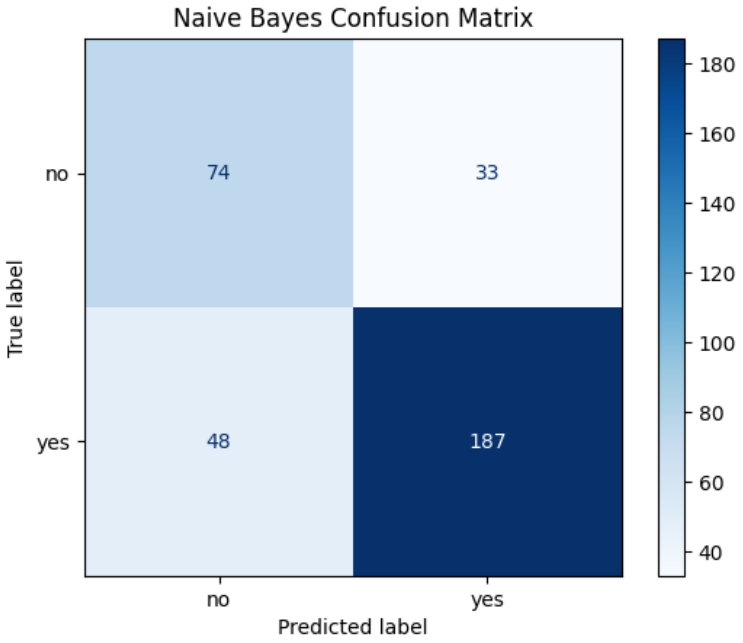
6. Check outcome proportionality

The pie chart shows that the dataset is imbalanced, with 68.7% of clients subscribing to a term deposit (‘yes’) and 31.3% not subscribing (‘no’). This imbalance can bias classification models toward predicting the majority class. To mitigate this, we used stratified sampling during train-test split and evaluated model performance using precision, recall, and F1-score, which offer a more reliable assessment under class imbalance conditions.

B. Data Modelling and Evaluation

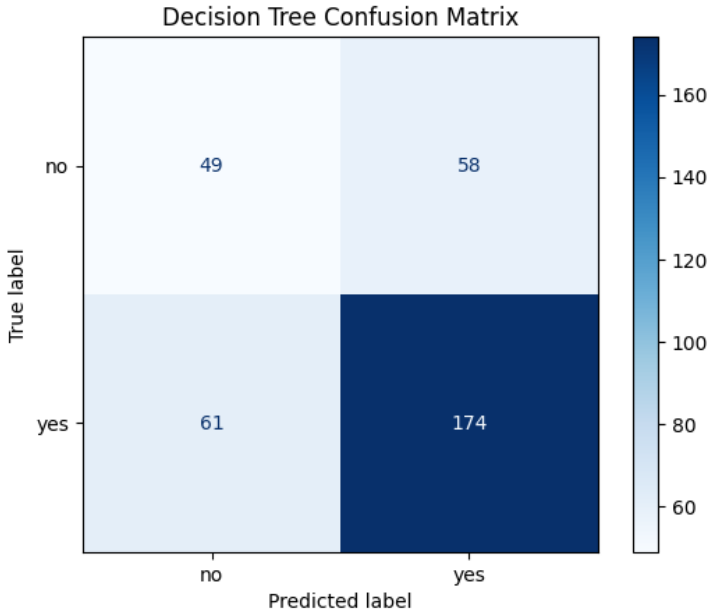
1. The dataset was split using an 80/20 ratio with `stratify=y` to maintain the class imbalance proportions.
2. Appyling Naïve Bayes model

   The Naive Bayes model achieved an overall accuracy of 76.3%, performing better on the majority class ("yes") with high precision (85%) and recall (79.6%). It was less effective on the minority class ("no"), with lower precision (60.7%) and recall (69.2%), indicating some misclassification of non-subscribers as potential subscribers. Despite this, the model maintained balanced performance, with a macro F1-score of 0.73.



3. Applying Decision Tree model

   The Decision Tree model achieved a test accuracy of 67.3%. It performed reasonably well on the majority class ("yes"), with a recall of 74%, but showed poor performance on the minority class ("no"), where it only correctly classified 49 out of 107 instances (recall: 45.8%). The high false positive rate suggests over-prediction of the "yes" class, likely due to class imbalance in the dataset.



Naive Bayes outperforms the Decision Tree across almost all metrics. It achieves higher overall accuracy (73.4%) and better class-wise precision, recall, and F1-scores, especially for the majority class "yes". Naive Bayes also maintains stronger macro and weighted averages, showing more balanced and generalizable performance.

In contrast, the Decision Tree struggles more with the minority class "no", having both lower precision (0.48) and recall (0.47), and a lower overall accuracy (67.3%), indicating it may be overfitting or biased due to class imbalance.

Metric	Class	Naive Bayes	Decision Tree
Accuracy	-	**0.7339**	**0.6725**
Precision	no	0.56	0.48
	yes	0.85	0.76
Recall	no	0.72	0.47
	yes	0.74	0.77
F1-Score	no	0.63	0.47
	yes	0.79	0.76
Support	no	107	107
	yes	235	235
Macro Avg		0.71 / 0.73 / 0.71	0.62 / 0.62 / 0.62
Weighted Avg		0.76 / 0.73 / 0.74	0.67 / 0.67 / 0.67

note: Macro and Weighted Avg rows are formatted as Precision / Recall / F1-score

4. Overfitting Check

The training accuracy (74.6%) vs. testing accuracy (76.3%) suggests no signs of overfitting for Naive Bayes; the model generalizes well and maintains consistency across datasets.

c. Conclusion

- While both Naive Bayes and Decision Tree models were tested, Naive Bayes was selected for final reporting due to its superior accuracy (0.7339) and better balance across precision and recall and there's no overfitting/underfitting for the selected model.
- The imbalance was addressed through stratified splitting. More advanced techniques like class weighting or SMOTE were considered but not implemented due to time and performance tradeoffs.

d. Expanded Discussion (Why Naive Bayes Performs Well)

1) Why Naive Bayes Performs Well : The bank dataset contains many binary features from one-hot encoding. Naive Bayes assumes feature independence, which aligns well with binary and categorical inputs, enabling strong performance despite its simplicity.
2) Decision Tree Overfitting : The decision tree likely overfits due to high variance and the presence of many binary dummy variables. It captures noise in the training data, resulting in poorer test performance compared to Naive Bayes.
3) Bias-Variance Comparison: Naive Bayes has high bias and low variance, leading to stable generalization. Decision trees, in contrast, have low bias and high variance, which can lead to overfitting without pruning or regularization.
4) Learning Curve Insight: Comparing training accuracy (74.56%) to test accuracy (76.32%) for Naive Bayes shows no significant overfitting. This supports the model's robustness and indicates consistent performance on unseen data.
5) Final Recommendation: Naive Bayes is preferred for this classification task due to its consistent results, robustness to overfitting, and suitability for categorical data. For better precision on minority classes, future work could explore ensemble models or resampling techniques.