ENGR 5005 – Machine Learning for Engineering Applications
Winter 2025 – Project Report
Machine Learning Models from Scratch

Sali Alsafadi [a], Meaghan Charest-Finn [a]

[a] *Department of Electrical, Computer and Software Engineering, Ontario Tech University*

1. Abstract

This project explores the manual implementation and comparative analysis of core machine learning classification models. Without using external libraries, five algorithms were developed from scratch: Logistic Regression, Linear Regression, Decision Tree, K-Nearest Neighbors (KNN), and Naive Bayes. These models were evaluated across five real-world datasets featuring both binary and multi-class classification tasks. Standard metrics including accuracy, precision, recall, F1 score, and confusion matrices were used to assess performance. Findings show that model performance varied significantly with dataset characteristics such as class balance, feature dimensionality, and noise. Decision Trees and KNN achieved strong results across most datasets, while Logistic Regression and Naive Bayes showed limitations in more complex or imbalanced settings. Linear Regression, although implemented, was not included in evaluation due to its unsuitability for classification tasks.

2. Introduction

Machine learning models play a vital role in modern data analysis, offering tools to discover patterns and make predictions from structured datasets. Despite the widespread availability of pre-built libraries such as Scikit-learn and TensorFlow, a deep understanding of how these algorithms operate internally is essential for developing intuition, troubleshooting models, and adapting them to new tasks. This project centers on implementing five foundational machine learning algorithms from scratch using Python and NumPy: *Logistic Regression*, *Linear Regression*, *Decision Tree*, *K-Nearest-Neighbors* (KNN), and *Naive Bayes*. The goal was to develop each model algorithmically, train and test them across diverse real-world datasets, and evaluate their performance using consistent, meaningful metrics.

The datasets chosen span a variety of tasks and challenges:

1. Breast Cancer
2. Mushroom: Binary classification of edibility based on physical traits.
3. Heart Failure Prediction: Binary prediction of patient mortality.
4. Wine Quality: Originally a regression problem, converted to binary classification.
5. Robot Execution Failures: A high-dimensional multi-class classification problem, filtered to the 10 most frequent classes for clarity.

Each dataset introduced its own preprocessing requirements, including Min-Max scaling, label encoding, and class balancing. Although Linear Regression was implemented, it was excluded from evaluation due to its incompatibility with classification tasks. In contrast, Logistic Regression was adapted to binary classification problems and extended to handle multi-class classification via a One-vs-Rest (OvR) strategy. The performance of each model was assessed using accuracy, precision, recall, F1 score, and confusion matrix breakdowns. These metrics provided a well-rounded view of model behavior, especially in cases of imbalanced datasets or skewed class distributions. This report details the implementation process, presents quantitative results, and discusses model performance in the context of each dataset.

3. Background Theory and Implementation

This section outlines the core theoretical principles of the machine learning models implemented in this project, as well as the approach taken to build them from scratch. All models were developed manually using Python and NumPy to reinforce algorithmic understanding and ensure transparency in learning. ***Linear Regression*** models the relationship between one or more input features and a continuous output by fitting a linear function. The model attempts to minimize the mean squared error (MSE) between predicted and actual values. Although this algorithm was implemented successfully, it was not used in the final classification results due to its unsuitability for categorical targets. Nonetheless, it served as a foundational step toward implementing more advanced models such as logistic regression. ***Logistic Regression*** is a linear model used for binary classification. Unlike linear regression, which predicts a continuous output, logistic regression applies the sigmoid function to the linear combination of input features to model the probability of a sample belonging to the positive class. The sigmoid function maps any real-valued input into the range (0, 1), making it suitable for probabilistic interpretation. The model was trained using batch gradient descent, minimizing the binary cross-entropy loss function. During each iteration, the algorithm updates the weights and bias using the gradients computed from the difference between predicted and actual class labels. To avoid numerical instability, the sigmoid function includes clipping to prevent overflow during exponentiation. The final prediction is obtained by thresholding the sigmoid output at 0.5 values above this threshold are classified as class 1, and those below as class 0. The learning rate and number of iterations are adjustable hyperparameters. While ***logistic regression*** is inherently a binary classifier, it can be extended to handle multi-class classification through the ***One-vs-Rest (OvR)*** strategy. This method involves training a separate binary classifier for each class, where the current class is treated as the positive class and all others as negative. In this implementation, a distinct logistic regression model is trained for every unique class label. During prediction, each model outputs a probability that the input belongs to its respective class. The final predicted class corresponds to the model that assigns the highest probability to the input sample. This approach is computationally efficient, easy to implement, and allows binary classifiers to be applied to multi-class problems. However, its performance may degrade when class probabilities are very close or when classes are highly imbalanced.

Decision Tree Classifier

A Decision Tree is a non-parametric, hierarchical learning algorithm used for classification and regression tasks. It works by recursively partitioning the dataset based on feature thresholds that yield the most informative splits. In this implementation, **entropy** is used as the impurity measure, and the best split is determined by maximizing the **information gain** at each node. The tree is built top-down using a recursive function that checks stopping conditions such as maximum depth, minimum sample split size, or label purity. Each internal node represents a decision based on a feature and threshold, while leaf nodes assign a class label. The model supports hyperparameters including max_depth to limit tree complexity and min_samples_split to avoid overfitting. During prediction, each sample is routed through the tree using binary comparisons at each node until a terminal leaf node is reached. This implementation captures the core logic of greedy tree construction and makes no external assumptions about feature types or class distributions. It performs well on structured datasets and offers high interpretability.

K-Nearest Neighbors (KNN)

K-Nearest Neighbors is a simple yet powerful ***non-parametric*** classification algorithm. It operates on the principle that similar instances exist in close proximity within the feature space. Rather than learning explicit model parameters, KNN defers computation until prediction time. For a given input, the algorithm calculates the ***Euclidean distance*** between the test sample and all training samples. It then selects the ***k-closest neighbors***, and the predicted class is determined by a majority vote among their labels. The custom implementation includes a configurable k parameter and uses NumPy for efficient vectorized distance computation. Since KNN does not make assumptions about data distribution, it performs well in many scenarios, especially when the decision boundary is nonlinear. However, its runtime and performance can degrade with high-dimensional data or large datasets due to the need to compute distances for each prediction.

Naive Bayes Classifier

Naive Bayes is a probabilistic model grounded in Bayes' Theorem, which assumes that input features are conditionally independent given the class label. In this project, a ***Gaussian Naive Bayes*** classifier was implemented to model continuous-valued features under the assumption that each feature follows a normal distribution. During training, the algorithm computes the mean, variance, and prior probabilities for each class. To ensure numerical stability in cases where variances approach zero, a small constant (alpha)is added to the variance values. This technique, known as variance smoothing, prevents division by zero errors and stabilizes likelihood computations. Additionally, log probabilities are used to avoid numerical underflow during prediction. This implementation does not use Laplace smoothing, as it is typically applied in categorical (discrete) Naive Bayes variants to handle zero-frequency problems.

Implementation Structure

The project was developed with a modular, maintainable architecture to facilitate scalability and reproducibility. The codebase is organized into the following main components:

utils/

1. data_loader.py: Dataset-specific preprocessing routines, such as label encoding, scaling, and class filtering.
2. metrics.py: Custom metric implementations: accuracy, precision, recall, F1 score, confusion matrix, and MSE.

utils/models/
Contains individual Python modules for each algorithm:

1. logistic_regression.py: Binary logistic regression via gradient descent
2. logistic_regression_ovr.py: One-vs-Rest extension for multi-class classification
3. linear_regression.py: Basic linear regression using MSE
4. decision_tree.py: Entropy-based decision tree classifier
5. k_nearest_neighbors.py: KNN classifier with Euclidean distance
6. naive_bayes.py: Gaussian Naive Bayes with smoothing

tests/
Includes separate scripts to evaluate all models on each dataset. Each script loads the data, splits it into training and testing sets, fits the models, and reports performance using the metrics module.

4. Results

Breast Cancer Dataset

The Breast Cancer dataset yielded high performance across all models, with Naive Bayes achieving the best overall results (accuracy: 96.46%, F1 score: 0.9512). Logistic Regression demonstrated excellent recall (1.0), correctly identifying all malignant cases, though this came at the cost of increased false positives. Both Decision Tree and KNN performed competitively, indicating that the dataset's numeric, well-distributed features were compatible with a wide range of learning algorithms. The Gaussian assumption made by Naive Bayes appears to align well with the distribution of the data, which explains its superior performance.

Table 1: Performance metrics of different models on the Breast Cancer dataset

Model	Accuracy	Precision	Recall	F1 Score	Confusion Matrix
Logistic Regression	0.9469	0.8750	1.0000	0.9333	TP=42, TN=65, FP=6, FN=0
Decision Tree	0.9292	0.9048	0.9048	0.9048	TP=38, TN=67, FP=4, FN=4
K-Nearest Neighbors	0.9292	0.9250	0.8810	0.9024	TP=37, TN=68, FP=3, FN=5
Naive Bayes	0.9646	0.9750	0.9286	0.9512	TP=39, TN=70, FP=1, FN=3

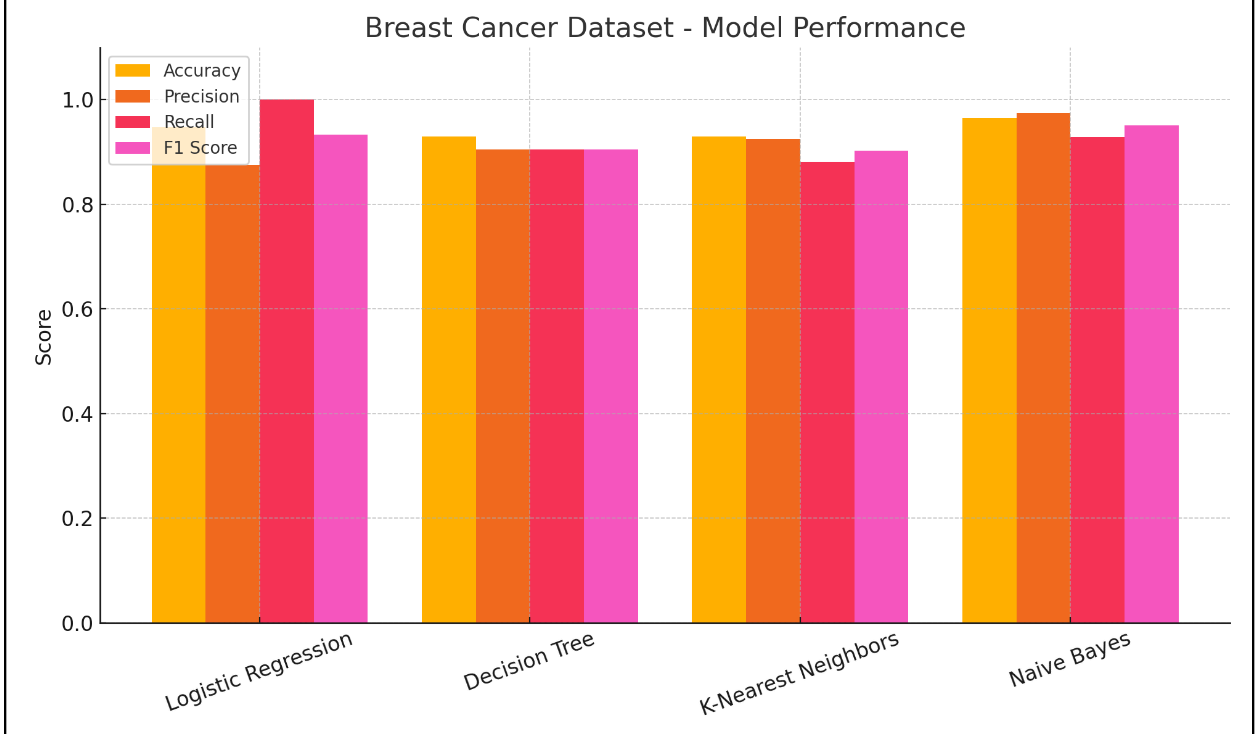

Figure 1: Performance comparison of various models on the Breast Cancer dataset. Naive Bayes achieved the highest accuracy and F1 score, while Logistic Regression showed perfect recall.

Mushroom Dataset

This dataset produced near-perfect results for several models. Decision Tree and KNN both achieved 100% accuracy, precision, recall, and F1 score. These results are not due to overfitting or data leakage but are instead a reflection of the dataset's inherent separability. Each mushroom sample is described entirely by categorical features, which, once encoded numerically, created very distinct patterns for each class. KNN was able to effectively "memorize" the dataset, while Decision Tree easily captured the feature thresholds that separate edible and poisonous mushrooms. Logistic Regression and Naive Bayes performed slightly worse due to the dataset's non-linear structure and lack of feature independence, respectively.

Table 2: Performance metrics of different models on the Mushroom dataset.

Model	Accuracy	Precision	Recall	F1 Score	Confusion Matrix
Logistic Regression	0. 9076	0.9135	0.8924	0.9028	TP=697, TN=777, FP=66, FN=84
Decision Tree	1.0000	1.0000	1.0000	1.0000	TP=781, TN=843, FP=0, FN=0
K-Nearest Neighbors	1.0000	1.0000	1.0000	1.0000	TP=781, TN=843, FP=0, FN=0
Naive Bayes	0.9212	0.9605	0.8720	0.9141	TP=681, TN=815, FP=28, FN=100

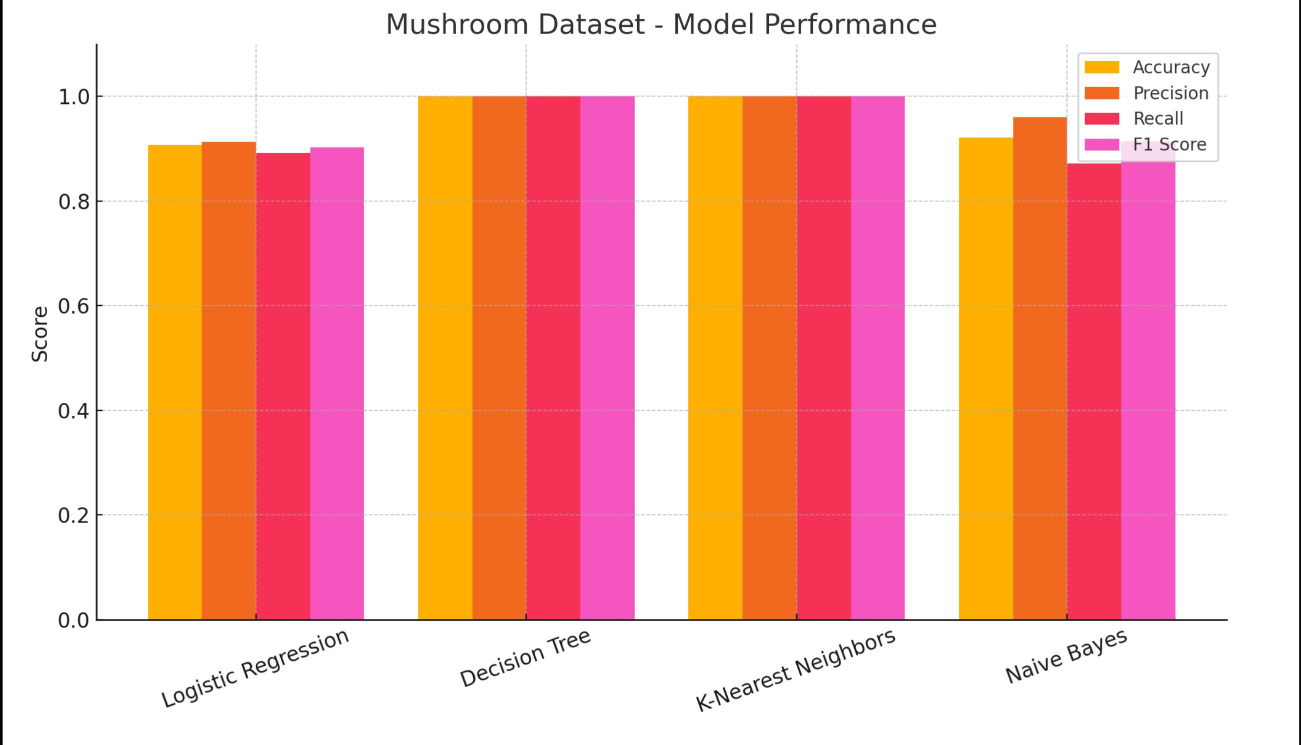

Figure 2: Comparative performance of classification models on the Mushroom dataset

Heart Failure dataset

Model performance on the Heart Failure dataset was more modest, with no model exceeding an F1 score of 0.51. Logistic Regression initially failed completely with only 1000 training iterations but improved substantially after increasing to 3000 iterations. Despite this, recall remained low across all models, indicating a consistent difficulty in identifying positive (death) cases. This is likely due to the small dataset size, class imbalance, and overlapping features between positive and negative samples. Naive

Bayes and Decision Tree achieved slightly better balance, but overall, this dataset presented a significant challenge to all implemented models.

Table 3: Performance metrics of different models on the Heart Failure dataset

Model	Accuracy	Precision	Recall	F1 Score	Confusion Matrix
Logistic Regression with iteration 1000	0.5932	0.0000	0.0000	0.0000	TP=0, TN=35, FP=0, FN=24
Logistic Regression with iteration 3000	0.7119	0.8889	0.3333	0.4848	TP=8, TN=34, FP=1, FN=16
Decision Tree	0.6610	0.6250	0.4167	0.5000	TP=10, TN=29, FP=6, FN=14
K-Nearest Neighbors	0.6780	0.8571	0.2500	0.3871	TP=6, TN=34, FP=1, FN=18
Naive Bayes	0.6949	0.8000	0.3333	0.4706	TP=8, TN=33, FP=2, FN=16

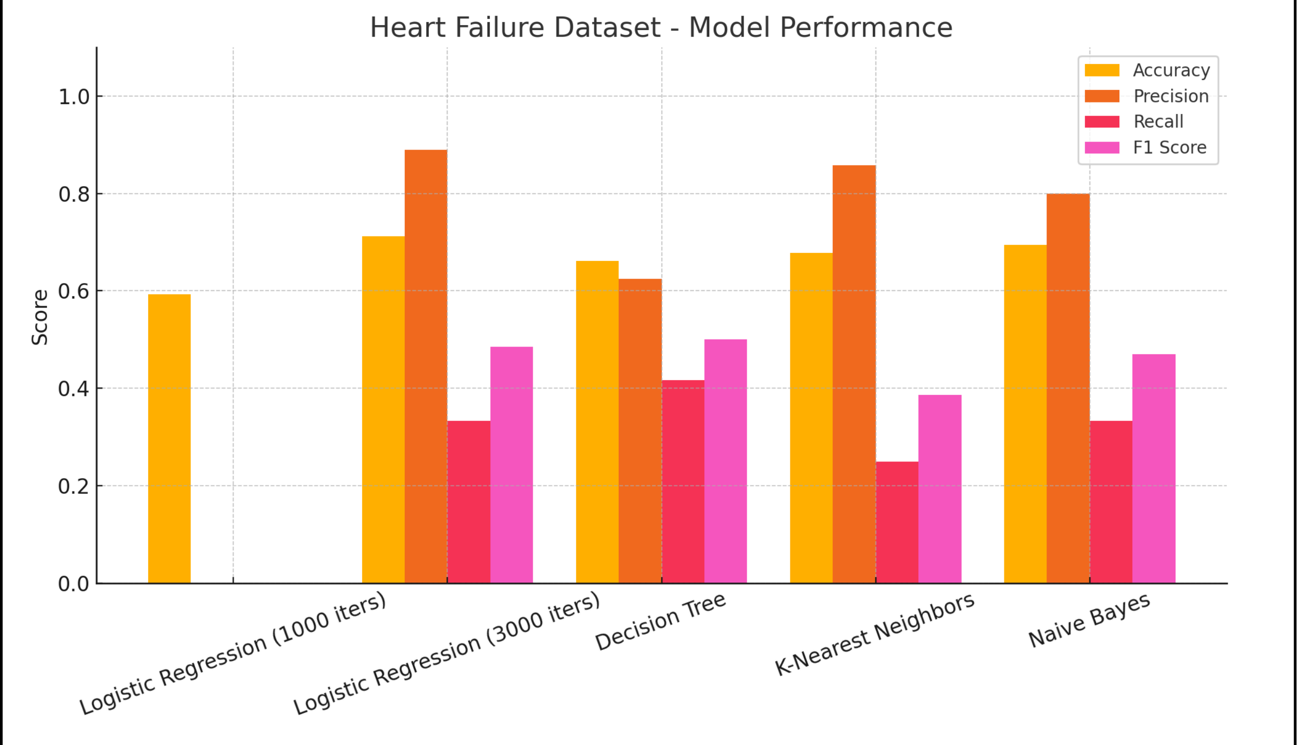

Figure 3: Comparative performance of classification models on the Heart Failure Dataset.

Wine Quality Dataset (Before Balancing)

When tested on the unbalanced Wine Quality dataset, all models struggled, particularly Logistic Regression, which produced zero predictions for the positive class. This reflects a classic failure in imbalanced classification: models default to the majority class and ignore the minority. Decision Tree and KNN performed better by capturing some of the signal in the features, though precision and recall were still relatively low. Naive Bayes showed high recall but a poor precision score due to a large number of false positives. These results emphasize the importance of class balance when evaluating model performance.

Table 4: Performance metrics of different models on the Wine Quality before balancing.

Model	Accuracy	Precision	Recall	F1 Score	Confusion Matrix
Logistic Regression with iteration 3000	0.7681	0.0000	0.0000	0.0000	TP = 0, TN = 752, FP = 0, FN = 227
Decision Tree	0.8212	0.6040	0.6652	0.6331	TP = 151, TN = 653, FP = 99, FN = 76
K-Nearest Neighbors	0.8304	0.6517	0.5771	0.6121	TP = 131, TN = 682, FP = 70, FN = 96
Naive Bayes	0.7109	0.4251	0.7004	0.5291	TP = 159, TN = 537, FP = 215, FN = 68

Wine Quality Dataset (After Balancing)

After balancing the classes in the Wine Quality dataset, performance improved dramatically across all models. KNN produced the best overall results, with an F1 score of 0.7723 and recall of 0.8480, showing its effectiveness in high-dimensional, structured spaces when class proportions are equal. Logistic Regression, previously ineffective, achieved a strong F1 score of 0.7352. Decision Tree and Naive Bayes also performed well, demonstrating that the features contain sufficient information for classification when the label distribution is even. These results clearly demonstrate how class balancing transforms model effectiveness.

Table 5: Performance metrics of different models on the Wine Quality after balancing.

Model	Accuracy	Precision	Recall	F1 Score	Confusion Matrix

Logistic Regression	0.7264	0. 6880	0.7892	0. 7352	TP = 161, TN = 147, FP = 73, FN = 43
Decision Tree	0.7500	0.7094	0.8137	0. 7580	TP = 166, TN = 152, FP = 68, FN = 38
K-Nearest Neighbors	0.7594	0.7090	0.8480	0. 7723	TP = 173, TN = 149, FP = 71, FN = 31
Naive Bayes	0.7358	0.6885	0.8235	0. 7500	TP = 168, TN = 144, FP = 76, FN = 36

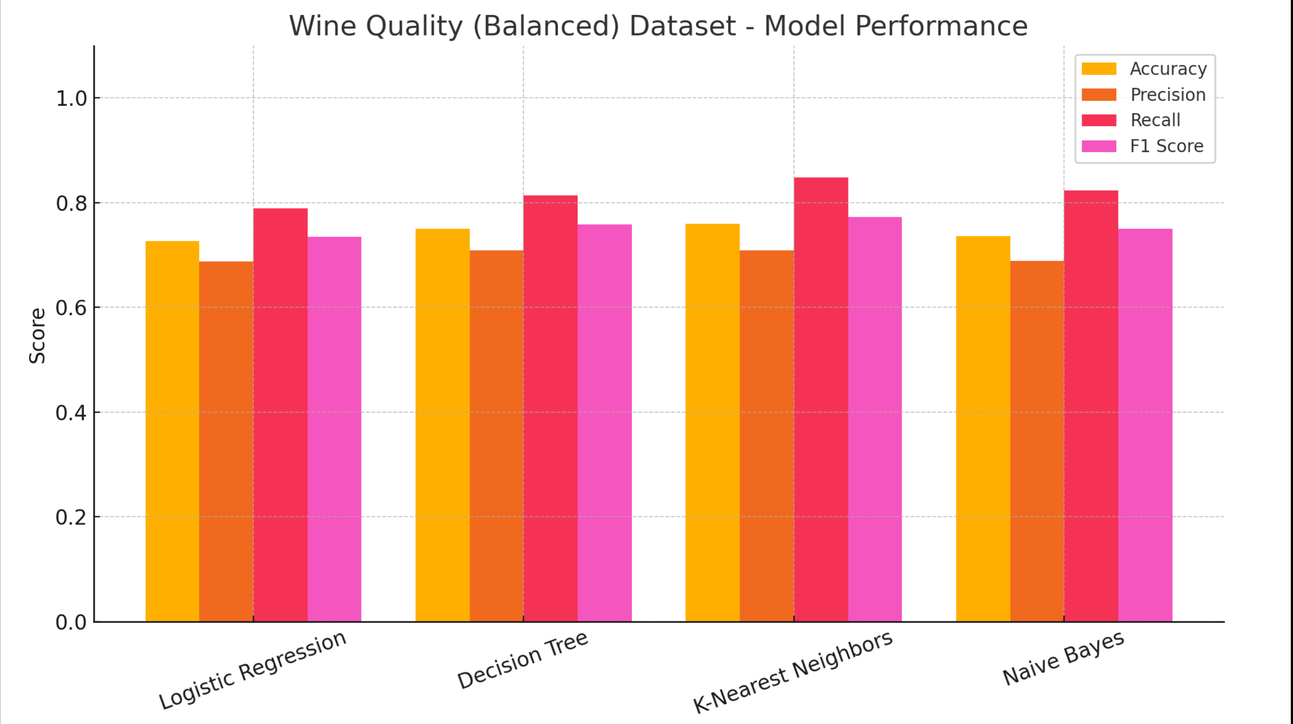

Figure 4: Comparative performance of classification models on the Wine Quality balanced Dataset.

Robot Execution Failures Dataset (All Classes)

With over 100 unique class labels, the full Robot Execution Failures dataset proved to be extremely challenging for all models. F1 scores remained below 0.16 for every classifier tested, reflecting the sparse, high-cardinality nature of the target labels and potentially noisy feature space. The One-vs-Rest Logistic

Regression model was removed due to poor numerical performance and extreme computational cost. Overall, none of the implemented algorithms could effectively model this dataset without further preprocessing or dimensionality reduction.

Table 6: Performance metrics of different models on the Robot Execution Failures (All classes).

Model	Accuracy	Precision (Macro)	Recall (Macro)	F1 Score (Macro)
Decision Tree	0.3712	0.1726	0.1716	0.1608
K-Nearest Neighbors	0.3712	0.1552	0.1460	0.1439
Naive Bayes	0.3144	0.0800	0.0808	0.0786

Robot Execution Failures Dataset (Top 10 Classes Only)

To mitigate the challenges of the full Robot dataset, a filtered version containing only the 10 most frequent classes was evaluated. This simplification resulted in modest performance improvements across all models, with KNN achieving the best F1 score of 0.2381. Nevertheless, the overall scores remained relatively low, suggesting that even the most common classes are difficult to separate based on the available features. This version of the dataset is still non-trivial and would likely benefit from additional feature engineering or more advanced models.

Table 7: Performance metrics of different models on the Robot Execution Failure (Top 10 Classes).

Model	Accuracy	Precision (Macro)	Recall (Macro)	F1 Score (Macro)
Decision Tree	0.4074	0.2125	0.2318	0.2181
K-Nearest Neighbors	0.4021	0.2514	0.2426	0.2381
Naive Bayes	0.3757	0.2325	0.1700	0.1438

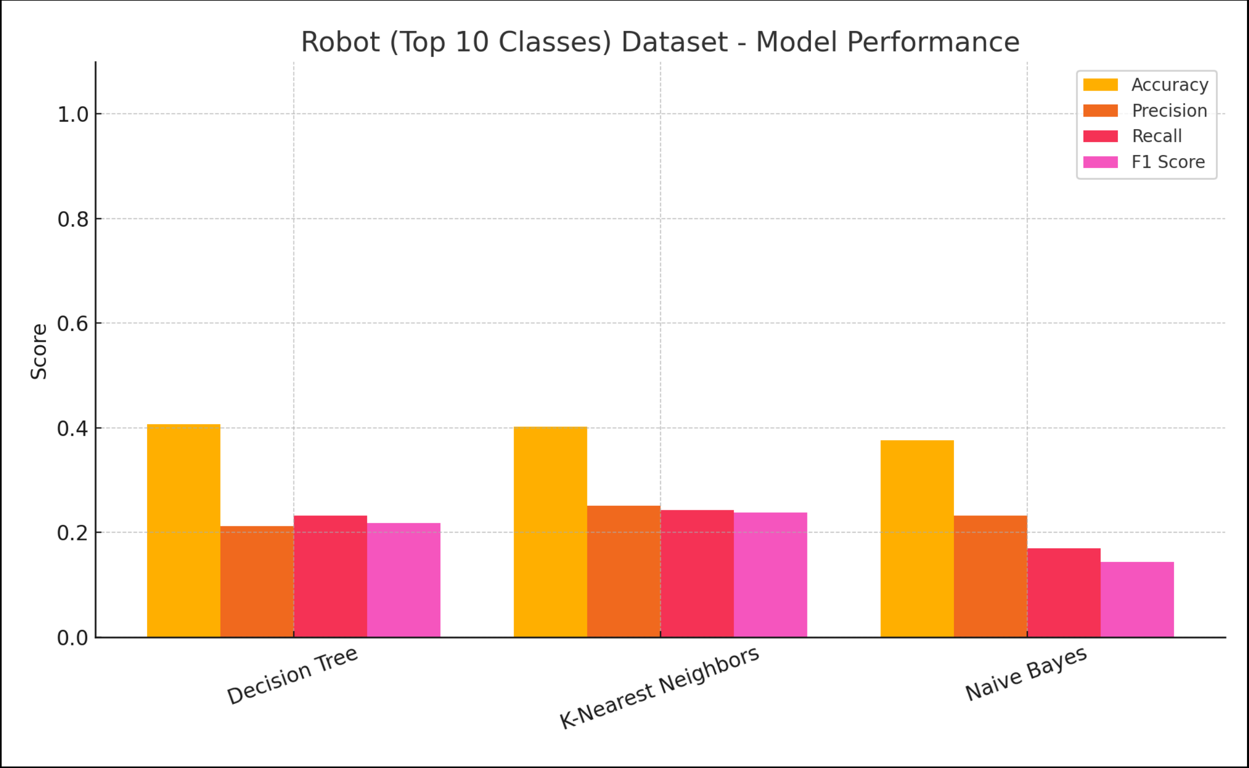

Figure 5: Comparative performance of classification models on the Robot Execution Failure (Top 10 classes).

Among the four implemented models, Naive Bayes consistently showed solid performance across multiple datasets, particularly excelling on the Breast Cancer and Mushroom datasets due to their well-separated features. Decision Trees stood out for their adaptability, achieving perfect accuracy and F1 Score on the Mushroom dataset, and performing reliably on Wine Quality and Breast Cancer. K-Nearest Neighbors (KNN) showed strong results on structured datasets and benefited greatly from data balancing,its performance improved noticeably on the balanced Wine Quality dataset. Logistic Regression had competitive accuracy on simpler binary tasks like Breast Cancer but struggled on imbalanced datasets like Heart Failure and Wine Quality until additional iterations and data preprocessing were applied. In more complex, multi-class settings like the Robot dataset, all models performed poorly, but Decision Tree and KNN showed relatively better generalization compared to Naive Bayes. Overall, Decision Tree and KNN were the most versatile, while Logistic Regression was more sensitive to data scaling and balance, and Naive Bayes worked best when feature independence assumptions were approximately valid.

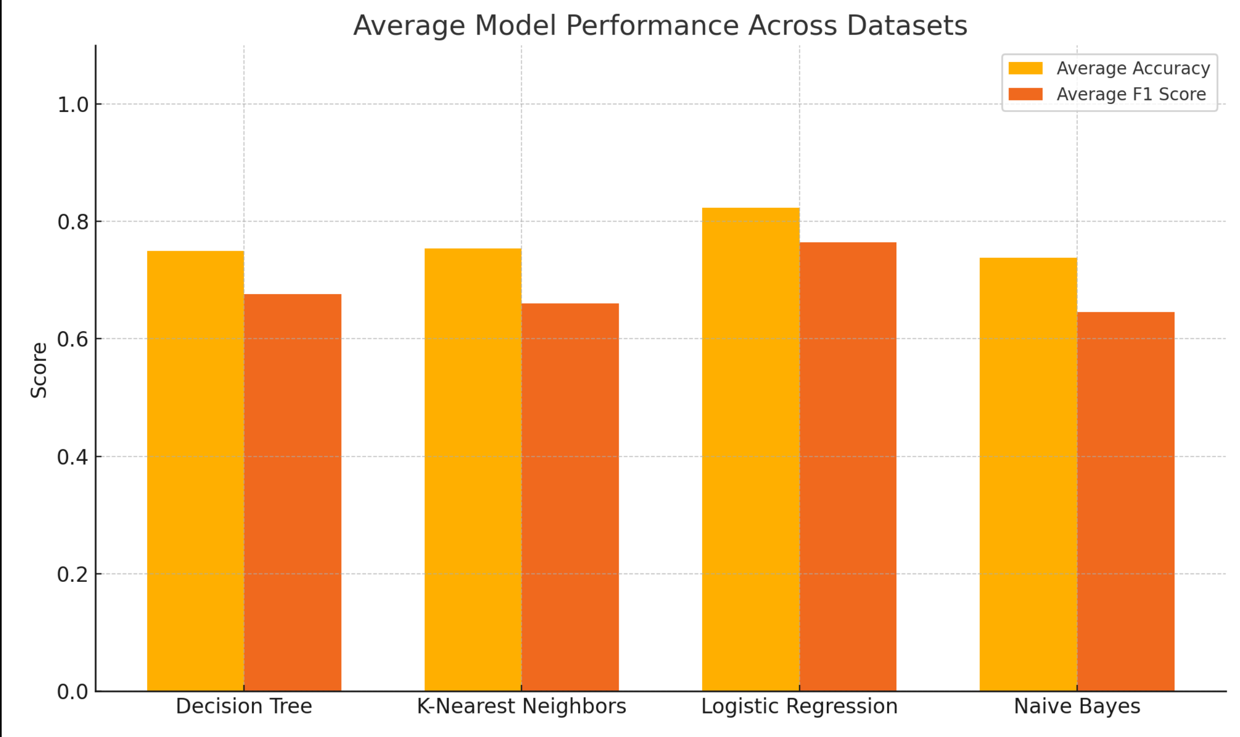

Figure 6: Average Model Performance Across Datasets (Accuracy , F1 score).

8. Conclusion

This project involved implementing four core classification algorithms: Logistic Regression, Decision Trees, Naive Bayes, and K-Nearest-Neighbors and testing them across five datasets with varying structures and levels of complexity. Each model was coded manually using only basic Python and NumPy operations, allowing for full control over their behavior and evaluation. The models performed well on structured datasets such as Breast Cancer and Mushroom, where patterns in the data were clear and easily separable. On the Mushroom dataset, both Decision Tree and KNN achieved perfect scores, highlighting the strength of non-parametric models on categorical data. For the Breast Cancer dataset, Naive Bayes showed strong results, likely due to the numeric nature of the features and their alignment with Gaussian assumptions. In contrast, datasets like Heart Failure and Wine Quality revealed limitations, particularly in the presence of class imbalance. Logistic Regression initially failed on the imbalanced Wine dataset but improved significantly after the data was balanced. The Heart Failure dataset produced low recall across all models, indicating difficulty in identifying positive cases. The Robot Execution dataset presented the most difficulty. Its high number of unique classes and irregular feature patterns led to low performance across all models, even after reducing it to the top 10 classes. The experiments show that model effectiveness is closely linked to the nature of the dataset. Preprocessing steps such as scaling, encoding, and balancing had a direct impact on results, and no single algorithm outperformed the others in every case. The evaluation highlighted how different models respond to different data challenges, reinforcing the importance of selecting algorithms based on data characteristics.