University of Dhaka

Department of Computer Science and Engineering

CSE-4255 : Introduction to Data Mining and Warehousing Lab

Lab Assignment: Implementation and Comparative Evaluation of Decision Tree and Naive Bayes Classification Algorithms on Multiple Datasets Using Performance Metrics

Submitted On:

June 25, 2025

Submitted To:

Dr. Chowdhury Farhan Ahmed

Professor
Department of Computer Science & Engineering
University of Dhaka

Md. Mahmudur Rahman

Assistant Professor
Department of Computer Science & Engineering
University of Dhaka

Submitted By:

Himel Chandra Roy

Roll No : 13

Registration No : 2020915623

Introduction

Classification is a fundamental task in data mining that involves predicting the category of data instances based on input features. This report focuses on the implementation and comparison of two widely used classification algorithms: Decision Tree and Naive Bayes. Both methods offer unique advantages—Decision Trees are intuitive and handle both numerical and categorical data well, while Naive Bayes is fast and effective, especially with high-dimensional data. To evaluate their performance, we apply these algorithms to multiple real-world datasets and assess them using several metrics, including accuracy, precision, recall, F1-score, and AUC. The goal is to provide insights into the strengths and limitations of each algorithm under varying data characteristics.

Algorithm Overview

Decision Tree

A **Decision Tree** is a flowchart-like tree structure where internal nodes represent tests on features, branches represent the outcomes of those tests, and leaf nodes represent class labels. It recursively splits the dataset into subsets based on the value of input features to improve class purity.

Key Concepts

- Entropy and Information Gain: Measure the impurity in a dataset and the reduction in impurity after a split, respectively.
- Gini Index: Measures the impurity by calculating the probability of incorrect classification.
- Recursive Partitioning: The process continues until stopping criteria are met (e.g., all data points in a node belong to one class or maximum depth is reached).
- The Gini Impurity is used as the measure of node impurity.
- All features are assumed to be categorical after label encoding.
- The algorithm searches for the best feature and value to split on by maximizing Information Gain, which is the reduction in impurity from a split.
- It recursively builds the tree until no further gain is possible.
- Prediction is made by traversing the tree from the root to a leaf node and returning the most frequent label at that node.

Naive Bayes

The **Naive Bayes** classifier is a probabilistic model based on Bayes' Theorem, with the simplifying assumption that all features are conditionally independent given the class label.

Key Concepts

- The algorithm computes prior probabilities $P(C)$ for each class and likelihoods $P(x_i|C)$ for each feature value.
- Likelihoods are estimated using frequency counts, normalized within each class.
- Bayes' theorem is used here:

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

- For prediction, the classifier computes the posterior probability of each class for a given instance using:

$$P(C|X) \propto P(C) \cdot \prod_{i=1}^{n} P(x_i|C)$$

  Where

  - $P(C|X)$: Posterior probability of class C given feature vector X
  - $P(X|C)$: Likelihood of features given class
  - $P(C)$: Prior probability of class
  - $P(X)$: Evidence or marginal probability of features

- A small smoothing value (e.g., 0.001) is used for unseen feature values to avoid zero probabilities.
- The class with the highest posterior is selected as the predicted label.

Coding Environment

All code implementations and experiments were performed using the following software and hardware configuration:

- **Programming Language:** Python 3.10
- **Integrated Development Environment (IDE):** Visual Studio Code
- **Operating System:** Windows 11 (64-bit)
- **Processor:** 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
- **RAM:** 16 GB

This environment provided a stable and efficient setup for developing and testing the algorithms implemented.

Implementation Details

The algorithms were implemented in **Python**, utilizing libraries such as `pandas`, `numpy`, and `scikit-learn` for data manipulation and metric evaluation.

Preprocessing Steps:

- **Data Loading:** Datasets were fetched using the `ucimlrepo` Python package.
- **Encoding:** All categorical features were label encoded to convert them into numerical form suitable for computation.
- **Handling Missing Values:** Datasets with missing values were either excluded or pre-cleaned depending on availability.
- **Splitting:** Each dataset was randomly split into 80% training and 20% testing subsets using a consistent random seed for reproducibility.

Both algorithms were evaluated using repeated runs (10 cycles) to average out randomness and ensure robustness of the performance metrics.

Evaluation Metrics

To assess the performance of the classification algorithms, we used five standard evaluation metrics. These metrics provide a comprehensive view of the models' effectiveness across different aspects such as correctness, sensitivity, and balance.

- **Accuracy:**
  Accuracy is the ratio of correctly predicted instances to the total instances:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

  It gives a general indication of how often the classifier is correct.

- **Precision:**
  Precision measures the proportion of true positive predictions among all positive predictions:

$$\text{Precision} = \frac{TP}{TP + FP}$$

  It is useful when the cost of false positives is high.

- **Recall:**
  Recall (also known as sensitivity or true positive rate) is the proportion of true positives identified correctly out of all actual positives:

$$\text{Recall} = \frac{TP}{TP + FN}$$

  It is crucial when the cost of missing positive instances is high.

- **F1-Score:**
  The F1-score is the harmonic mean of precision and recall, providing a balance between the two:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

  It is especially useful for imbalanced datasets.

- **ROC AUC (Receiver Operating Characteristic - Area Under Curve):**
  AUC measures the area under the ROC curve, which plots the true positive rate against the false positive rate at various threshold settings. A higher AUC indicates better model discrimination between classes.

Dataset Used

Dataset	Instances	Features	Target Classes
Iris	150	4	3
Abalone	4177	8	2
Mushroom	8124	22	2
Automobile	205	25	7
Breast Cancer	569	30	4
Car Evaluation	1728	6	4
Heart Disease	303	13	3
Wine	178	13	3
Wine Quality	4898	11	11
Solar Flare	1389	10	3
Adult	48842	14	2
Bank Marketing	45211	16	2

Table 1: Summary of datasets used in the experiments.

These datasets are taken from University of California Irvine Machine Learning repository.
Link: https://archive.ics.uci.edu/

Results

The performance of both Decision Tree and Naive Bayes classifiers was evaluated on three datasets: Iris, Heart Disease, and Mushroom. Each algorithm was executed 10 times per dataset to reduce the impact of random data splitting. The average values of five evaluation metrics were recorded.

Dataset	**Algorithm**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC**
Iris	Decision Tree	0.8519	0.8889	0.8519	0.8460	0.8633
	Naive Bayes	0.8889	0.9333	0.8721	0.8571	0.8952
Abalone	Decision Tree	0.2139	0.2054	0.2139	0.2072	0.5372
	Naive Bayes	0.0999	0.0869	0.1107	0.0797	0.5326
Mushroom	Decision Tree	1.0000	1.0000	1.0000	1.0000	1.0000
	Naive Bayes	0.9219	0.9316	0.9202	0.9163	0.9163
Automobile	Decision Tree	0.6944	0.7061	0.6944	0.6852	0.6509
	Naive Bayes	0.9444	0.7346	0.7333	0.7234	0.8595
Breast Cancer	Decision Tree	0.6847	0.6001	0.0833	0.1463	0.5283
	Naive Bayes	0.3243	0.1622	0.5000	0.2449	0.5000
Car Evaluation	Decision Tree	0.9783	0.9787	0.9783	0.9784	0.9732
	Naive Bayes	0.8645	0.8371	0.6444	0.6993	0.7888
Heart Disease	Decision Tree	0.4667	0.4690	0.4667	0.4678	0.5319
	Naive Bayes	0.5333	0.2352	0.2551	0.2398	0.5697
Wine	Decision Tree	0.3529	0.3851	0.3529	0.3243	0.4998
	Naive Bayes	0.8824	0.8788	0.8860	0.8778	0.9151
Wine Quality	Decision Tree	0.5922	0.5873	0.5922	0.5887	0.6178
	Naive Bayes	0.1169	0.1848	0.2233	0.0915	0.5428
Solar Flare	Decision Tree	0.9931	0.0000	0.0000	0.0000	0.4983
	Naive Bayes	0.8125	0.2230	0.2190	0.2197	0.5526
Adult	Decision Tree	0.5845	0.5587	0.6123	0.5431	0.6457
	Naive Bayes	0.4305	0.3271	0.3405	0.3270	0.5599
Bank Marketing	Decision Tree	0.3654	0.3578	0.3598	0.3611	0.3584
	Naive Bayes	0.8254	0.4705	0.4852	0.4739	0.4852

Table 2: Performance comparison of Decision Tree and Naive Bayes classifiers across datasets.

Result analysis

Table 2 presents a comprehensive performance comparison between the Decision Tree and Naive Bayes classification algorithms over ten benchmark datasets, evaluated using five metrics: Accuracy, Precision, Recall, F1-Score, and AUC (Area Under the ROC Curve).

Overall Performance

In general, Naive Bayes outperforms Decision Tree in most datasets, particularly in terms of Accuracy and AUC, which are critical for evaluating classification effectiveness. However, the performance varies significantly across datasets, highlighting the importance of data characteristics in algorithm selection.

Dataset-wise Observation

- **Iris:** Naive Bayes slightly outperforms Decision Tree across all metrics, indicating its suitability for small, well-separated datasets.
- **Abalone:** Both algorithms perform poorly, suggesting the dataset may have high complexity or noise. Decision Tree edges out in all metrics except AUC.
- **Mushroom:** Decision Tree achieves perfect classification (1.0 in all metrics), while Naive Bayes also performs very well. This indicates that both algorithms handle this dataset effectively, but the Decision Tree is ideal when exact rule-based classification is feasible.
- **Automobile:** Naive Bayes shows significantly better performance, particularly in Accuracy (0.9444 vs. 0.6944) and AUC (0.8595 vs. 0.6509), suggesting that it models the probabilistic relationships well.
- **Breast Cancer:** Decision Tree has higher Accuracy and Precision, but extremely low Recall (0.0833) and F1-Score (0.1463), implying it fails to capture positive cases. Naive Bayes offers a more balanced, though still suboptimal, performance.
- **Car Evaluation:** Decision Tree performs exceptionally well across all metrics, especially F1-Score (0.9784), which indicates it effectively captures complex decision boundaries. Naive Bayes is reasonably good but notably weaker in Recall and F1.
- **Heart Disease:** Both classifiers perform poorly, with Naive Bayes slightly better overall. The low values across all metrics suggest this dataset is challenging for both algorithms.
- **Wine:** Naive Bayes significantly outperforms Decision Tree, especially in Accuracy (0.8824 vs. 0.3529) and AUC (0.9151 vs. 0.4992). This implies that the probabilistic approach handles the feature distribution of this dataset better.
- **Wine Quality:** Decision Tree shows much stronger results compared to Naive Bayes, especially in Accuracy (0.5922 vs. 0.1169), indicating its robustness in handling real-valued, possibly non-Gaussian features.

- **Solar Flare:** Decision Tree again outperforms Naive Bayes in Accuracy (0.9931 vs. 0.8125), though both perform poorly in Recall and F1, suggesting class imbalance or difficulty in positive class identification.
- **Adult:** The Decision Tree performs much better than Naive Bayes for the Adult dataset, indicating that it handles the feature relationships in this dataset more effectively.
- **Bank Marketing:** Naive Bayes outperforms Decision Tree significantly on this dataset. This suggests the features in Bank Marketing might be more conditionally independent or favor probabilistic modeling.

Metric-Specific Insights

Accuracy

Naive Bayes often achieves higher Accuracy on datasets with well-separated classes and probabilistic feature relationships (e.g., Iris, Wine, Automobile). However, Decision Tree can sometimes artificially inflate Accuracy (e.g., Solar Flare) by predicting only the majority class.

Precision

Decision Tree often yields high Precision, especially on imbalanced datasets, where it tends to favor majority class predictions. Naive Bayes maintains moderately good Precision in balanced datasets but suffers when class overlap or feature dependency increases.

Recall

Naive Bayes consistently provides non-zero Recall, even in difficult datasets (e.g., Solar Flare, Breast Cancer), while Decision Tree sometimes fails completely (e.g., Recall = 0 in Solar Flare). This makes Naive Bayes more reliable in identifying minority classes, despite lower overall Accuracy.

F1 Score

F1-Score reveals Decision Tree's inability to generalize on imbalanced datasets, especially when Recall is zero (e.g., Solar Flare, Breast Cancer). Naive Bayes tends to retain a baseline F1-Score even in difficult datasets, reflecting its more balanced trade-off.

AUC (Area Under Curve)

Naive Bayes usually outperforms Decision Tree in AUC (e.g., Wine, Automobile), indicating a better ranking of predictions in binary/multiclass tasks. Decision Tree performs well only in a few datasets (e.g., Mushroom, Car Evaluation) where the decision boundaries are sharp and feature splits are informative.

Conclusion

The comparative results across all twelve datasets reinforce that no single classifier is universally best performance is highly dataset-specific.

Naive Bayes is more consistent across datasets, especially those with probabilistic relationships, balanced classes, and structured numeric features. It shows better Recall and AUC, making it suitable when false negatives are costly.

Decision Tree, on the other hand, can excel dramatically in structured datasets with clear decision boundaries (e.g., Mushroom, Car Evaluation) but can overfit and fail catastrophically on imbalanced or noisy data (e.g., Breast Cancer, Solar Flare).

In practical applications, it is essential to analyze dataset characteristics—especially class distribution, feature type, and noise—before selecting a classifier. In many cases, ensemble methods like Random Forest or hybrid models could offer better performance than either algorithm alone.