University of Dhaka

Department of Computer Science and Engineering

CSE-4255 : Introduction to Data Mining and Warehousing Lab

Lab Report: Comparative Analysis of Classification Algorithms (Decision Tree and Naïve Bayes)

Submitted By:

Name : Ahaj Mahhin Faiak

Roll No : 01

Submitted On:

JUNE 25, 2025

Submitted To:

Dr. Chowdhury Farhan Ahmed

Md. Mahmudur Rahman

Contents

1 Introduction

Data classification is a fundamental task in data mining that categorizes data points into predefined classes based on their attributes. The objective is to construct a model from a labeled training dataset which can then be used to accurately predict the class of new, unclassified data. This is also called supervised learning. The efficacy of this predictive task is heavily dependent on the underlying algorithm used to build the classification model.

In this assignment. I presents a comparative analysis of two widely recognized classification algorithms: the Decision Tree and the Naïve Bayes classifier. While both are used for predictive modeling, they originate from different theoretical paradigms. Decision Trees employ a rule-based, hierarchical structure, whereas Naïve Bayes classifiers utilize principles of probability. Analyzing the process of this two algorithm provides with a deep knowledge on how they work on different types of data in real world.

Decision Tree Classifier

A Decision Tree is a non-parametric supervised learning method that predicts the value of a target variable by learning simple decision rules inferred from the data features. It employs a tree-like structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node holds a class label.

The construction of the tree, known as induction, is a recursive process of partitioning the data. At each node, an attribute selection measure is used to identify the attribute that most effectively splits the data into purer subsets. A prominent measure for this purpose is **Information Gain**, which is based on the concept of entropy. The information gain, $Gain(A)$, of an attribute A is calculated as:

$$Gain(A) = Info(D) - Info_A(D)$$

Where:

- $Info(D)$ is the entropy of the original dataset D, calculated as:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

Here, m is the number of classes and p_i is the probability of a tuple in D belonging to class C_i.

- $Info_A(D)$ is the expected information (entropy) after the dataset D has been partitioned by attribute A:

$$Info_A(D) = \sum_{j \in Values(A)} \frac{|D_j|}{|D|} xInfo(D_j)$$

Here, $Values(A)$ is the set of all possible values for attribute A, and D_j is the subset of D for which attribute A has value j. The attribute with the highest information gain is chosen as the splitting criterion. While interpretable, Decision Trees can be susceptible to overfitting, where the model captures noise in the training data, potentially leading to lower accuracy on unseen data.

Naive Bayes Classifier

In contrast, the Naïve Bayes classifier is a probabilistic model grounded in **Bayes' Theorem**. It calculates the probability of a data instance belonging to a specific class given a set of attributes. For a data instance $\mathbf{X}$ with attributes $(x_1, x_2, \ldots, x_n)$ and a set of classes $(C_1, C_2, \ldots, C_m)$, the classifier predicts that $\mathbf{X}$ belongs to the class C_i if and only if:

$$P(C_i|\mathbf{X}) > P(C_j|\mathbf{X}) \quad \text{for } 1 \leq j \leq m, j \neq i$$

The posterior probability, $P(C_i|\mathbf{X})$, is calculated using Bayes' Theorem:

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

Where:

- $P(C_i|\mathbf{X})$ is the posterior probability of class C_i given the attribute vector $\mathbf{X}$.
- $P(\mathbf{X}|C_i)$ is the likelihood of observing $\mathbf{X}$ given that it belongs to class C_i.
- $P(C_i)$ is the prior probability of class C_i.

- $P(\mathbf{X})$ is the prior probability of the predictor vector $\mathbf{X}$.

The algorithm's "naive" assumption is the class-conditional independence of attributes. This presumes that the effect of an attribute's value on a given class is independent of the values of other attributes. This simplifies the computation of the likelihood to:

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

Consequently, this simplifies the classification process, making the algorithm highly efficient and particularly effective for high-dimensional data, such as in text classification.

2 Implementation Details

Naive Bayes Classifier Implementation Details

This custom `NaiveBayesClassifier` class supports both **discrete** and **continuous** features. It is built from scratch using probability theory and does not rely on external machine learning libraries.

Class Components

- `self.alpha`: Smoothing parameter used for Laplace smoothing to avoid zero probabilities.
- `self.label_probs`: Stores the log prior probabilities for each class.
- `self.cond_probs`: Stores log conditional probabilities for discrete attributes.
- `self.mean_store, self.std_store`: Stores mean and standard deviation for each continuous attribute per class.
- `self.is_continuous_map`: Tracks whether each attribute is continuous.
- `self.labels, self.attributes`: Lists of class labels and feature names.

Continuous Attribute Detection

To determine whether an attribute is continuous, the function `_is_continuous()` checks:

- If the attribute is numeric.
- If the number of unique values is large enough relative to the number of rows (greater than 1%).

Gaussian Log Probability

For continuous attributes, probabilities are estimated using the Gaussian (Normal) distribution:

$$\log P(x|\mu, \sigma) = -0.5 \log(2\pi) - \log(\sigma) - \frac{(x-\mu)^2}{2\sigma^2}$$

Training Phase: `fit(X_train, y_train)`

- The function extracts all attribute names and class labels.
- It computes the prior probability of each class using Laplace smoothing:

$$P(C_i) = \frac{\text{count}(C_i) + \alpha}{\text{total} + \alpha \cdot \text{num\_labels}}$$

- For each attribute:
  - If continuous, it stores the `mean` and `standard deviation` for each class.
  - If discrete, it computes and stores the smoothed log-probabilities for each attribute value given the class.

Prediction Phase: `predict_single()` **and** `predict()`

- `predict_single()` calculates the total log-probability of each class given a row, and returns the class with the highest probability.
- For each attribute:
  - If continuous: uses Gaussian probability.
  - If discrete: uses stored log-probabilities.
- `predict()` applies `predict_single()` on every row of the test dataset.

Advantages

- Handles both continuous and discrete data types.
- Uses logarithmic probabilities to prevent numerical underflow.
- Efficient due to pre-computation of statistics.
- Can generalize well even with small datasets due to smoothing.

Decision Tree Classifier Implementation Details

This custom `DecisionTreeClassifier` builds a decision tree using the **ID3 algorithm** with support for both **discrete** and **continuous** attributes.

Attribute Type Detection

The function `_is_continuous()` determines whether a feature is continuous:

- The feature must be numeric.
- It must have a sufficiently high ratio of unique values (>1%).

Entropy and Information Gain

The algorithm uses **Information Gain** to choose the best attribute for splitting:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

$$Gain(D, A) = Info(D) - Info_A(D)$$

$$Info_A(D) = \sum_{j} \frac{|D_j|}{|D|} \cdot Info(D_j)$$

Tree Node Representation

Each tree node is represented using a `Node` class containing:

- `split_attribute`: Attribute chosen for splitting.
- `attribute_selection_criteria`: Stores either a discrete value or a continuous range.
- `children`: Dictionary of child nodes.
- `isLeaf`: Boolean indicating if the node is a leaf.
- `returning_class`: Class to return if it's a leaf.

Splitting Strategy

- For **continuous attributes**, all possible midpoints are computed from sorted unique values.
- For **discrete attributes**, the gain is calculated based on each unique value.

- The attribute with the highest gain is selected for the current split.
- If multiple splits are not allowed, the used attribute is removed from further consideration.

Stopping Conditions

The recursion stops when:

- All instances at the node have the same label.
- There are no more attributes left to split.
- No split improves information gain.

Handling Continuous Splits

- The best split value is stored in a `ContinuousAttributeSelectionCriteria` object.
- Two child nodes are created: one for values $\leq$ split, the other for $>$ split.

Handling Discrete Splits

- A `DiscreteAttributeSelectionCriteria` object is used to store the chosen split values.
- One child node is created for each unique value.

Prediction Strategy

To predict a test instance:

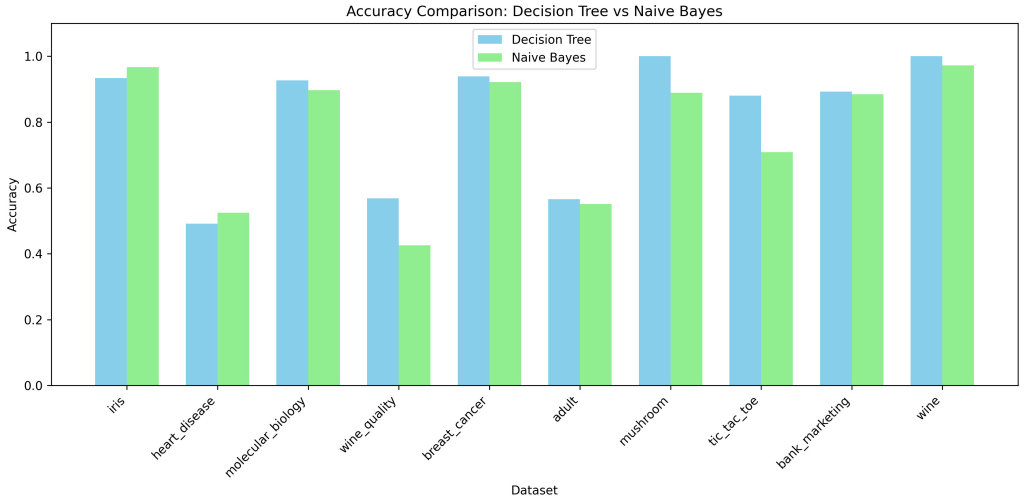
- The algorithm starts at the root and follows the child that satisfies the condition.
- If a missing or unseen attribute value is encountered, it returns the **majority class** of the subtree.

Advantages

- Supports both continuous and categorical data.
- Easy to interpret decision logic.
- Handles missing branches with majority voting fallback.

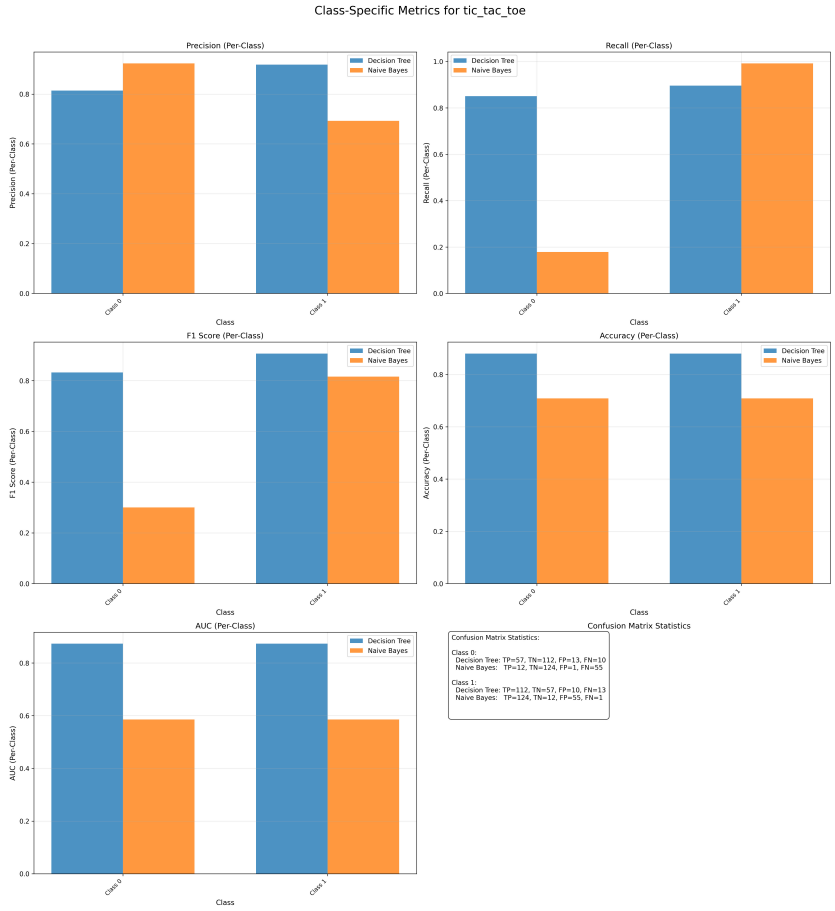
3 Experimental Result

Here I present the experimental results of Decision Tree and Naive Bayes Classifier on different datasets.

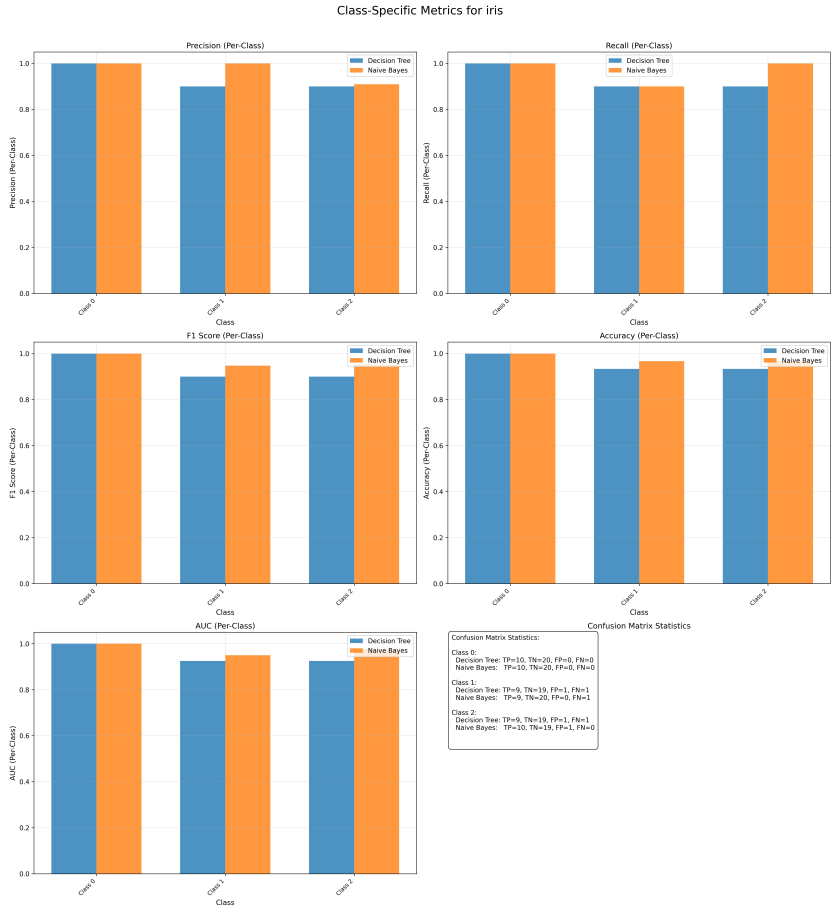


(a) Overall Accuracy Comparison Graph

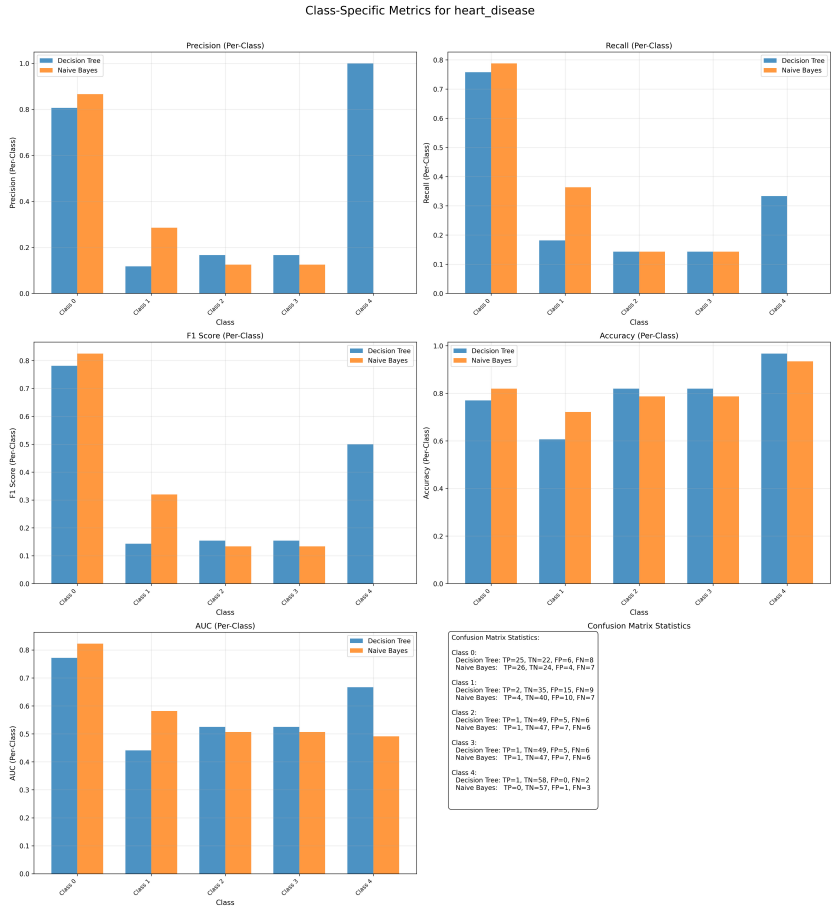
Dataset: Tic-Tac-Toe



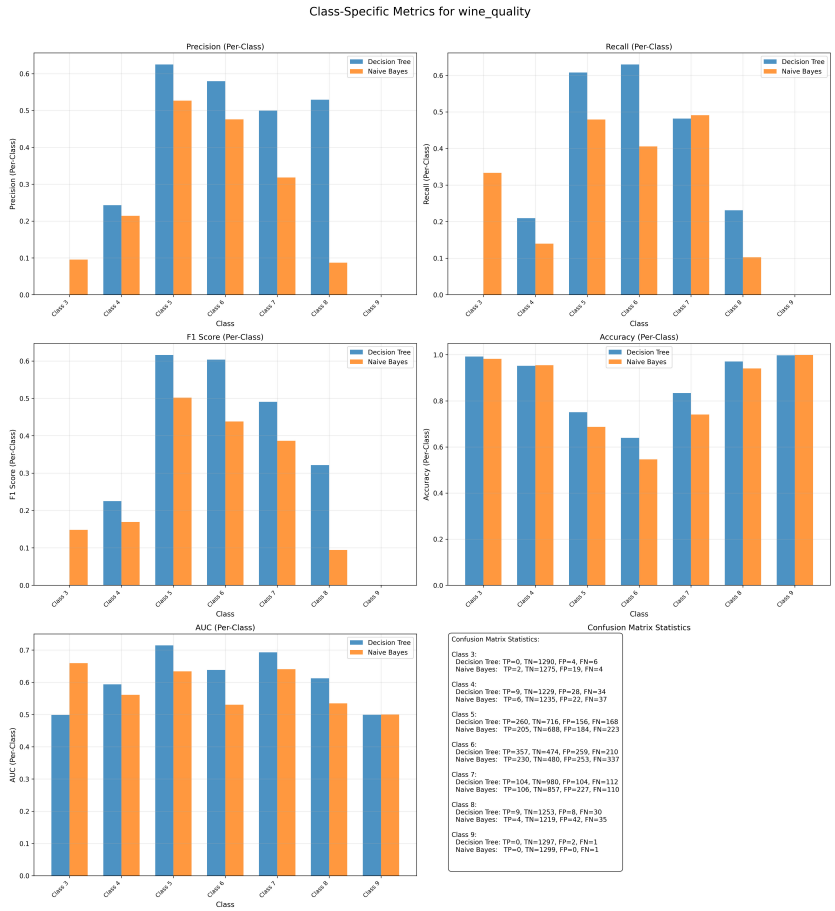
Dataset: Iris



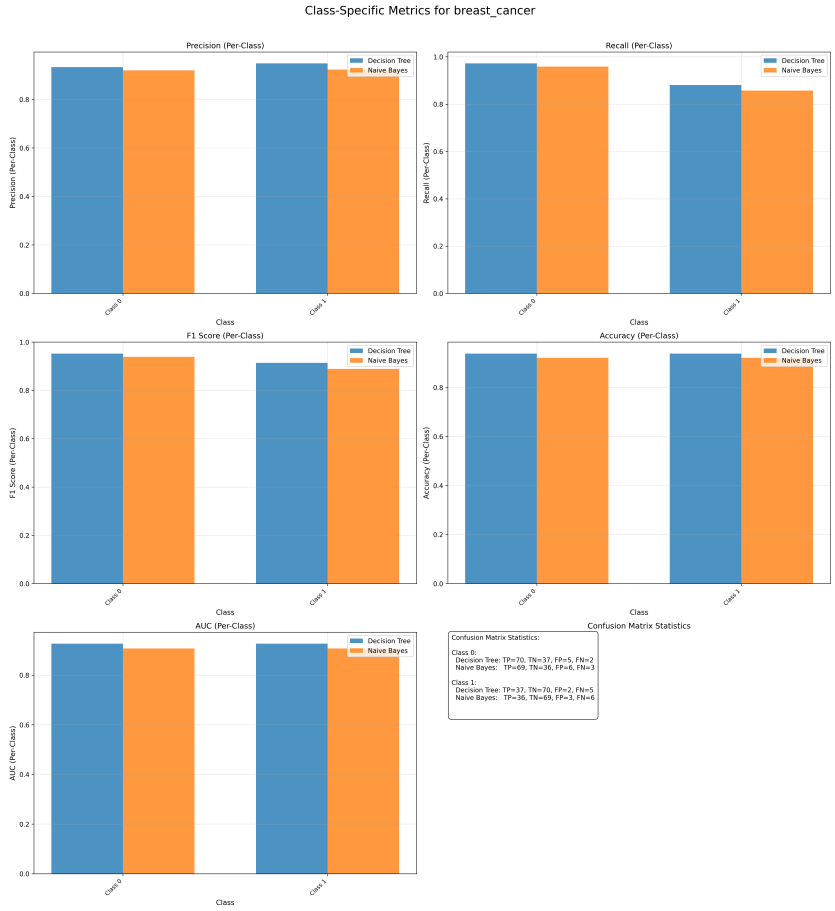
Dataset: Heart Disease



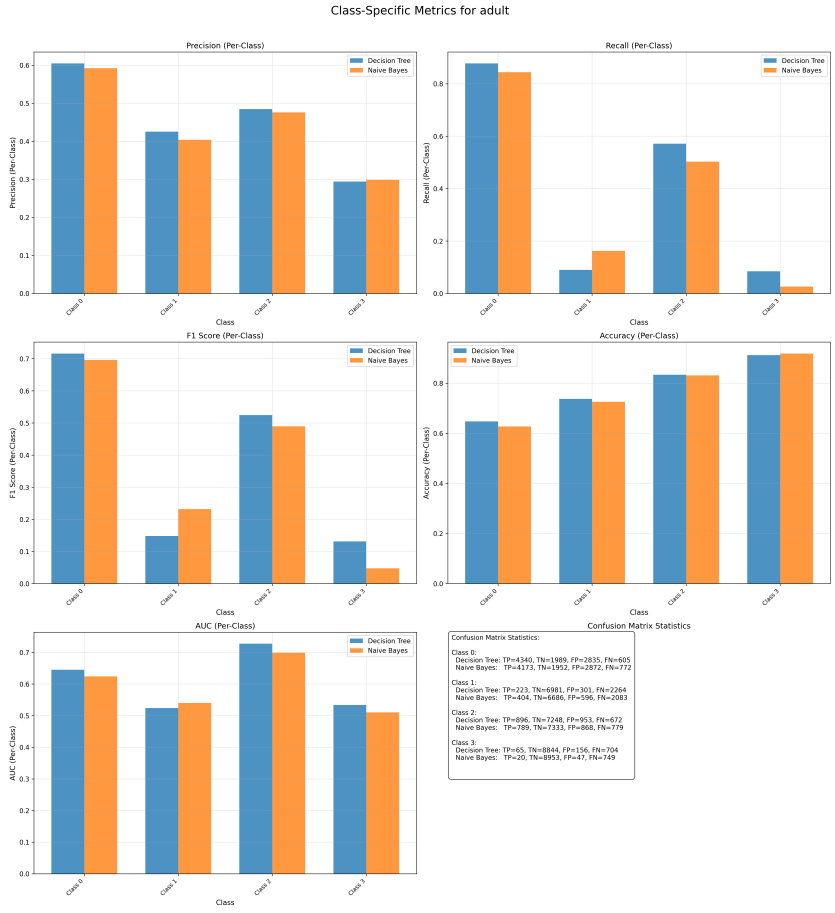
Dataset: Wine Quality



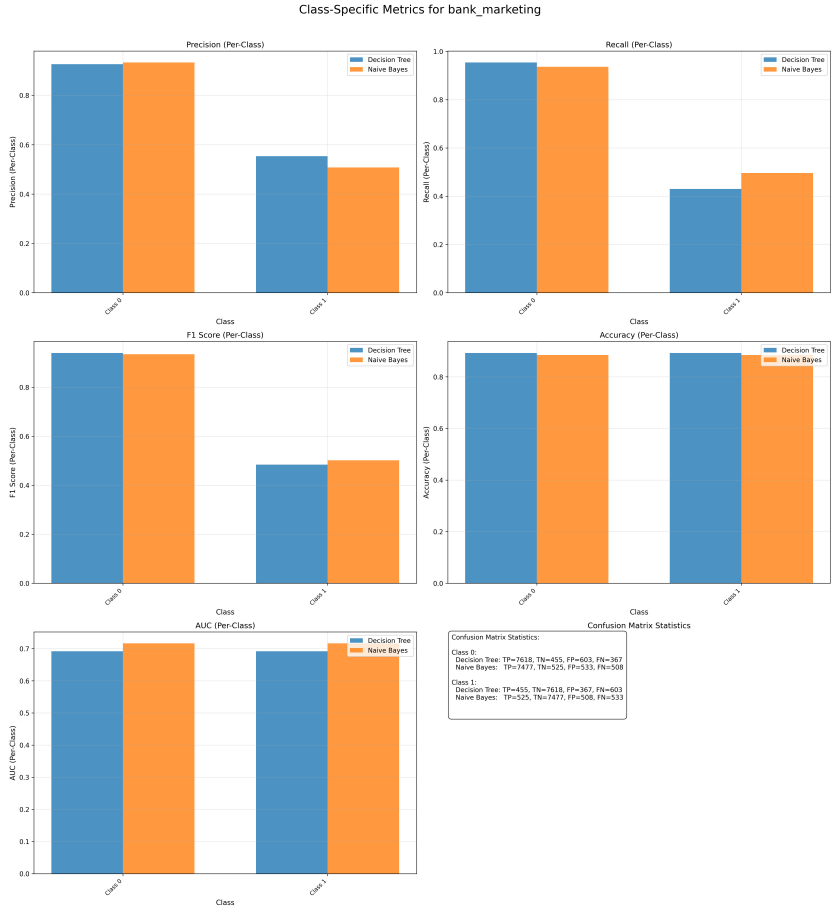
Dataset: Breast Cancer



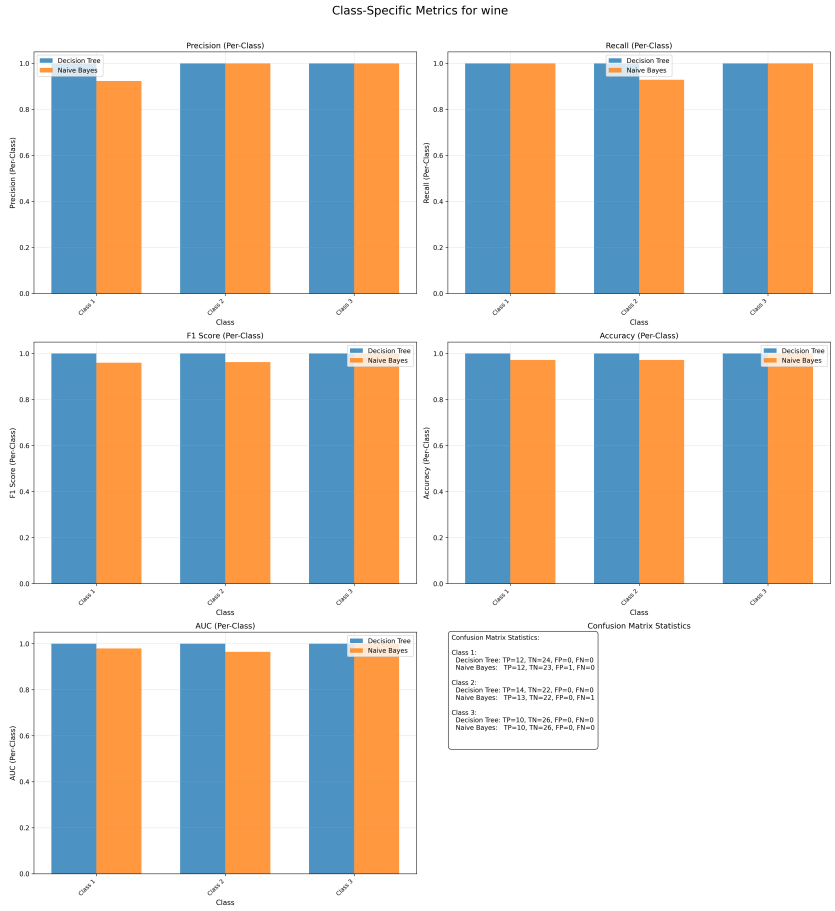
Dataset: Adult



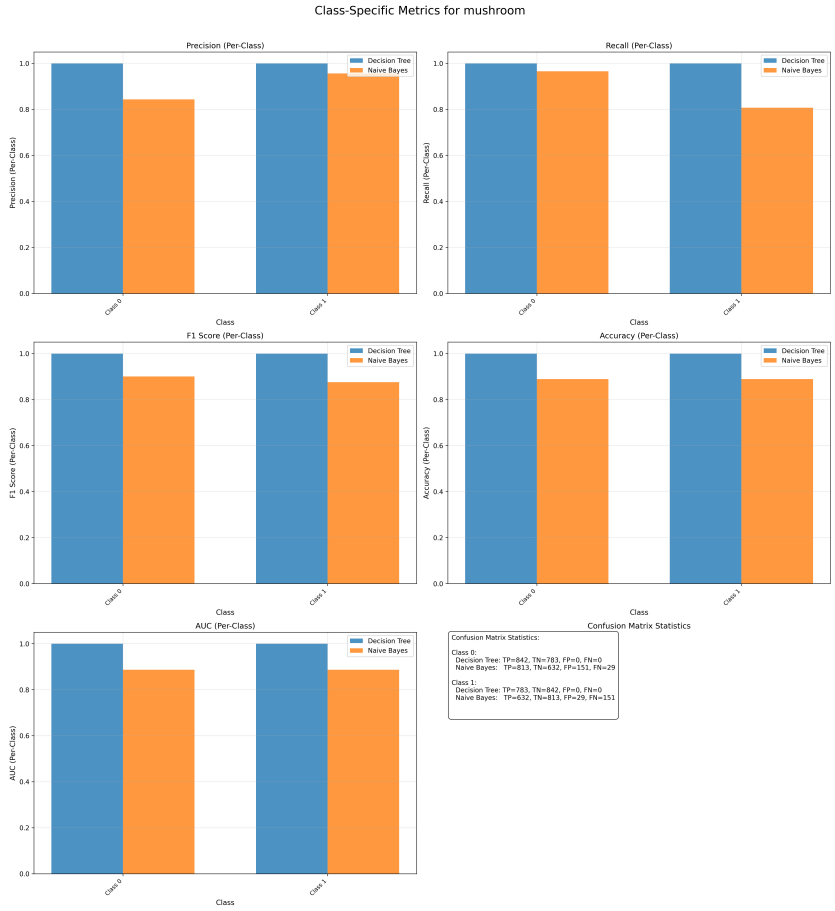
Dataset: Bank Marketing



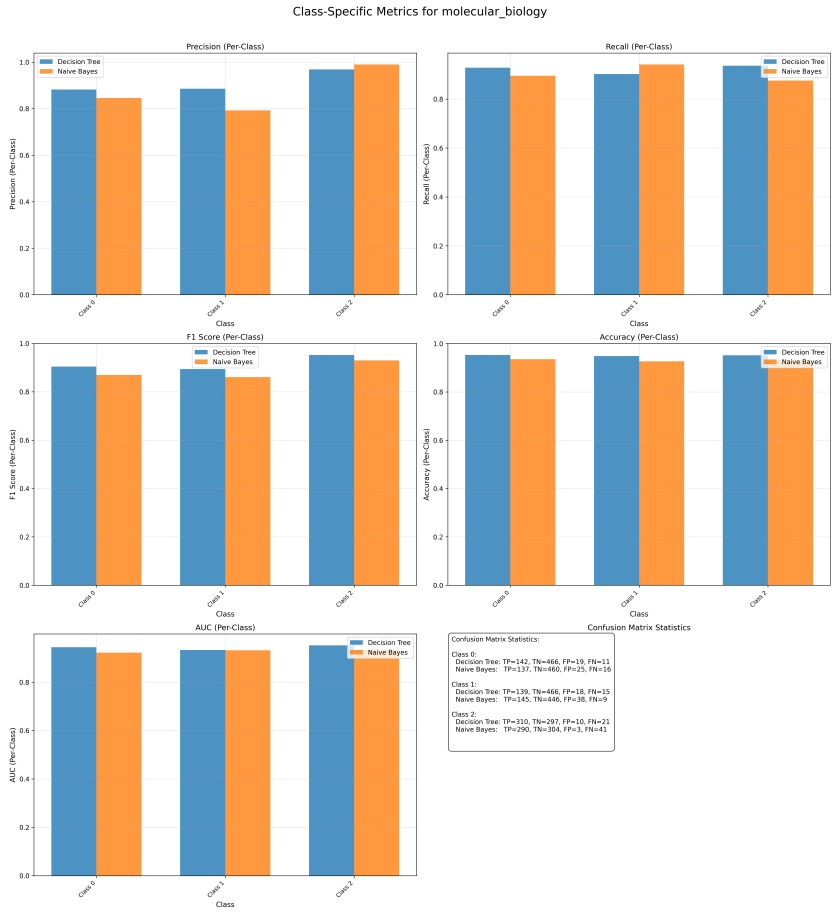
Dataset: Wine



Dataset: Mushroom



Dataset: Molecular Biology



4 Conclusion

This study presents a comparison between two popular classification algorithms: Decision Tree and Naïve Bayes. The results indicate that Decision Tree generally performs better than Naïve Bayes in key evaluation metrics such as accuracy, precision, recall, and F1-score. This suggests that Decision Tree is more effective for making correct predictions across different datasets.

Although Naïve Bayes is faster and easier to implement, its predictive performance is usually lower compared to Decision Tree. Therefore, in practical applications where accuracy is important, Decision Tree is often the better choice.