INDIAN INSTITUTE OF TECHNOLOGY MADRAS ZANZIBAR

School of Science and Engineering

Z5007: Programming and Data Structures
M.Tech Data Science & Artificial Intelligence

PROJECT PROGRESS REPORT (MILESTONE 2)

Naive Bayes Classifier from Scratch for Income Prediction

Submitted by:

Student: Guggella Chaitanya Reddy (ZDA25M010)
Email: zda25m010@iitmz.ac.in

Instructor: Innocent Nyalala

Submission Date: December 23, 2025

Contents

1 Progress Summary

This project implements a Naive Bayes classifier **from scratch** to predict income categories using the Adult dataset. The implementation avoids the use of machine learning libraries such as `scikit-learn` and instead focuses on manual construction of the algorithm and its supporting data structures.

As part of Milestone 2, the model was trained and evaluated on the **full Adult dataset consisting of 32,561 instances**, using an **80/20 train–test split** with a fixed random seed. The system successfully runs end-to-end and produces meaningful classification results.

The project is approximately **70% complete**, with all core algorithmic components implemented. Remaining work primarily involves extended evaluation, benchmarking, and final documentation.

Component Status

Component	Status
OpenAddressingHashMap	Complete
CountMatrix	Complete
RunningStats	Complete
Data Loading & Encoding	Complete
Naive Bayes Training	Complete
Naive Bayes Prediction	Complete
Evaluation Metrics	Partial
Final Report & Benchmarking	Pending

2 System Architecture

The system follows a simple pipeline:

1. **Data Loading (main.py):** Read `data/adult.csv`, auto-detect delimiter (tab/comma).
2. **Preprocessing (main.py):** Identify numeric vs categorical columns, handle missing values (`?`), encode categories to integer IDs, encode labels.
3. **Training (nb_model/naive_bayes.py):**
   - Compute class priors.
   - Numeric features: update `RunningStats` and store mean/variance per class.
   - Categorical features: update `CountMatrix` counts.
4. **Inference (nb_model/naive_bayes.py):** Compute class log-scores using log priors + log likelihoods, choose best class.
5. **Diagnostics (main.py):** Print log-score sample, accuracy, and hash map collision/probe/resize statistics; save to `outputs/sample_run.txt`.

3 Technical Details

3.1 Custom Data Structures Implemented

To satisfy the requirement of implementing at least two custom data structures, the following structures were developed and actively used in the system:

3.1.1 OpenAddressingHashMap

A hash map based on open addressing with linear probing was implemented from scratch. It supports insertion, lookup, deletion, containment checks, and incremental updates. The hash map dynamically resizes when a specified load factor threshold is exceeded.

In addition to standard operations, the structure tracks:

- Number of collisions
- Number of probe operations
- Current load factor
- Number of resize operations

These statistics are printed during execution to demonstrate correct collision handling and resizing behavior under full-dataset load.

3.1.2 CountMatrix

The CountMatrix is a higher-level data structure built on top of the custom hash map. It stores frequency counts for categorical features using composite keys of the form:

$$(\text{class}, \text{feature index}, \text{value id})$$

This structure is used to compute likelihoods for categorical features in the Multinomial Naive Bayes component.

3.1.3 RunningStats

RunningStats maintains online estimates of the mean and variance using Welford's algorithm. It is used to support Gaussian Naive Bayes. A small variance floor is applied to prevent numerical instability when a feature has near-zero variance within a class.

3.2 Naive Bayes Algorithm

A **hybrid Naive Bayes classifier** was implemented:

- **Gaussian Naive Bayes** for continuous numeric features
- **Multinomial Naive Bayes** for categorical features

This hybrid approach models each feature type using an appropriate likelihood function. Class priors and feature likelihoods are estimated during training, and predictions are made by evaluating class scores during inference.

3.3 Log-Probability Computation

To avoid numerical underflow caused by multiplying many small probabilities, all probability computations are performed in the **log domain**. For a class c, the score is computed as:

$$\log P(c \mid x) = \log P(c) + \sum_i \log P(x_i \mid c)$$

Laplace smoothing is applied to categorical probabilities to ensure that probabilities never become zero, thereby avoiding undefined $\log(0)$ values. During execution, sample log-scores are printed to explicitly demonstrate log-domain computation.

3.4 Time Complexity Analysis

Let:

- N be the number of training samples
- d_n be the number of numeric features
- d_c be the number of categorical features
- C be the number of classes

Training Complexity:

$$O(N \cdot (d_n + d_c))$$

Prediction Complexity (per sample):

$$O(C \cdot (d_n + d_c))$$

Hash map operations run in expected $O(1)$ time.

4 Challenges and Solutions

4.1 Collision Handling in Hash Map

Linear probing can degrade performance at high load factors. This was addressed by resizing the hash table when the load factor exceeds a predefined threshold. Collision counts, probe counts, and resize operations are tracked and reported to validate correct behavior.

4.2 Numerical Underflow

Direct multiplication of probabilities leads to floating-point underflow. This was solved by computing all probabilities in log space and summing log-likelihoods instead.

4.3 Zero Variance in Gaussian Naive Bayes

When a numeric feature has zero variance within a class, Gaussian likelihood computation becomes unstable. A small variance epsilon was introduced to ensure numerical stability.

4.4 Handling Missing Values

The Adult dataset contains missing values represented by `?`. These were handled by:

- Replacing missing categorical values with a dedicated `MISSING` token
- Replacing missing numeric values with zero for this milestone

5 Results

The hybrid Naive Bayes classifier was trained and evaluated on the full Adult dataset. Using an 80/20 train–test split, the model achieved an accuracy of:

83.57%

This result demonstrates correct end-to-end functionality and validates the implemented data structures under full dataset load.

6 Remaining Work

The following tasks remain for the final milestone:

- Add additional evaluation metrics (precision, recall, F1-score)
- Benchmark performance against a scikit-learn Naive Bayes baseline
- Improve numeric feature imputation
- Prepare the final technical report, presentation slides, and demo

Updated Timeline

- Week 11–12: Full evaluation and metric expansion
- Week 12–13: Benchmarking and optimization
- Week 13: Final report writing and presentation preparation