Technical Report: Implementation of Naive Bayes Classifier from Scratch

Khamis K Haji (zda25m002)
Juweayria Farouk (zda25m003)
Instructor: Dr. Innocent Nyalala
Course: Z5007: Programming and Data Structures
Institution: IIT Madras Zanzibar

January 19, 2026

Contents

1 Problem Description and Motivation

Naive Bayes classifiers are fundamental probabilistic machine learning models widely used in text classification, medical diagnosis, spam filtering, and recommendation systems. Despite their simplicity and efficiency, many implementations rely on high-level libraries such as scikit-learn, which abstract away underlying mathematical and computational details.

1.1 Motivation

The primary objectives of this project were:

- To deepen understanding of Bayesian probability theory and its application to classification
- To gain hands-on experience in designing and implementing efficient data structures
- To address real-world challenges including numerical underflow, class imbalance, and scalability
- To benchmark custom implementations against industry-standard tools for performance evaluation
- To implement a custom hash table for storing model parameters

2 System Architecture and Design

2.1 Overall System Design

The system follows a modular architecture with the following components:

1. **Custom Data Structures** – Hash table with dynamic resizing and collision handling
2. **Core Classifiers** – Three Naive Bayes variants implemented as separate classes
3. **Preprocessing Pipeline** – Handles data loading, normalization, and splitting
4. **Testing Framework** – Unit, integration, and performance tests
5. **Evaluation Module** – Compares models using standard metrics

2.2 Class Structure

```
HashTable
 insert(key, value)
 search(key)
 _resize_if_needed()

BaseClassifier (Abstract)
 fit(X, y)
 predict(X)

GaussianNaiveBayes
 _mean_table: HashTable
 _variance_table: HashTable
 _gaussian_log_pdf()

BernoulliNaiveBayes
 _likelihood_table: HashTable
 alpha: float

MultinomialNaiveBayes
 feature_log_probs: ndarray
 alpha: float
```

2.3 Workflow

1. **Data Loading** – Wisconsin Breast Cancer dataset (569 samples, 30 features)
2. **Preprocessing** – Stratified train-test split (80:20)
3. **Training** – Fit model using training data
4. **Prediction** – Classify test samples
5. **Evaluation** – Compare with scikit-learn baseline

3 Data Structures and Algorithms Explained

3.1 Custom Hash Table Implementation

A hash table with separate chaining was implemented with the following features:

- Dynamic resizing when load factor ¿ 0.75
- SHA-256 hashing for key distribution
- Separate chaining for collision resolution

Methods:

- `insert(key, value)` – O(1) average, O(n) worst-case
- `search(key)` – O(1) average, O(n) worst-case
- `_resize_if_needed()` – O(n) when triggered

3.2 Gaussian Naive Bayes

3.2.1 Assumption

Features follow Gaussian distribution: $x_j \sim \mathcal{N}(\mu_{yj}, \sigma^2_{yj})$

3.2.2 Training

$$\mu_{yj} = \frac{1}{n_y} \sum_{x \in y} x_j$$
$$\sigma_{yj}^2 = \frac{1}{n_y} \sum_{x \in y} (x_j - \mu_{yj})^2 + \epsilon \quad (\epsilon = 10^{-9})$$

3.2.3 Prediction

$$P(y|x) \propto P(y) \prod_{j=1}^{m} P(x_j|y)$$
$$\log P(y|x) = \log P(y) + \sum_{j=1}^{m} \log P(x_j|y)$$
$$\log P(x_j|y) = -\frac{1}{2} \log(2\pi\sigma_{yj}^2) - \frac{(x_j - \mu_{yj})^2}{2\sigma_{yj}^2}$$

3.3 Bernoulli Naive Bayes

3.3.1 Assumption

Features are binary: $x_j \in \{0, 1\}$

3.3.2 Training

$$P(x_j = 1|y) = \frac{\text{count}(x_j = 1, y) + \alpha}{n_y + 2\alpha}$$
$$P(x_j = 0|y) = 1 - P(x_j = 1|y)$$

3.3.3 Prediction

$$\log P(y|x) = \log P(y) + \sum_{j=1}^{m} \log P(x_j|y)$$

3.4 Multinomial Naive Bayes

3.4.1 Assumption

Features represent frequency counts

3.4.2 Training

$$\log P(x_j|y) = \log \left(\frac{\text{count}(x_j, y) + \alpha}{\sum_{k=1}^{m} \text{count}(x_k, y) + m\alpha} \right)$$

3.4.3 Prediction

$$\log P(y|x) = \log P(y) + \sum_{j: x_j > 0} x_j \cdot \log P(x_j|y)$$

4 Complexity Analysis

4.1 Time Complexity

Operation	**GaussianNB**	**BernoulliNB**	**MultinomialNB**
Training	$O(n \cdot m \cdot c)$	$O(n \cdot m \cdot c)$	$O(n \cdot m \cdot c)$
Prediction (per sample)	$O(m \cdot c)$	$O(m \cdot c)$	$O(m \cdot c)$

Table 1: Time complexity analysis where n = samples, m = features, c = classes

4.2 Space Complexity

Component	**Space Complexity**
Hash Tables (Gaussian/Bernoulli)	$O(c \cdot m)$
Feature Log Probabilities (Multinomial)	$O(c \cdot m)$
Overall	$O(c \cdot m)$

4.3 Hash Table Complexity

Operation	**Average Case**	**Worst Case**
Insert	$O(1)$	$O(n)$
Search	$O(1)$	$O(n)$
Resize	$O(n)$	$O(n)$

5 Experimental Results and Analysis

5.1 Dataset Specifications

- **Dataset:** Wisconsin Breast Cancer Dataset
- **Samples:** 569
- **Features:** 30
- **Classes:** 2 (Benign/Malignant)
- **Class Distribution:** 62.7% Benign, 37.3% Malignant

Model	Accuracy	Precision	Recall	F1-Score
GaussianNB (Ours)	96.46%	97.50%	92.86%	95.16%
GaussianNB (scikit-learn)	97.37%	97.56%	93.02%	95.22%
BernoulliNB (Ours)	98.25%	98.25%	98.25%	98.25%
MultinomialNB (Ours)	89.38%	89.47%	89.38%	89.42%

Table 2: Performance comparison on Wisconsin Breast Cancer dataset

Operation	Our Implementation	scikit-learn	Ratio
GaussianNB Training	0.0027s	0.0014s	1.93× slower
BernoulliNB Training	0.0004s	0.00015s	2.67× slower
MultinomialNB Training	0.002s	0.0023s	**1.15× faster**
Average Training	0.0017s	0.0013s	1.31× slower
Prediction (per sample)	0.00008s	0.00005s	1.60× slower

Table 3: Execution time comparison

5.2 Performance Metrics

5.3 Speed Comparison

5.4 Key Findings

1. **Accuracy:** Custom implementations achieved comparable accuracy to scikit-learn, with GaussianNB within 0.91%
2. **Speed:** Slightly slower training times due to Python-level loops
3. **Numerical Stability:** Log-space calculations effectively prevented underflow
4. **Class Imbalance:** Stratified splitting ensured fair evaluation across classes

5.5 Ablation Study

- Removing log-space calculations caused underflow for datasets with ¿10 features
- Removing variance smoothing resulted in division-by-zero errors
- Without Laplace smoothing, models performed poorly on unseen features

6 Conclusion

This project successfully implemented three Naive Bayes classifiers from scratch, demonstrating:

1. **Theoretical Understanding:** Deepened knowledge of Bayesian methods and probability theory
2. **Practical Skills:** Designed custom data structures and handled real-world data challenges
3. **Performance:** Achieved accuracy comparable to scikit-learn with minor trade-offs in speed
4. **Reproducibility:** Fully documented code and mathematical derivations

References

1. Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective.* MIT Press.
2. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.* Springer.
3. Scikit-learn Developers. (2021). Naive Bayes Documentation. `https://scikit-learn.org`
4. Dua, D. & Graff, C. (2019). UCI Machine Learning Repository.
5. Manning, C. D. et al. (2008). *Introduction to Information Retrieval.* Cambridge University Press.

Appendix: GitHub Repository

Complete source code available at:
`https://github.com/fjuweariya-dotcom/NaiveBayes`