Implementation of Naive Bayes Classifier from Scratch

Technical Report Presentation

Khamis K Haji (zda25m002) Juweayria Farouk (zda25m003)

Instructor: Dr. Innocent Nyalala
Course: Z5007: Programming and Data Structures
Institution: IIT Madras Zanzibar

January 19, 2026

Motivation & Objectives

Why Implement from Scratch?

- Deepen understanding of Bayesian probability theory and classification
- Gain hands-on experience in designing efficient data structures
- Address real-world challenges:
  - Numerical underflow
  - Class imbalance
  - Scalability
- Benchmark against industry-standard tools (scikit-learn)
- Implement a custom hash table for storing model parameters

System Architecture Overview

Modular Design

1. **Custom Data Structures**
   - Hash table with dynamic resizing & collision handling
2. **Core Classifiers**
   - Gaussian, Bernoulli, Multinomial Naive Bayes
3. **Preprocessing Pipeline**
   - Data loading, normalization, splitting
4. **Testing Framework**
   - Unit, integration, performance tests
5. **Evaluation Module**
   - Compares models using standard metrics

Class Structure

Object-Oriented Design

- `BaseClassifier` (Abstract)
  - `fit(X, y)`
  - `predict(X)`
- `GaussianNaiveBayes`
  - `_mean_table`, `_variance_table` (HashTable)
- `BernoulliNaiveBayes`
  - `_likelihood_table` (HashTable)
- `MultinomialNaiveBayes`
  - `feature_log_probs` (ndarray)

Workflow

End-to-End Process

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

Custom Hash Table Implementation

Key Features

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

Complexity:

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

Gaussian Naive Bayes

Assumption & Training

- Assumes features follow Gaussian distribution:

$$x_j \sim \mathcal{N}(\mu_{yj}, \sigma_{yj}^2)$$

- Training:

$$\mu_{yj} = \frac{1}{n_y} \sum_{x \in y} x_j$$

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

where $\epsilon = 10^{-9}$

Bernoulli & Multinomial Naive Bayes

Bernoulli:

- Binary features: $x_j \in \{0, 1\}$
- Training with Laplace smoothing (α)

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

Multinomial:

- Frequency count features
- Training in log-space to prevent underflow

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

Complexity Analysis

Time & Space Complexity

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

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

Space Complexity: $O(c \cdot m)$

Experimental Results – Performance

Wisconsin Breast Cancer Dataset

Model	Accuracy	Precision	Recall	F1-Score
GaussianNB (Ours)	96.46%	97.50%	92.86%	95.16%
GaussianNB (sklearn)	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: Performance comparison on Wisconsin Breast Cancer dataset

Experimental Results – Speed

Execution Time Comparison

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
Prediction (per sample)	0.00008s	0.00005s	1.60× slower
Average Training	0.0017s	0.0013s	1.31× slower

Table: Speed comparison between implementations

Key Findings & Ablation Study

What We Learned

- **Accuracy:** Comparable to scikit-learn (within 0.91% for GaussianNB)
- **Speed:** Slightly slower due to Python-level loops
- **Stability:** Log-space calculations prevented underflow
- **Imbalance:** Stratified splitting ensured fair evaluation

Ablation Insights:

- Removing log-space → underflow
- Removing variance smoothing → division-by-zero errors
- Without Laplace smoothing → poor performance on unseen features

Conclusion & Future Work

Successfully Demonstrated:

1. Deepened theoretical understanding of Bayesian methods
2. Built practical skills in data structure design
3. Achieved performance comparable to scikit-learn
4. Ensured reproducibility with documented code & math

Future Work:

- Parallelize training using NumPy vectorization
- Extend to sparse data formats
- Implement online learning for streaming data
- Add support for categorical features

References & GitHub Repository

References

1. Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*.
2. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*.
3. Scikit-learn Naive Bayes Documentation
4. UCI Machine Learning Repository
5. Manning, C. D. et al. (2008). *Introduction to Information Retrieval*.

GitHub Repository

Complete source code available at:

`https://github.com/fjuweariya-dotcom/NaiveBayes`

Thank You!

Questions?