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

Implementation of Naïve Bayes Classifier from Scratch

Submitted by:
Student 1: Khamis K Haji (zda25m002)
Email: zda25m002@iitmz.ac.in
Student 2: Juweayria Farouk (zda25m003)
Email: zda25m003@iitmz.ac.in
Instructor: Dr. Innocent Nyalala
Report Date: December 15, 2025
Project Period: Week 6 - Week 10

Contents

1 Executive Summary

This progress report summarizes the work completed during Weeks 6-10 of the project "Designing and Implementing Naïve Bayes Classifier from Scratch." As of Week 10, we have successfully implemented core data structures, completed the Gaussian Naïve Bayes classifier, and made significant progress on the Multinomial and Bernoulli variants. The project is proceeding according to schedule with minor adjustments to the original timeline.

Key accomplishments include:

- Development environment setup and repository initialization
- Implementation of custom hash table with collision handling
- Complete implementation of Gaussian Naïve Bayes classifier
- Complete implementation of Bernoulli Naive Bayes Classifier
- Partial implementation of Multinomial
- Data preprocessing pipeline for the Breast Cancer Wisconsin dataset
- Initial testing framework and unit tests

The project remains on track to meet all proposed objectives, with performance metrics aligning with initial expectations.

2 Project Status Overview

2.1 Timeline Comparison

Week	Planned Tasks	Actual Progress
Week 6	Proposal submission, environment setup	**Completed:** Proposal submitted, Python environment configured, GitHub repository created
Week 7	Implement data structures, load and preprocess data	**Completed:** Hash table with separate chaining implemented, data loading module created
Week 8	Begin algorithm implementation	**Completed:** Gaussian and Bernoulli Naïve Bayes fully implemented, basic probability calculations working
Week 9	Complete algorithm implementation, basic testing	**Partially Completed:** Multinomial variants 50% complete, unit tests implemented
Week 10	Submit progress report, refine implementation	**In Progress:** Progress report preparation, debugging and optimization ongoing

Table 1: Planned vs. Actual Progress

2.2 Completion Metrics

Component	Planned Completion	Actual Completion
Development Environment	Week 6	100%
Core Data Structures	Week 7	100%
Gaussian Naïve Bayes	Week 8	100%
Multinomial Naïve Bayes	Week 9	50%
Bernoulli Naïve Bayes	Week 9	100%
Data Preprocessing	Week 7	100%
Unit Testing	Week 9	70%
Integration Testing	Week 10	50%
Overall Progress	**Week 10**	**84%**

Table 2: Component Completion Status

3 Technical Progress Details

3.1 Completed Components

3.1.1 Development Environment Setup

- Python 3.9.7 installed with virtual environment
- Required packages: NumPy 1.21.2, Pandas 1.3.3, Matplotlib 3.4.3
- Git repository initialized with proper branching strategy
- VS Code configured with Python extensions and linting

3.1.2 Custom Data Structures Implementation

Hash Table Implementation:

Listing 1: Hash Table Implementation Snippet

```
class HashTable :
    def __init__ ( self , size =101) :
        self . size = size
        self . table = [[] for _ in range ( size ) ]
    def _hash ( self , key ) :
        return hash ( key ) % self . size
    def insert ( self , key , value ) :
        index = self . _hash ( key )
        for i , (k , v ) in enumerate ( self . table [ index ]) :
            if k == key :
                self . table [ index ][ i ] = ( key , value )
                return
        self . table [ index ]. append (( key , value ) )
    def search ( self , key ) :
        index = self . _hash ( key )
        for k , v in self . table [ index ]:
            if k == key :
                return v
        return None
```

Features implemented:

- Division method hash function with prime number size
- Separate chaining for collision resolution
- Dynamic resizing when load factor ¿ 0.75
- Average O(1) lookup time achieved

3.1.3 Gaussian Naïve Bayes Classifier

Key implementation details:

Listing 2: Gaussian Naïve Bayes Implementation

```
import numpy as np

class GaussianNaiveBayesFromScratch:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.n_classes = len(self.classes)
        self.n_features = X.shape[1]

        self.class_priors = np.zeros(self.n_classes, dtype=np.float64)
        self.class_means = np.zeros((self.n_classes, self.n_features), dtype=np
            .float64)
        self.class_stds = np.zeros((self.n_classes, self.n_features), dtype=np.
            float64)

        epsilon = 1e-9 # Small value to prevent division by zero in std

        for i, c in enumerate(self.classes):
            X_c = X[y == c]
            self.class_priors[i] = X_c.shape[0] / X.shape[0]

            for j in range(self.n_features):
                self.class_means[i, j] = np.mean(X_c[:, j])
```

```
                self.class_stds[i, j] = np.std(X_c[:, j]) + epsilon # Add
                    epsilon

    def predict(self, X):
        predictions = [self._predict_sample(x) for x in X]
        return np.array(predictions)

    def _predict_sample(self, x):
        log_posteriors = []

        for i, c in enumerate(self.classes):
            log_prior = np.log(self.class_priors[i])
            log_likelihood = 0.0

            for j in range(self.n_features):
                mean = self.class_means[i, j]
                std = self.class_stds[i, j]

                # Gaussian PDF formula: P(x|c) = (1 / (sqrt(2 * pi * sigma^2)))
                     * exp(-((x - mu)^2 / (2 * sigma^2)))
                # Log of Gaussian PDF: log(P(x|c)) = -0.5 * np.log(2 * np.pi *
                    std*2) - ((x[j] - mean)2 / (2 * std*2))
                log_likelihood += -0.5 * np.log(2 * np.pi * std**2) - ((x[j] -
                    mean)**2 / (2 * std**2))

            log_posteriors.append(log_prior + log_likelihood)

        return self.classes[np.argmax(log_posteriors)]

# --- Train and Evaluate Gaussian NB ---
gnb = GaussianNaiveBayesFromScratch()
gnb.fit(X_train_gnb, y_train_gnb)
y_pred_gnb = gnb.predict(X_test_gnb)

# Calculate accuracy for Gaussian NB
accuracy_gnb = np.sum(y_pred_gnb == y_test_gnb) / len(y_test_gnb)
print(f"Gaussian Naive Bayes Accuracy: {accuracy_gnb * 100:.2f}%")
```

Listing 3: Bernoulli NaiveBayes Implementation

```
import numpy as np

class GaussianNaiveBayesFromScratch:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.n_classes = len(self.classes)
        self.n_features = X.shape[1]

        self.class_priors = np.zeros(self.n_classes, dtype=np.float64)
        self.class_means = np.zeros((self.n_classes, self.n_features), dtype=np
            .float64)
        self.class_stds = np.zeros((self.n_classes, self.n_features), dtype=np.
            float64)

        epsilon = 1e-9 # Small value to prevent division by zero in std

        for i, c in enumerate(self.classes):
            X_c = X[y == c]
            self.class_priors[i] = X_c.shape[0] / X.shape[0]

            for j in range(self.n_features):
                self.class_means[i, j] = np.mean(X_c[:, j])
                self.class_stds[i, j] = np.std(X_c[:, j]) + epsilon # Add
                    epsilon

    def predict(self, X):
        predictions = [self._predict_sample(x) for x in X]
        return np.array(predictions)
```

```
    def _predict_sample(self, x):
        log_posteriors = []

        for i, c in enumerate(self.classes):
            log_prior = np.log(self.class_priors[i])
            log_likelihood = 0.0

            for j in range(self.n_features):
                mean = self.class_means[i, j]
                std = self.class_stds[i, j]

                # Gaussian PDF formula: P(x|c) = (1 / (sqrt(2 * pi * sigma^2)))
                    * exp(-((x - mu)^2 / (2 * sigma^2)))
                # Log of Gaussian PDF: log(P(x|c)) = -0.5 * np.log(2 * np.pi *
                    std*2) - ((x[j] - mean)2 / (2 * std*2))
                log_likelihood += -0.5 * np.log(2 * np.pi * std**2) - ((x[j] -
                    mean)**2 / (2 * std**2))

            log_posteriors.append(log_prior + log_likelihood)

        return self.classes[np.argmax(log_posteriors)]

# --- Train and Evaluate Gaussian NB ---
gnb = GaussianNaiveBayesFromScratch()
gnb.fit(X_train_gnb, y_train_gnb)
y_pred_gnb = gnb.predict(X_test_gnb)

# Calculate accuracy for Gaussian NB
accuracy_gnb = np.sum(y_pred_gnb == y_test_gnb) / len(y_test_gnb)
print(f"Gaussian Naive Bayes Accuracy: {accuracy_gnb * 100:.2f}%")
```

3.2 In-Progress Components

3.2.1 Multinomial Naïve Bayes

- Feature counting mechanism implemented
- Laplace smoothing ($\alpha = 1$) integrated
- Log-probability calculations working
- **Remaining:** Optimization for large feature spaces

3.2.2 Data Preprocessing Pipeline

Listing 4: Data Preprocessing Implementation

```
def preprocess_breast_cancer_data(filepath):
    # Load data
    df = pd.read_csv(filepath)

    # Remove ID column and encode diagnosis
    df = df.drop(columns=['id'])
    df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})

    # Separate features and target
    X = df.drop(columns=['diagnosis']).values
    y = df['diagnosis'].values

    # Normalize features
    scaler = StandardScaler()
    X_normalized = scaler.fit_transform(X)

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(
        X_normalized, y, test_size=0.2, random_state=42, stratify=y
    )

    return X_train, X_test, y_train, y_test, scaler
```

4 Testing and Validation Results

4.1 Unit Testing Results

Test Category	Test Cases	Passed	Pass Rate
Hash Table Operations	15	15	100%
Gaussian Probability Calculations	12	12	100%
Data Loading and Preprocessing	8	8	100%
Bernoulli Probability Calculations	10	7	70%
Total	**45**	**42**	**93.3%**

Table 3: Unit Test Results

4.2 Comparison with scikit-learn Baseline

Metric	Our Implementation	scikit-learn	Difference
Accuracy	96.46%	96.46%	
Precision	97.50%	100.0%	-2.5%
Recall	92.86%	90.48%	2.42%
F1-Score	95.12%	95.0%	0.12%
Training Time	0.0035s	0.003s	+0.0005s

Table 4: Comparison with scikit-learn GaussianNB

Metric	Our Implementation	scikit-learn	Difference
Accuracy	98.23%	98.23%	
Precision	97.62%	97.62%	
Recall	97.62%	97.62%	
F1-Score	97.62%	97.62%	
Training Time	0.0006s	0.0041s	-0.0034s

Table 5: Comparison with scikit-learn Bernoulli

5 Challenges Encountered and Solutions

5.1 Technical Challenges

5.1.1 Numerical Underflow

Problem: Probability multiplication for many features resulted in underflow.
Solution: Implemented log-probability calculations throughout the prediction process.

5.1.2 Class Imbalance Handling

Problem: The dataset has 62.7% benign vs 37.3% malignant samples.
Solution: Implemented stratified sampling during train-test split and adjusted class priors.

5.1.3 Hash Table Performance

Problem: Initial hash table implementation had poor collision handling.
Solution: Implemented separate chaining and dynamic resizing based on load factor.

5.2 Collaboration Challenges

5.2.1 Code Integration

Problem: Merging individual implementations led to conflicts.
Solution: Established Git workflow with feature branches and regular integration meetings.

5.2.2 Documentation Consistency

Problem: Inconsistent documentation styles between team members.
Solution: Created documentation templates and coding standards document.

6 Team Collaboration and Contributions

6.1 Individual Contributions

Team Member	Contributions	Hours Contributed
Khamis Haji	• Gaussian Naïve Bayes implementation • Data preprocessing pipeline • Performance testing framework • Mathematical derivation documentation	45 hours
Juweayria Farouk	• Hash table data structure • Multinomial Naïve Bayes implementation • Bernoulli Naïve Bayes implementation • Unit test framework	42 hours
Shared	• Project planning and timeline management • Code review and integration • Progress report preparation • Weekly meetings and coordination	20 hours

Table 6: Team Contributions Summary

7 Next Steps and Revised Timeline

7.1 Revised Implementation Plan

Week	Original Plan	Revised Plan
Week 11	Performance optimization, comprehensive testing	• Complete Multinomial implementations • Performance optimization • Comprehensive integration testing
Week 12	Benchmarking and comparison, begin documentation	• Final benchmarking against scikit-learn • Cross-validation experiments • Begin comprehensive documentation
Week 13	Complete documentation, create presentation	• Complete all documentation • Prepare presentation materials • Final code cleanup and optimization
Week 14	Final testing, submit final project, presentation	• Final testing and validation • Project submission • Final presentation

Table 7: Revised Project Timeline

7.2 Specific Tasks for Next Phase

1. **Week 11 Priorities:**
   - Complete remaining 50% of Multinomial Naïve Bayes
   - Implement comprehensive integration tests
   - Optimize hash table performance for large datasets
2. **Week 12 Priorities:**
   - Detailed benchmarking against all scikit-learn variants
   - Statistical significance testing
   - Finalizing the documentation
3. **Risk Mitigation:**
   - Allocate buffer time for unexpected challenges
   - Daily code integration to prevent merge conflicts
   - Regular performance monitoring and profiling

8 Conclusion

The project is progressing well and remains on track to meet all proposed objectives. Key achievements include the successful implementation of core data structures, complete Gaussian Naïve Bayes classifier and Bernoulli , and significant progress on Multinomial. Initial performance results are promising, with our implementation achieving 96.46% in Gaussian and 98.23% in Bernoulli accuracy on the Breast Cancer Wisconsin dataset, with 0% and -0.003% respectively below the scikit-learn baseline.

The team has demonstrated effective collaboration through regular meetings, systematic code integration, and shared responsibility for testing and documentation. While some minor challenges were encountered, appropriate solutions were implemented, and the project timeline has been adjusted accordingly. We are confident that the remaining work can be completed within the revised timeline, and we look forward to delivering a robust, well-documented implementation of the Naïve Bayes classifier that meets all functional and performance requirements.