A Comparative Analysis and Implementation on Gaussian Naive Bayes (GNB)

Janet Joseph, Jimmy Lin, Paneri Patel, Zehui Zhou

Team - Learning not Machine

Contents

What is Naive Bayes?

Naive Bayes is a **probabilistic classifier** based on Bayes' theorem:

$$P(Y \mid X) = \frac{P(X \mid Y)\, P(Y)}{P(X)}$$

Conditional independence assumption: given the class (Y = c), all features (x_1,...,x_d) are independent.

$$P(X \mid Y = c) = \prod_{j=1}^{d} P(x_j \mid Y = c)$$

Why "Gaussian"?

For continuous features, Gaussian Naive Bayes assumes each feature follows a normal (Gaussian) distribution under each class:

$$x_j \mid Y = c \sim \mathcal{N}(\mu_{jc}, \sigma_{jc}^2)$$

GNB Training Pipeline

1. **Estimate priors**:

The prior probability of each class (c) is: $$P(Y = c) = \frac{\text{count of class } c}{N}$$

2. **Estimate mean and variance** for every feature (j) and every class (c).

Mean ($\mu_{c,j}$) = $\frac{1}{N_c} \sum_{i=1}^{N_c} x_{i,j}$ Variance ($\sigma_{c,j}^2$) = $\frac{1}{N_c - 1} \sum_{i=1}^{N_c} (x_{i,j} - \mu_{c,j})^2$

3. **Prediction**:

Compute the **posterior probability** for each class: $$P(Y = c \mid X) \propto P(Y = c) \prod_{j=1}^{d} P(x_j \mid Y = c)$$

Then **choose the class with the maximum posterior probability**.

Representation

Gaussian Naive Bayes converts input features into a class prediction by computing the **posterior probability** for each class:

$$P(Y = c \mid X = x_1, \ldots, x_d) \propto P(Y = c) \prod_{j=1}^{d} \mathcal{N}(x_j \mid \mu_{jc}, \sigma_{jc}^2)$$

The classifier predicts the class with the highest posterior. In practice, we work with **log probabilities** (to avoid numerical underflow):

$$\hat{y} = \arg\max_c \left[\log P(Y = c) + \sum_{j=1}^{d} \log \mathcal{N}(x_j ; \mu_{jc}, \sigma_{jc}^2) \right]$$

We use **log probabilities for numerical stability** because multiplying many small Gaussian likelihoods can lead to extremely tiny numbers that computers cannot represent reliably. Summing logs avoids this problem.

Loss Function

Naive Bayes does not minimize a traditional loss like MSE or cross-entropy using gradient descent.

Instead, the model is trained by **maximizing the likelihood** of the data.

Equivalently, the loss is the **negative log-likelihood**:

$$L = -\sum_{i=1}^{N} \log P\left(y^{(i)} \mid x^{(i)}\right)$$

Gaussian Naive Bayes maximizes the likelihood under the assumption that each feature is normally distributed for each class.

Maximum Likelihood Estimation (MLE). For class c and feature j, the likelihood is modeled as:

$$P(x_j \mid Y = c) = \frac{1}{\sqrt{2\pi\sigma_{c,j}^2}} \exp\left(-\frac{(x_j - \mu_{c,j})^2}{2\sigma_{c,j}^2}\right)$$

Because the parameters have **closed-form MLE solutions**, no gradient descent, no iterative optimization is required during training.

Optimizer

Gaussian Naive Bayes does **not** use an iterative optimizer such as gradient descent.

Gaussian Naive Bayes has **closed-form Maximum Likelihood Estimates (MLE)** for all parameters.

Prior probabilities:

Gaussian parameters for each feature (j) and class (c):

Mean ($\mu_{c,j}$) = $\frac{1}{N_c}\sum_{i=1}^{N_c} x_{i,j}$ Variance ($\sigma_{c,j}^2$) = $\frac{1}{N_c - 1}\sum_{i=1}^{N_c}(x_{i,j} - \mu_{c,j})^2$

These are computed **directly from the data** using simple MLE formulas.

No optimization loop is required.

Code Overview

- Train the model
- Predict outcomes
- Evaluate accuracy
- Model test results

Train the Model

This method computes, for each class:

- Prior probabilities based on class frequencies
- The mean of each feature
- The variance of each feature

X_train : numpy.ndarray
A 2D array of shape (n_examples, n_attributes) containing continuous features.

y_train : numpy.ndarray
A 1D array of shape (n_examples,) containing integer class labels in {0, …, n_classes-1}.

Pseudo-code

1. Convert X_train and y_train to arrays
2. If X_train is not 2D, raise error
3. If y_train is not 1D, raise error
4. If number of rows in X_train ≠ length of y_train, raise error
5. n_examples ← number of rows in X_train
   n_attributes ← number of columns in X_train
6. Initialize:
   means ← empty list
   vars ← empty list
   priors ← empty list
7. epsilon ← var_smoothing × max(variance of X_train across features)
8. For each class c = 0 to n_classes − 1:
   a. X_c ← all rows of X_train where y_train = c
   b. If X_c is empty, raise error
   c. prior_c ← |X_c| / n_examples
      Append prior_c to priors
   d. mu_c ← mean of X_c across features
   e. var_c ← variance of X_c across features + epsilon
   f. Append mu_c to means
   g. Append var_c to vars
9. means ← array(means)
   vars ← array(vars)
   priors ← array(priors)
10. Return the trained model

Gaussian Probability Function

_gaussian_pdf: Computes Gaussian pdf value for a scalar x.

@params:
x: float
mu: float (mean)
var: float (variance)
@return:
pdf value at x

Pseudo-code

1. coeff ← 1 / sqrt($2\pi\sigma^2$)
2. exp_term ← exp($-(x-\mu)^2 / (2\sigma^2)$)
3. pdf ← coeff × exp_term
4. Return pdf

Predict Outcomes

- predict: Outputs a predicted label for each input in inputs. Uses log-space to avoid overflow/underflow.

@params:
inputs: a 2D NumPy array containing inputs
@return:
a 1D numpy array of predictions

Pseudo-code

1. Convert inputs to array
2. If inputs is 1D, reshape to 2D
3. If inputs is not 2D, raise error
4. If number of features in inputs ≠ number of trained features, raise error
5. Initialize predictions ← empty list
6. For each input vector x in inputs:
    a. Initialize log_joint_probs ← empty list
    b. For each class c = 0 to n_classes − 1:
        i. log_prob ← log(prior[c])
        ii. For each feature j:
            pdf ← GaussianPDF(x[j], mean[c][j], var[c][j])
            log_prob ← log_prob + log(pdf + ε)
        iii. Append log_prob to log_joint_probs
    c. predicted_label ← index of maximum value in log_joint_probs
    d. Append predicted_label to predictions
7. Return predictions as an array

Predict Outcomes

- predict_proba: Outputs posterior probabilities for each class.

@params:
inputs: a 2D NumPy array containing inputs
@return:
a 2D numpy array of probabilities (n_examples x n_classes)

Pseudo-code

1. Convert inputs to array
2. If inputs is 1D, reshape to 2D
3. Initialize probas ← empty list
4. For each input vector x in inputs:
    a. Initialize log_joint_probs ← empty list
    b. For each class c = 0 to n_classes − 1:
        i. log_prob ← log(prior[c])
        ii. For each feature j:
            pdf ← GaussianPDF(x[j], mean[c][j], var[c][j])
            log_prob ← log_prob + log(pdf + ε)
        iii. Append log_prob to log_joint_probs
    c. Convert log values to probabilities:
        i. max_log ← max(log_joint_probs)
        ii. exp_shifted ← exp(log_joint_probs − max_log)
        iii. probs ← exp_shifted / sum(exp_shifted)
    d. Append probs to probas
5. Return probas as an array

Loss Function

loss: Computes average negative log-likelihood (NLL) loss.

@params:
X: a 2D numpy array of examples
y: a 1D numpy array of labels
@return:
float, average negative log-likelihood

Pseudo-code

1. Convert X and y to arrays
2. proba ← PredictProba(X)
3. ε ← small constant
4. Initialize log_probs ← empty list
5. For each index i = 0 to length(y) − 1:
    a. true_class ← y[i]
    b. log_p ← log(proba[i, true_class] + ε)
    c. Append log_p to log_probs
6. loss ← − mean(log_probs)
7. Return loss

Evaluate Accuracy

accuracy: Outputs the accuracy of the trained model on a given dataset.

score: Returns the mean accuracy on the given test data and labels.

Pseudo-code (accuracy)

@params:
    X_test: a 2D numpy array of examples
    y_test: a 1D numpy array of labels
@return:
    float, accuracy between 0 and 1

1. predictions ← Predict(X_test)
2. correct ← number of indices where predictions = y_test
3. accuracy ← correct / length(y_test)
4. Return accuracy

Pseudo-code (score)

@params:
    X: a 2D numpy array of examples
    y: a 1D numpy array of true labels
@return:
    float, accuracy score

1. score ← Accuracy(X, y)
2. Return score

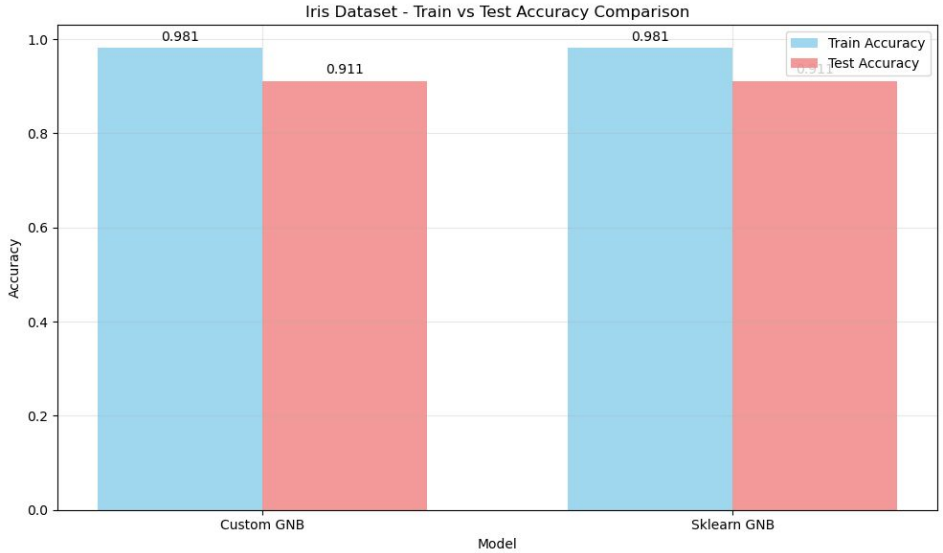
Custom GNB vs Sklearn: Statistical Results and Plots

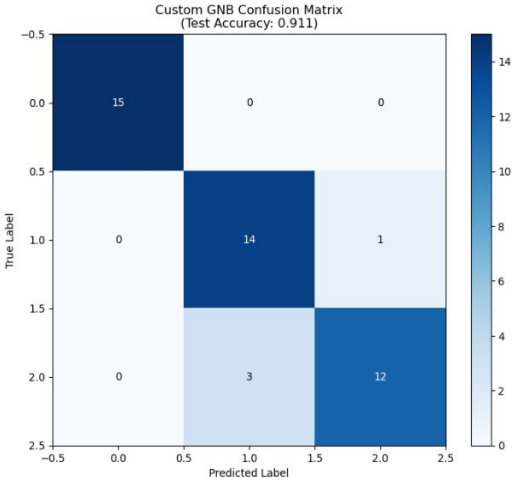
- Prediction Agreement
  - Percentage of identical predictions between models
- Accuracy Results
  - Percentage of correct predictions
- Confidence Statistics
  - Mean confidence: average of max(predict_proba)
- Probability Differences
  - Differences in probability outputs between models
- Timing Statistics
  - How long training takes
- Accuracy Comparison Bar Plot
  - Train vs test accuracy for both implementations
- Confusion Matrix Comparison
  - Where predictions go right/wrong in a grid format
- Training Time Comparison
  - Bar chart of training speed with error bars
- Decision Boundaries (2D only)
  - Colored regions where each class "wins" in feature space

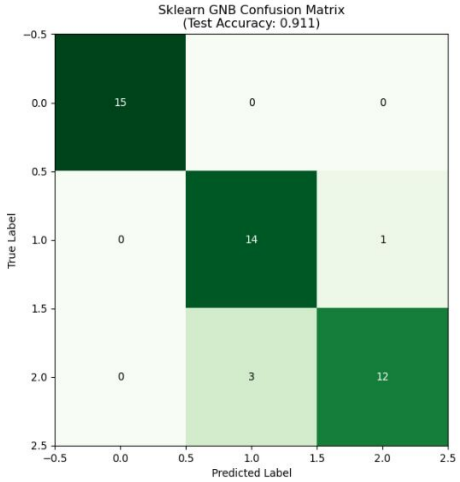
GNB Test Result - ***Iris*** dataset

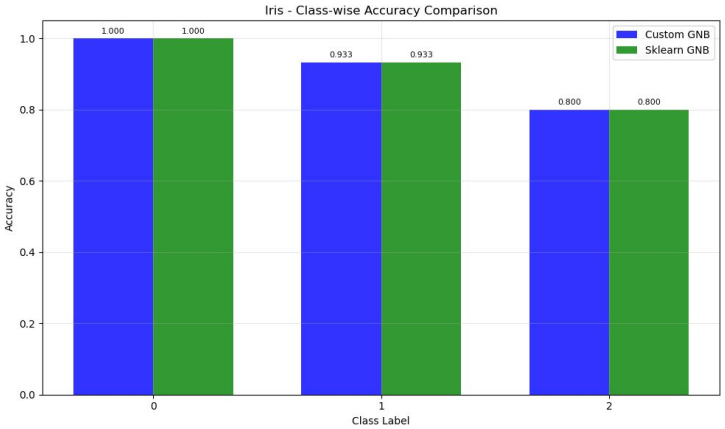
We compared our implementation with the scikit-learn Gaussian Naive Bayes model using ***Iris*** datasets provided by scikit-learn.

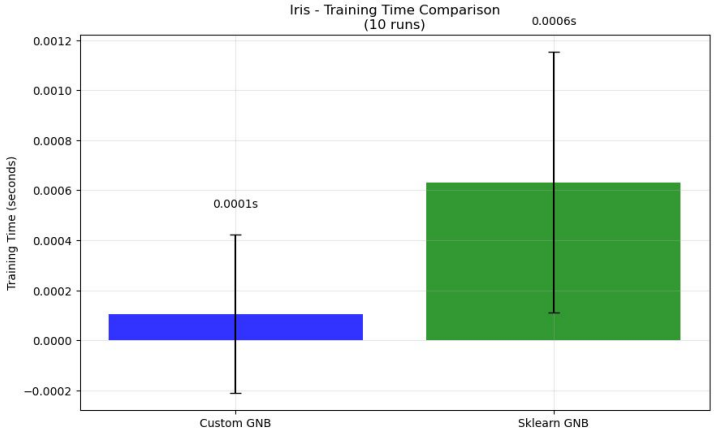
```
 Statistical summary - Iris:
Prediction Agreement: 1.0000
Custom GNB  - Train: 0.9810, Test: 0.9111
Sklearn GNB - Train: 0.9810, Test: 0.9111
Accuracy Difference: 0.000000

Confidence Statistics:
Custom GNB Mean Confidence: 0.9698
Sklearn GNB Mean Confidence: 0.9698
Custom GNB Correct Prediction Mean Confidence: 0.9800
Custom GNB Wrong Prediction Mean Confidence: 0.8656

Timing Statistics (10 runs):
Custom GNB Avg Training Time: 0.000105s ± 0.000316s
Sklearn GNB Avg Training Time: 0.000632s ± 0.000520s
Speed Ratio (Custom/Sklearn): 0.17x
```

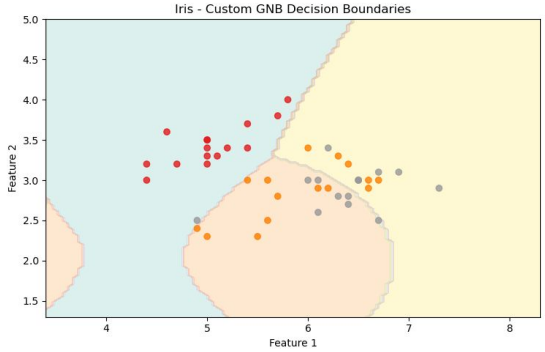


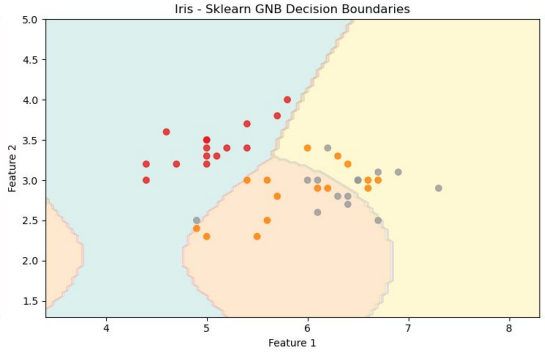

Custom GNB Confusion Matrix
(Test Accuracy: 0.911)
True Label
Predicted Label
15
0
0
0
14
1
0
3
12


Sklearn GNB Confusion Matrix
(Test Accuracy: 0.911)
True Label
Predicted Label
15
0
0
0
14
1
0
3
12


Iris - Class-wise Accuracy Comparison
Custom GNB
Sklearn GNB
Accuracy
Class Label
1.000
1.000
0.933
0.933
0.800
0.800


Iris - Training Time Comparison
(10 runs)
Training Time (seconds)
0.0001s
0.0006s
Custom GNB
Sklearn GNB

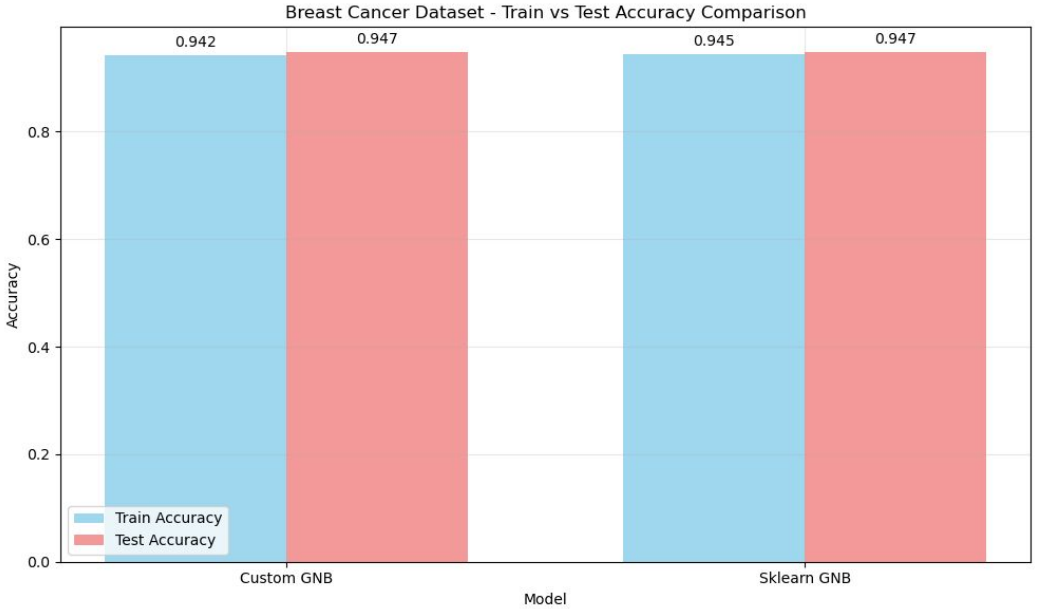

Iris - Custom GNB Decision Boundaries
Feature 1
Feature 2

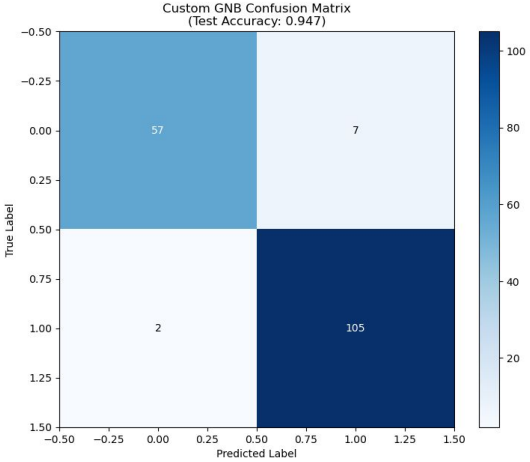

Iris - Sklearn GNB Decision Boundaries
Feature 1
Feature 2

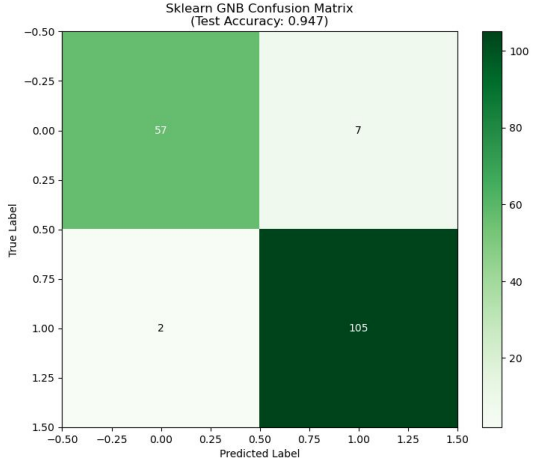
GNB Test Result - ***Breast Cancer*** dataset

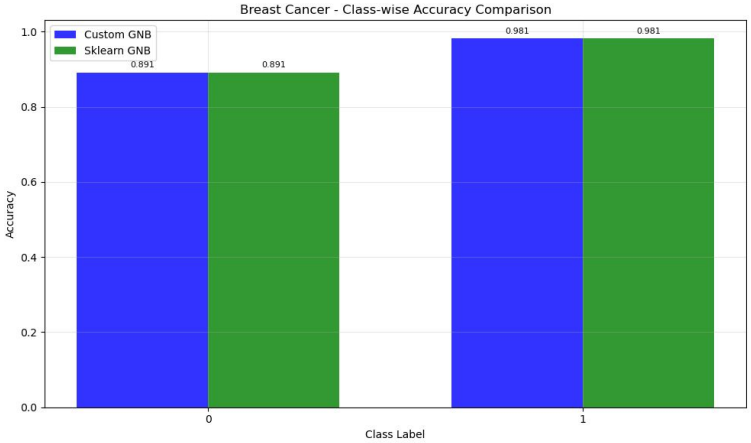
We compared our implementation with the scikit-learn Gaussian Naive Bayes model using ***Breast Cancer*** datasets provided by scikit-learn.

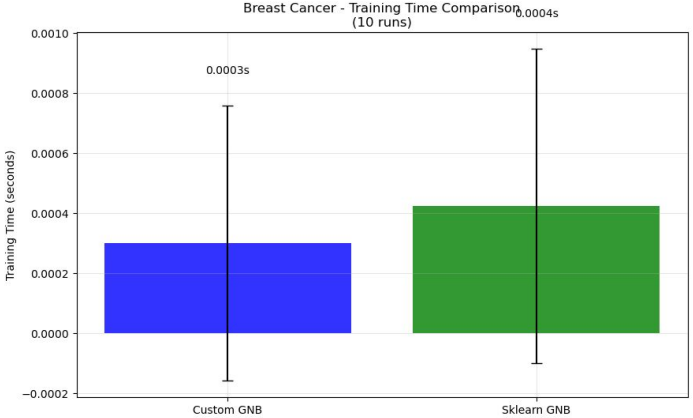
```
 Statistical summary - Breast Cancer:
Prediction Agreement: 1.0000
Custom GNB  - Train: 0.9422, Test: 0.9474
Sklearn GNB - Train: 0.9447, Test: 0.9474
Accuracy Difference: 0.000000

Confidence Statistics:
Custom GNB Mean Confidence: 0.9871
Sklearn GNB Mean Confidence: 0.9871
Custom GNB Correct Prediction Mean Confidence: 0.9906
Custom GNB Wrong Prediction Mean Confidence: 0.9240

Timing Statistics (10 runs):
Custom GNB Avg Training Time: 0.000300s ± 0.000458s
Sklearn GNB Avg Training Time: 0.000424s ± 0.000523s
Speed Ratio (Custom/Sklearn): 0.71x
```

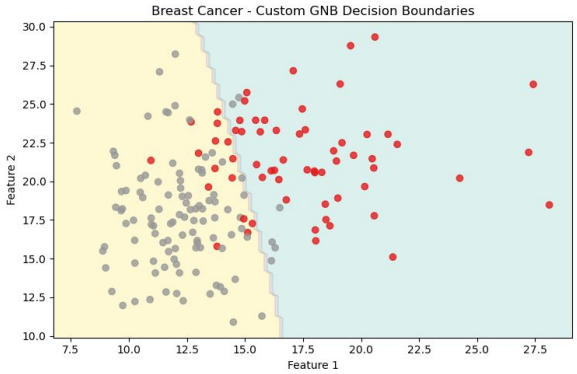


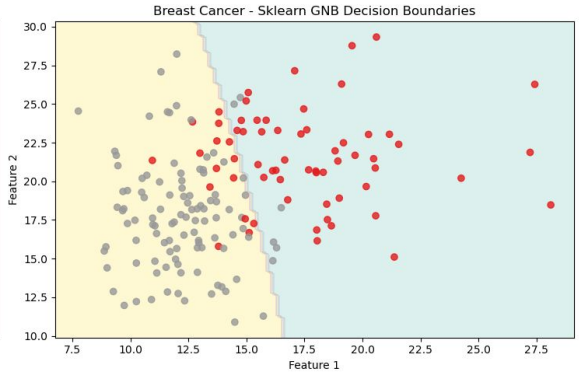

Custom GNB Confusion Matrix
(Test Accuracy: 0.947)
True Label
Predicted Label
57
7
2
105


Sklearn GNB Confusion Matrix
(Test Accuracy: 0.947)
True Label
Predicted Label
57
7
2
105


Breast Cancer - Class-wise Accuracy Comparison
Custom GNB
Sklearn GNB
Accuracy
Class Label
0.891
0.891
0.981
0.981


Breast Cancer - Training Time Comparison
(10 runs)
Training Time (seconds)
0.0003s
0.0004s
Custom GNB
Sklearn GNB

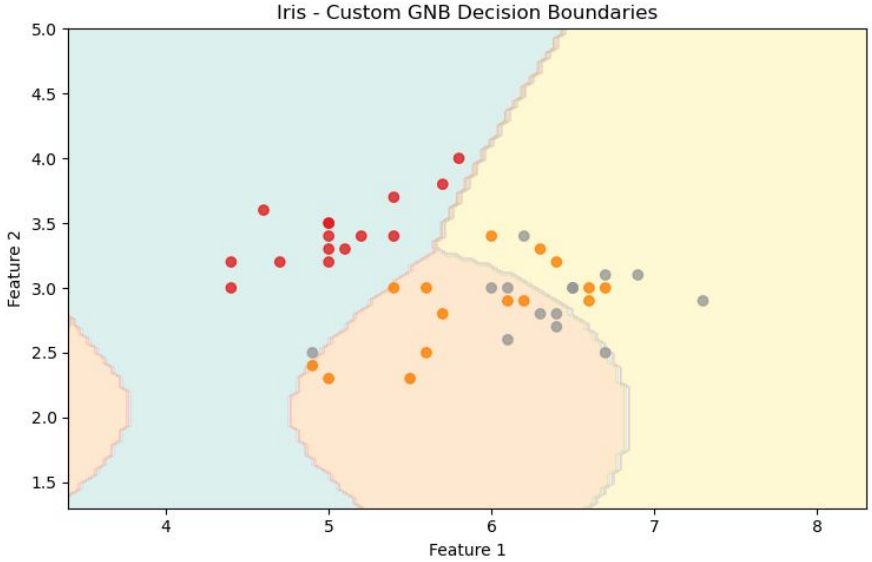

Breast Cancer - Custom GNB Decision Boundaries
Feature 1
Feature 2

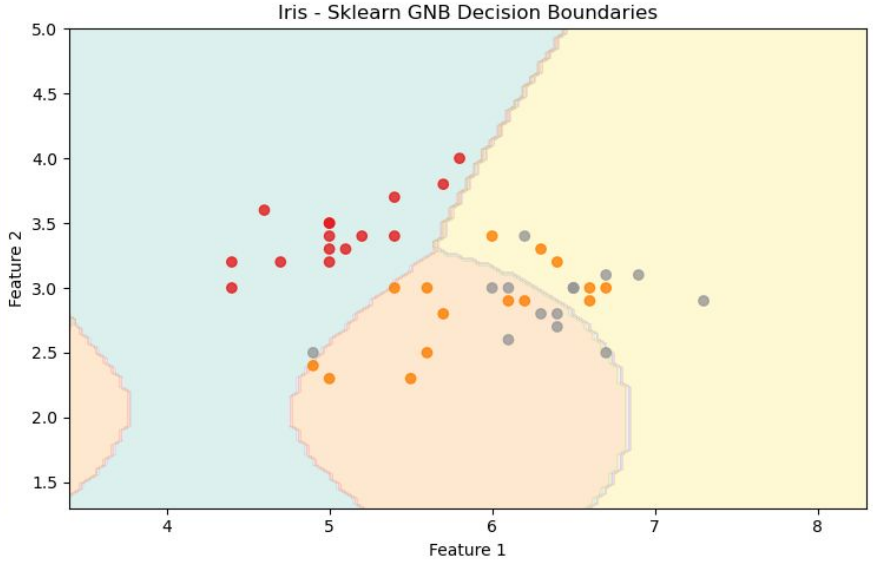

Breast Cancer - Sklearn GNB Decision Boundaries
Feature 1
Feature 2

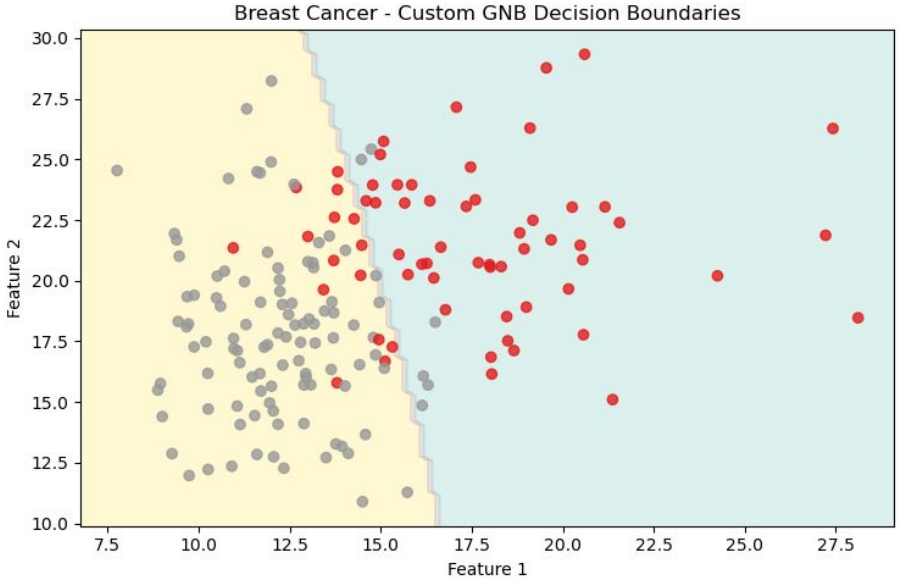
Conclusion

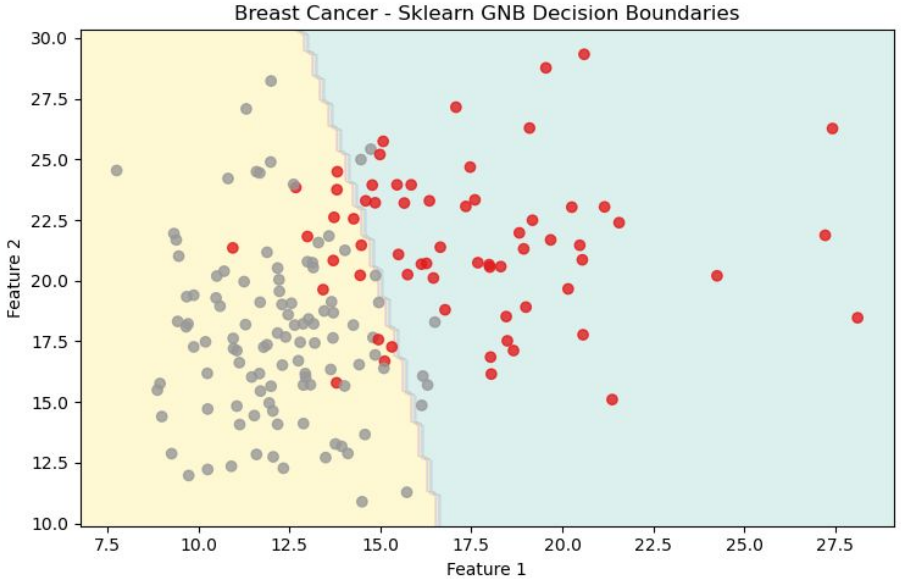
- We implemented the Gaussian Naive Bayes model by computing the class-specific means and variances from the training data.
- Using these parameters, we constructed the Gaussian likelihood for each class and used it to compute posterior probabilities for prediction.
- Our implementation achieves the similar accuracy as the scikit-learn Gaussian Naive Bayes classifier across the datasets we tested.
- *Challenges*: Maintaining numerical stability throughout the probability computations and ensuring correct tensor shapes and feature alignment across training and prediction

Q & A


Iris - Custom GNB Decision Boundaries
Feature 2
Feature 1
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
4
5
6
7
8


Iris - Sklearn GNB Decision Boundaries
Feature 2
Feature 1
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
4
5
6
7
8


Breast Cancer - Custom GNB Decision Boundaries
Feature 2
Feature 1
30.0
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5


Breast Cancer - Sklearn GNB Decision Boundaries
Feature 2
Feature 1
30.0
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5