Gaussian Naive Bayes for Classification

DATA 2060 Final Project

Professor Andras Zsom

Axle Zhao, Jiaqi Zhang, Ghirish Thaenraj

Overview - Gaussian Naive Bayes

- Probabilistic *generative* classifier based on Bayes' Theorem
- Assumes conditional independence of features
- Continuous features modeled with Gaussian distributions
- Very fast, interpretable, and surprisingly strong baseline
- Common in: medical diagnosis, spam filtering, real-time systems

Bayes' Theorem for Classification

Bayes' Theorem:

$$P(y \mid x) = \frac{P(x \mid y)P(y)}{P(x)}$$

The set up of our learning problem:

$$P(y = k \mid x) = \frac{P(y = k)P(x \mid y = k)}{P(x)} = \frac{P(y = k)\prod_{i=1}^{d} P(x_i \mid y = k)}{P(x)}$$

where

$$P(x) = \sum_k P(x \cap (y = k)) = \sum_k P(y = k) \prod_{i=1}^{d} P(x_i \mid y = k)$$

$$P(x \mid y = k) = \prod_{i=1}^{d} P(x_i \mid y = k)$$

because we assume features are independent given the class label

Since $P(x)$ is constant:

$$\hat{y} = \arg\max_k \left[\log P(y = k) + \sum_{j=1}^{d} \log P(x_j \mid y = k) \right]$$

Note: We work in log-space for numerical stability. (To prevent underflow)

Gaussian Likelihood Model

$P(y = k)$ is the fraction of the input data that has the label of k

For each feature x_j under class k:

$$P(x_j|y = k) = \frac{1}{\sqrt{2\pi\sigma_{jk}^2}} \exp\left(-\frac{(x_j - \mu_{jk})^2}{2\sigma_{jk}^2}\right)$$

$$\mu_{jk} = \text{mean of } x_j \text{ over samples with } y = k$$

$$\sigma_{jk}^2 = \text{variance of } x_j \text{ over samples with } y = k.$$

Numerical Stability & Smoothing

Why smoothing?

- Zero variance → division by zero in Gaussian PDF
- Solve by adding smoothing value ϵ

Sklearn-style smoothing:

$$\epsilon = 10^{-9} \cdot \max(\mathrm{Var}(X))$$

Key Numerical Techniques

- Maximum Likelihood Estimation (MLE) for parameters
- Log-space computation to avoid underflow
- Variance smoothing to prevent division-by-zero
- Gaussian log-PDF for stable likelihood calculations
- Log-Sum-Exp normalization for probability outputs
- Fully vectorized operations for efficiency

Training Algorithm (Pseudo-Code)

```
Input:
    X (n × d matrix of features)
    y (n labels)

For each class k:
    X_k ← all samples in X with label k
    prior[k] ← |X_k| / n
    mean[k]  ← average of each feature in X_k
    var[k]   ← variance of each feature in X_k  +  epsilon   # smoothing

Output:
    priors, means, variances
```

Prediction Algorithm (Pseudo-Code)

```
Input:
    x (1 × d feature vector), priors, means, variances

For each class k:
    log_prior[k] ← log( prior[k] )

    log_likelihood[k] ← 0
    For each feature j:
        log_likelihood[k] ← log_likelihood[k]
                                + log Gaussian_PDF( x_j ; mean[k,j], var[k,j] )

    log_posterior[k] ← log_prior[k] + log_likelihood[k]

Return the class with the maximum log_posterior
```

Probability Normalization

To compute P(y=k | x):

Shift all log_posterior values by subtracting their maximum

Compute exp(log_posterior_shifted[k]) for each class

Normalize by dividing by the sum of exponentials

Unit Testing Overview

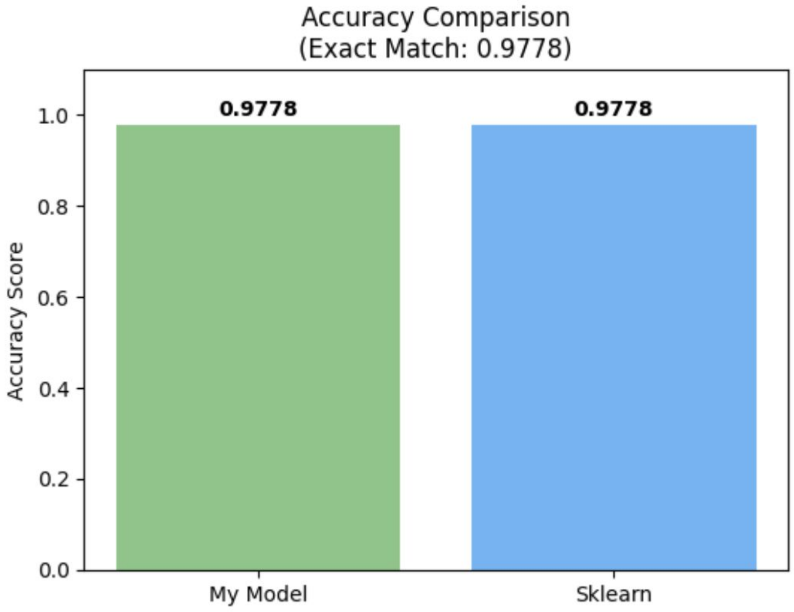
We implemented granular tests for:

- Input reshaping: 1D → 2D（Typo, we somehow included it by accident.)
- Zero-variance features
- Single-class datasets
- Loss function correctness
- Log-likelihood computation
- Probability normalization

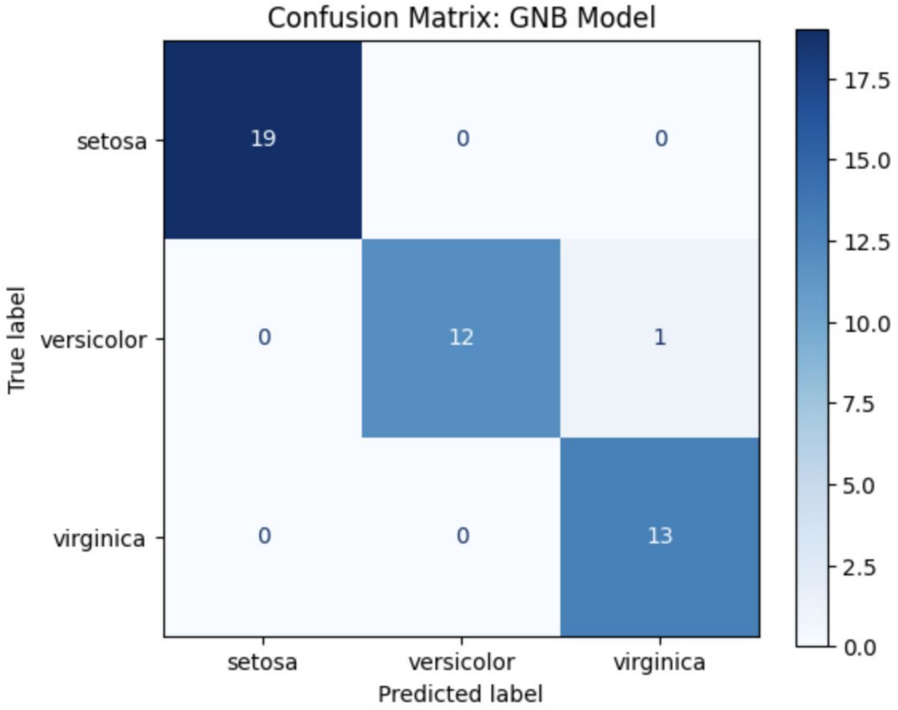
Outcome: all tests pass and validate correct numerical behavior.

Accuracy Comparison (Iris Dataset)

- Dataset: Iris (multiclass, 4 continuous features)
- Train/test split: 70/30, fixed random state
- Our model accuracy: 97.78%
- Sklearn accuracy: 97.78%
- All predicted class labels match exactly



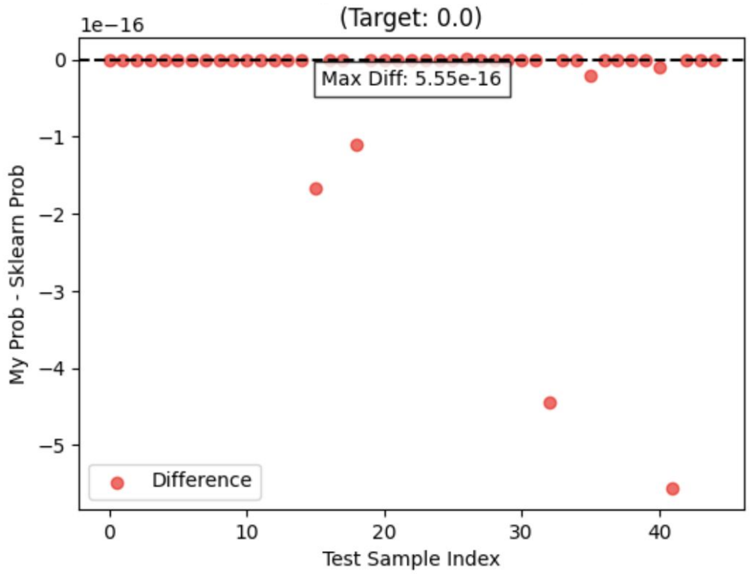
Confusion Matrix — GNB Model Performance



Probability Comparison

Probabilities match Sklearn to within:

$$\text{max difference} = 5.55 \times 10^{-16}$$

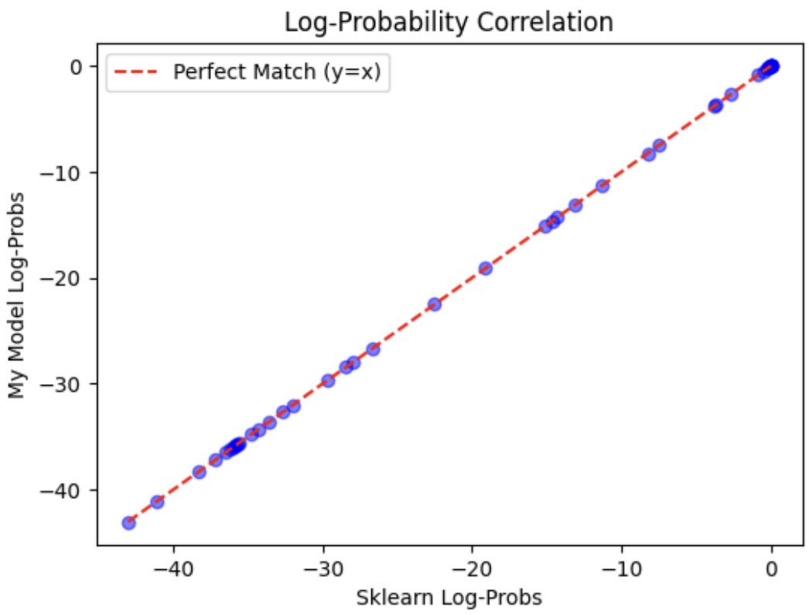


Log Probability Comparison

Log probabilities also match to machine precision

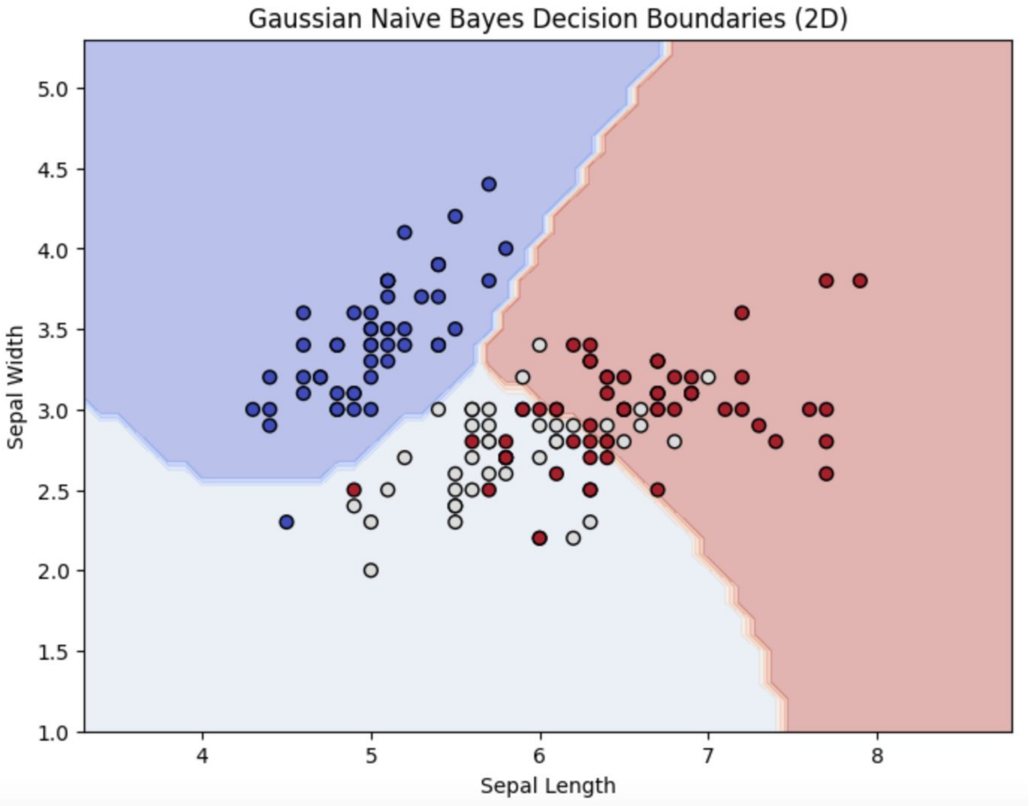
Confirms correct implementation of:

- Gaussian log-PDF
- Log-sum-exp normalization
- Class prior incorporation



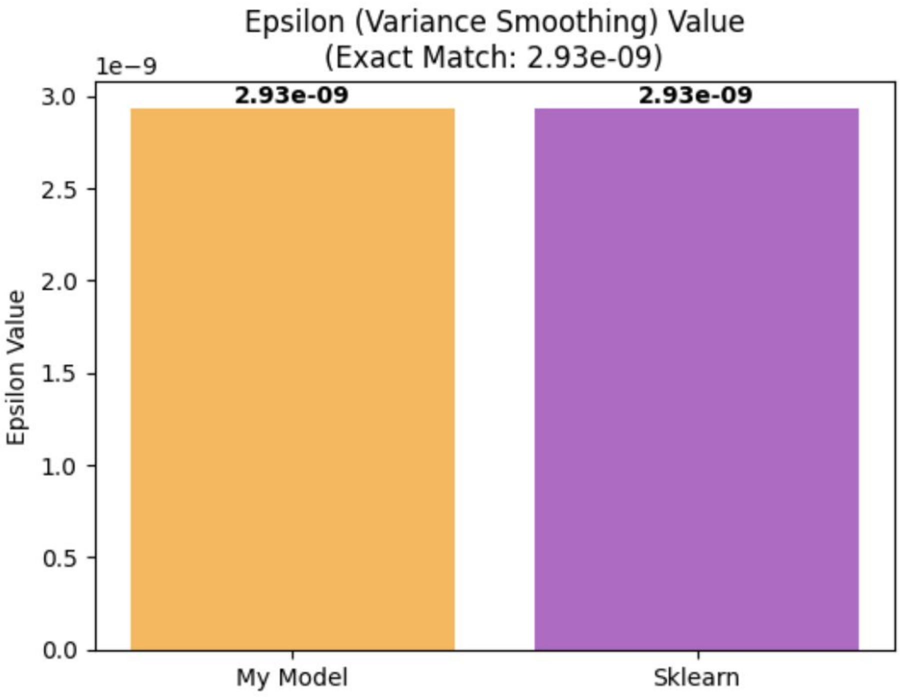
GNB Decision Boundaries (2D Projection)

```
Generating 2D Decision Boundary (Sepal Length vs Width)
```



Epsilon Smoothing Verification

- Our computed epsilon = Sklearn's epsilon
- This was crucial for exact probability reproduction
- Confirms correct replication of Sklearn internals



Key Takeaways

- GNB is simple but powerful
- Independence assumption → extremely efficient
- Generative models provide full probability distributions
- Numerical stability (log-space + smoothing) is essential
- Our implementation reproduces Sklearn exactly

Challenges We Encountered

- Understanding Sklearn’s dynamic variance smoothing
- Handling zero-variance features
- Normalizing log probabilities correctly
- Ensuring vectorization without losing numerical stability
- Designing comprehensive unit tests

Summary

We accomplished:

- Full scratch implementation of Gaussian Naive Bayes
- Correct mathematical modeling
- Robust unit tests covering edge cases
- Exact match with Sklearn in:
    - Predictions
    - Probabilities
    - Log-probabilities
    - Smoothing calculation
- Clear visual verification through our graphs

Thank You