Gaussian Naive Bayes

DATA 2060 (Fall 2025) Final Project Presentation

Zachk Huang | yifei_huang1@brown.edu
Zixi (Valerie) Li | zixi_li1@brown.edu

Gaussian Naive Bayes - Memes

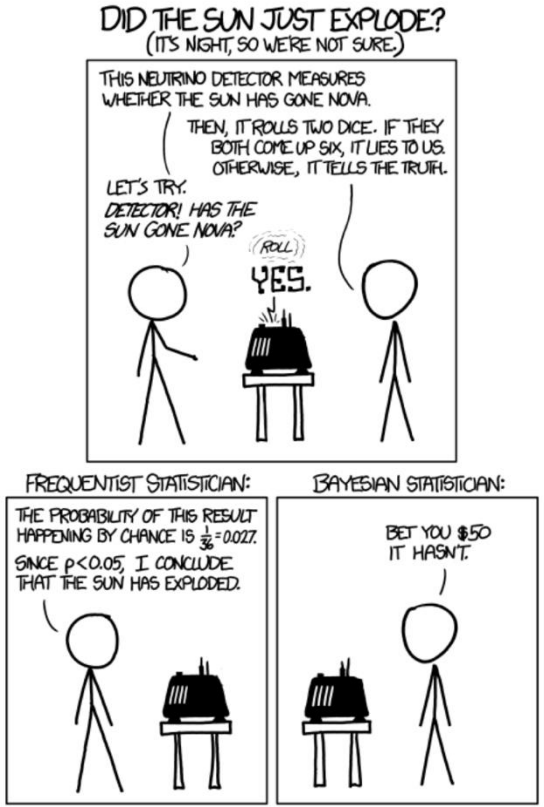


Photo Source: https://elec-club-iitb.github.io/blog/2016/03/glimpse-of-the-naive-bayes-classifier/

GaussianNB looking at its correlated features:

Photo Source: https://www.npr.org/2023/01/16/1149232763/this-is-fine-meme-anniversary-gunshow-web-comic

Gaussian Naive Bayes - Math

1. Bayes' Rule

Gaussian Naive Bayes classifies by choosing the class y that maximizes the posterior:

$$\hat{y} = \arg\max_{k} P(y = k \mid x),$$

using Bayes' rule:

$$P(y = k \mid x) \propto P(y = k)P(x \mid y = k).$$

2. Naive Conditional Independence Assumption

Features are assumed independent given the class:

$$P(x \mid y = k) = \prod_{j=1}^{d} P\left(x_j \mid y = k\right),$$

And this is exactly the "naive" part.

Gaussian Naive Bayes - Math Continued

3. Gaussian Assumption
For each feature x_j in class k:

$$x_j \mid y = k \sim \mathcal{N}(\mu_{k,j}, \sigma^2_{k,j})$$

Which means, GaussianNB assumes each feature follows a normal distribution within the class k.

4. So, the likelihood of seeing x_j for class k is:

$$P(x_j \mid y = k) = \frac{1}{\sqrt{2\pi\sigma^2_{k,j}}} \exp\left(-\frac{(x_j - \mu_{k,j})^2}{2\sigma^2_{k,j}}\right)$$

Therefore, the closer a feature value is to the class mean, the higher the likelihood. The farther away, the smaller the likelihood.

5. Thus, log-likelihood (GaussianNB sums log-likelihoods):

$$\boldsymbol{\log P(x_j \mid y = k) = -\frac{1}{2} \sum_j \left[\log(2\pi\sigma^2_{k,j}) + \frac{(x_j - \mu_{k,j})^2}{\sigma^2_{k,j}}\right]}$$

Gaussian Naive Bayes - Math Continued

6. Class Prior would be:

$$\log P(y = k) = \log(prior_k)$$

7. Posterior Used for Prediction

Our model tries to compute (unnormalized):

$$log\ P(y = k \mid x) = logP(y = k) + log\ P(x \mid y = k)$$

Convert back to real probabilities:

$$P\ (y = k \mid x) = \frac{\exp(\log P\ (y = k \mid x))}{\sum_{k\prime} \exp(\log P\ (y = k' \mid x))}$$

Final prediction:

$$\hat{y} = \arg\max_{k} P(y = k \mid x)$$

Normalization does not change the predicted label.

Gaussian Naive Bayes - Numerical Techniques

- As a reminder, numerical techniques matter in Gaussian Naive Bayes because:
  - Compute everything in **log-space** to avoid underflow.
  - To prevent division by zero, we add small ϵ to variances
  - Estimating means and variances using closed-form MLE
  - Summing log-likelihoods instead of multiplying Gaussians
  - After computing log-posteriors, **convert back to real probabilities and normalize**.
  - Final prediction is the **argmax** over the normalized probabilities (same as using log-posteriors).
  - Priors are estimated directly from **class frequencies** in the training data.

Gaussian Naive Bayes - Numerical Techniques Continued

- Pseudo-Code for Training (MLE Estimation):

```
Input: X_train (N × d), y_train (length N), K classes

Initialize:
    means[K][d], vars[K][d], priors[K]

For each class k:
    X_k = all rows of X_train where y_train == k
    priors[k] = (# rows in X_k) / N

    For each feature j:
        means[k][j] = average of feature j in X_k
        vars[k][j]  = variance of feature j in X_k + ε   # stability
```

So, no gradient descent, no iteration, although we are the gradient descenters ☺

Gaussian Naive Bayes - Numerical Techniques Continued

- Pseudo-Code for Prediction (Log-Space Computation):

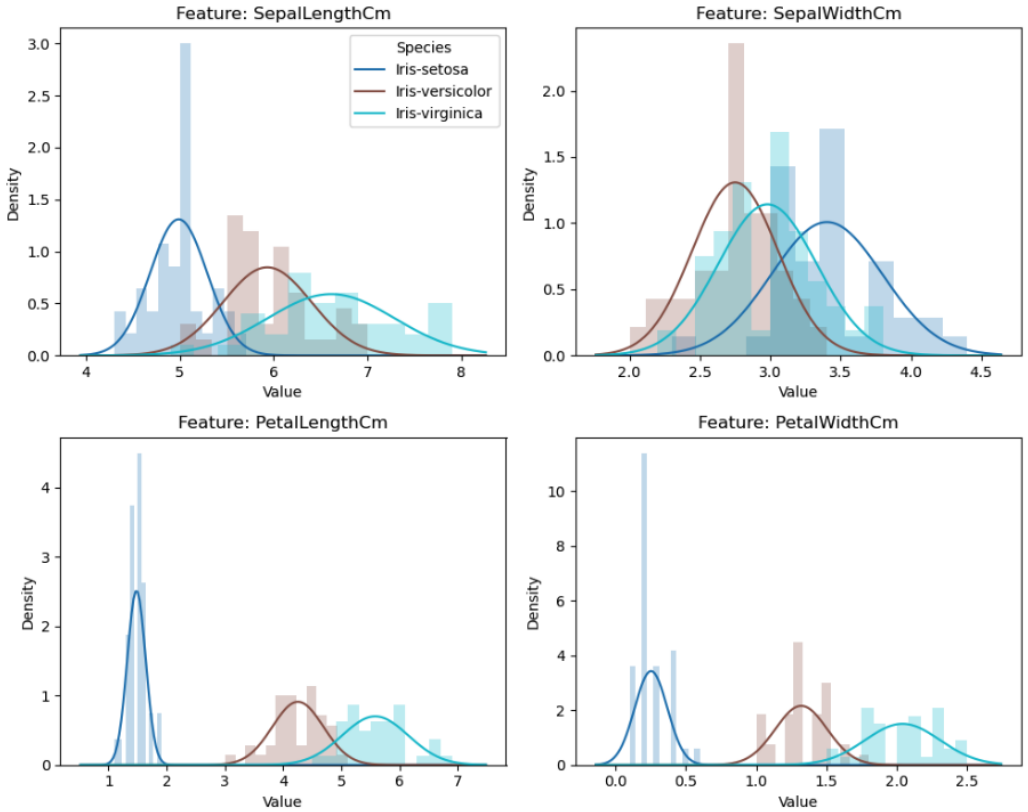
```
Input: x (single example), means, vars, priors

For each class k:
    log_prior = log(priors[k])
    log_likelihood = 0

    For each feature j:
        log_likelihood +=
            -0.5 * [ log(2π * vars[k][j])
                    + (x_j - means[k][j])^2 / vars[k][j] ]

    log_post[k] = log_prior + log_likelihood

# Convert log-posteriors back to real probabilities
joint[k] = exp(log_post[k])
Normalize:  prob[k] = joint[k] / sum(joint)

# Final prediction (normalization does not affect argmax)
y_hat = argmax_k prob[k]
```

Dataset and Setup

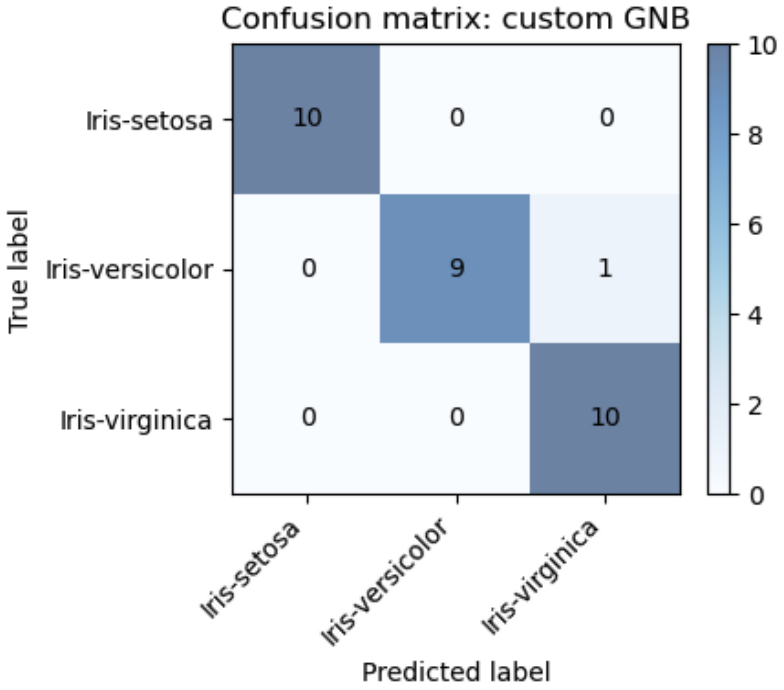
- Dataset: **Iris (N=150, 4 features, 3 classes, very balanced)**
- Train/test split: **80/20**, stratified, random_state=42
- Models compared:
  - **Our custom Gaussian Naive Bayes (from scratch)**
  - **sklearn GaussianNB**

Goal: verify that our implementation reproduces sklearn's behavior.

Class-Conditional Gaussian Fits Learned by Our Model



Confusion Matrix - Custom GaussianNB



Comparison with Sklearn results

- Our scratch implementation matches sklearn (GaussianNB)’s accuracy exactly on the held-out test set.
- Posterior probabilities are identical within the tolerance of 1e-9
- Predicted class for all test points are identical
- Both have an accuracy score of 0.967

Summary and Reflections

- **What we found interesting**
  - Gaussian has very high accuracy on this small dataset, similar to Neural Network according to the UCI ML Repository
  - Even though only petal features are highly separated, Naive Bayes assumes conditional independence, so their sharply peaked likelihoods get multiplied, leading to extremely high or low predicted probabilities (0s and 1s).
- **What was challenging**
  - Ensuring **numerical stability** (variance smoothing, log-space).
  - small variance causes issue!
    - class 0: N(0,0.05) → P(x=0.9|y=0)= 5.39×10^{-4}
    - class 1: N(1,1e-9)→ P(x=0.9|y=1)= $1.26 \times 10^{-2171468}$

Thank You! ☺
Questions?

DATA 2060 (Fall 2025) Final Project Presentation

Zachk Huang | yifei_huang1@brown.edu
Zixi (Valerie) Li | zixi_li1@brown.edu