Na”ıve Bayes: Theory and Practice

September 9, 2025

1 Introduction

Na”ıve Bayes (NB) is a family of simple yet effective probabilistic classifiers. Under the *conditional independence* assumption, the posterior is

$$p(y \mid \mathbf{x}) \propto p(y) \prod_{j=1}^{d} p(x_j \mid y), \tag{1}$$

where y is the class and $\mathbf{x} = (x_1, \ldots, x_d)$ are features. Despite the strong assumption, NB can work surprisingly well in many domains, especially with high-dimensional sparse inputs.

2 Theory and Formulas

Consider the Gaussian NB model for continuous features. For class $c \in \{1, \ldots, C\}$, assume

$$x_j \mid y = c \ \sim \ \mathcal{N}(\mu_{c,j},\, \sigma_{c,j}^2), \quad j = 1, \ldots, d. \tag{2}$$

Then the class-conditional density factorizes: $p(\mathbf{x} \mid y = c) = \prod_j \mathcal{N}(x_j\,;\, \mu_{c,j}, \sigma_{c,j}^2)$. With prior $p(y = c)$, the (unnormalized) log-posterior is

$$\begin{aligned} \log p(y = c \mid \mathbf{x}) &\propto \log p(y = c) + \sum_{j=1}^{d} \log \mathcal{N}(x_j\,;\, \mu_{c,j}, \sigma_{c,j}^2) && (3) \\ &\propto \log p(y = c) - \sum_{j=1}^{d} \left[\tfrac{1}{2} \log(2\pi\sigma_{c,j}^2) + \tfrac{(x_j - \mu_{c,j})^2}{2\sigma_{c,j}^2} \right]. && (4) \end{aligned}$$

The predicted class is $\hat{y} = \arg\max_c \log p(y = c \mid \mathbf{x})$. Estimation is straightforward via sample means and variances within each class.

Notes. NB variants include Gaussian NB for continuous features and Multinomial/Bernoulli NB for count/binary features with Laplace (additive) smoothing. Calibration may be needed if probabilities are used downstream.

3 Applications and Tips

- **When it works:** high-dimensional sparse text features (bag-of-words), simple sensor data, baseline models.

- **Preprocessing:** standardize continuous features for Gaussian NB; for text, TF-IDF or raw counts for Multinomial NB.
- **Class priors:** either empirical (class frequencies) or domain-informed.
- **Independence assumption:** correlations between features may harm performance; use as a baseline and compare.
- **Evaluation:** compare with logistic regression/SVMs; use cross-validation.

4 Python Practice

The script below generates figures for Gaussian NB decision boundaries and simple diagnostics. Run it in the chapter folder; it saves images under `figures/`.

Listing 1: Generate Naive Bayes figures

```
# Terminal
python gen_naive_bayes_figures.py
```

Listing 2: gen_naive_bayes_figures.py

```
"""
Figure generator for the Naive Bayes chapter.

Generates illustrative figures and saves them into the local 'figures/' folder.

Requirements:
- Python 3.8+
- numpy, matplotlib, scikit-learn

Install (if needed):
  pip install numpy matplotlib scikit-learn

This script avoids newer or experimental APIs to stay compatible with older
versions of the dependencies.
"""
from __future__ import annotations

import os
import math
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

try:
    from sklearn.datasets import make_blobs
    from sklearn.naive_bayes import GaussianNB
    from sklearn.linear_model import LogisticRegression
    from sklearn.preprocessing import StandardScaler
except Exception as e:
    raise SystemExit(
```

```python
        "Missing scikit-learn dependency. Please install with: pip install
            scikit-learn"
    )


def _ensure_figures_dir(path: str | None = None) -> str:
    """Create figures directory under this chapter regardless of CWD.

    If `path` is None, resolve to `<this_file_dir>/figures`.
    """
    if path is None:
        base = os.path.dirname(os.path.abspath(__file__))
        path = os.path.join(base, "figures")
    os.makedirs(path, exist_ok=True)
    return path


def _plot_decision_boundary(ax, clf, X, y, title: str, cmap_light, cmap_bold):
    # Create a mesh grid for decision surface
    x_min, x_max = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
    y_min, y_max = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0
    xx, yy = np.meshgrid(
        np.linspace(x_min, x_max, 300), np.linspace(y_min, y_max, 300)
    )
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    ax.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.8, levels=np.unique(Z).
        size)
    # Training points
    scatter = ax.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolors="k"
        , s=25)
    ax.set_xlabel("Feature 1")
    ax.set_ylabel("Feature 2")
    ax.set_title(title)
    return scatter


def fig_gnb_decision_boundary_2class(out_dir: str) -> str:
    np.random.seed(42)
    X, y = make_blobs(n_samples=400, centers=2, cluster_std=[1.2, 1.2],
        random_state=42)

    clf = GaussianNB()
    clf.fit(X, y)

    cmap_light = ListedColormap(["#FFEEEE", "#EEEEFF"])
    cmap_bold = ListedColormap(["#E74C3C", "#3498DB"])

    fig, ax = plt.subplots(figsize=(5.5, 4.5), dpi=150)
    _plot_decision_boundary(ax, clf, X, y, "Gaussian Naive Bayes (2-class)",
        cmap_light, cmap_bold)
    out_path = os.path.join(out_dir, "gnb_decision_boundary_2class.png")
    fig.tight_layout()
```

```python
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_gnb_decision_boundary_3class(out_dir: str) -> str:
    np.random.seed(7)
    X, y = make_blobs(
        n_samples=600,
        centers=3,
        cluster_std=[1.1, 1.0, 1.2],
        random_state=7,
    )

    clf = GaussianNB()
    clf.fit(X, y)

    cmap_light = ListedColormap(["#FFEEEE", "#EEFFEE", "#EEEEFF"])
    cmap_bold = ListedColormap(["#E74C3C", "#2ECC71", "#3498DB"])

    fig, ax = plt.subplots(figsize=(5.5, 4.5), dpi=150)
    _plot_decision_boundary(ax, clf, X, y, "Gaussian Naive Bayes (3-class)",
        cmap_light, cmap_bold)
    out_path = os.path.join(out_dir, "gnb_decision_boundary_3class.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def _gaussian_pdf(x: np.ndarray, mu: float, sigma: float) -> np.ndarray:
    coef = 1.0 / (math.sqrt(2.0 * math.pi) * sigma)
    return coef * np.exp(-0.5 * ((x - mu) / sigma) ** 2)


def fig_class_conditional_densities_1d(out_dir: str) -> str:
    # Two 1D Gaussians with equal priors
    mu0, sigma0 = -1.0, 1.0
    mu1, sigma1 = 1.2, 0.8
    xs = np.linspace(-5, 5, 500)
    p_x_c0 = _gaussian_pdf(xs, mu0, sigma0)
    p_x_c1 = _gaussian_pdf(xs, mu1, sigma1)

    # Decision threshold where p(x|c0) = p(x|c1)
    # For illustration, compute numerically
    idx = np.argmin(np.abs(p_x_c0 - p_x_c1))
    x_star = xs[idx]

    fig, ax = plt.subplots(figsize=(6, 4), dpi=150)
    ax.plot(xs, p_x_c0, label="p(x|class 0)", color="#E74C3C", lw=2)
    ax.plot(xs, p_x_c1, label="p(x|class 1)", color="#3498DB", lw=2)
    ax.axvline(x_star, color="#7F8C8D", ls="--", lw=1)
    ax.text(x_star + 0.1, max(p_x_c0[idx], p_x_c1[idx]) * 0.9, "decision",
        color="#7F8C8D")
```

```
    ax.set_xlabel("x")
    ax.set_ylabel("density")
    ax.set_title("Class-conditional densities (1D)")
    ax.legend(frameon=False)
    out_path = os.path.join(out_dir, "class_conditional_densities_1d.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_feature_independence_heatmap(out_dir: str) -> str:
    # Create 3 correlated features to illustrate independence assumption
        violation
    np.random.seed(123)
    mean = np.array([0.0, 0.0, 0.0])
    cov = np.array(
        [
            [1.0, 0.7, 0.4],
            [0.7, 1.0, 0.5],
            [0.4, 0.5, 1.0],
        ]
    )
    X = np.random.multivariate_normal(mean, cov, size=1000)
    # Empirical correlation matrix
    C = np.corrcoef(X, rowvar=False)

    fig, ax = plt.subplots(figsize=(4.8, 4.2), dpi=160)
    im = ax.imshow(C, cmap="coolwarm", vmin=-1, vmax=1)
    for i in range(C.shape[0]):
        for j in range(C.shape[1]):
            ax.text(j, i, f"{C[i, j]:.2f}", ha="center", va="center", color="
                black")
    ax.set_xticks([0, 1, 2])
    ax.set_yticks([0, 1, 2])
    ax.set_xticklabels(["f1", "f2", "f3"])
    ax.set_yticklabels(["f1", "f2", "f3"])
    ax.set_title("Feature correlation (independence assumption)")
    fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04, label="correlation")
    out_path = os.path.join(out_dir, "feature_independence_heatmap.png")
    fig.tight_layout()
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def fig_gnb_vs_logreg_boundary(out_dir: str) -> str:
    # Dataset with partially overlapping Gaussians
    np.random.seed(0)
    X, y = make_blobs(n_samples=500, centers=[(-2, -2), (2.5, 2.0)],
        cluster_std=[1.6, 1.2], random_state=0)

    scaler = StandardScaler()
    Xs = scaler.fit_transform(X)
```

```
    gnb = GaussianNB().fit(Xs, y)
    # Use lbfgs which supports multinomial/binary and is widely available
    lr = LogisticRegression(solver="lbfgs", max_iter=1000).fit(Xs, y)

    x_min, x_max = Xs[:, 0].min() - 2.0, Xs[:, 0].max() + 2.0
    y_min, y_max = Xs[:, 1].min() - 2.0, Xs[:, 1].max() + 2.0
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 300), np.linspace(y_min, y_max, 300))
    grid = np.c_[xx.ravel(), yy.ravel()]
    Z_gnb = gnb.predict(grid).reshape(xx.shape)
    Z_lr = lr.predict(grid).reshape(xx.shape)

    fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=True, sharey=True)
    for ax, Z, title in [
        (axes[0], Z_gnb, "Gaussian NB boundary"),
        (axes[1], Z_lr, "Logistic Regression boundary"),
    ]:
        ax.contourf(xx, yy, Z, alpha=0.25, levels=np.unique(y).size, cmap=ListedColormap(["#FFBBBB", "#BBBBFF"]))
        ax.scatter(Xs[:, 0], Xs[:, 1], c=y, s=15, cmap=ListedColormap(["#E74C3C", "#3498DB"]), edgecolors="k")
        ax.set_title(title)
        ax.set_xlabel("feature 1 (scaled)")
        ax.set_ylabel("feature 2 (scaled)")
    fig.suptitle("Naive Bayes vs Logistic Regression")
    out_path = os.path.join(out_dir, "gnb_vs_logreg_boundary.png")
    fig.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.savefig(out_path)
    plt.close(fig)
    return out_path


def main():
    # Always save figures inside the current chapter directory
    out_dir = _ensure_figures_dir(None)
    generators = [
        fig_gnb_decision_boundary_2class,
        fig_gnb_decision_boundary_3class,
        fig_class_conditional_densities_1d,
        fig_feature_independence_heatmap,
        fig_gnb_vs_logreg_boundary,
    ]

    print("Generating figures into:", os.path.abspath(out_dir))
    for gen in generators:
        try:
            path = gen(out_dir)
            print("Saved:", path)
        except Exception as e:
            print("Failed generating", gen.__name__, ":", e)
```

```python
if __name__ == "__main__":
    main()
```

5 Result

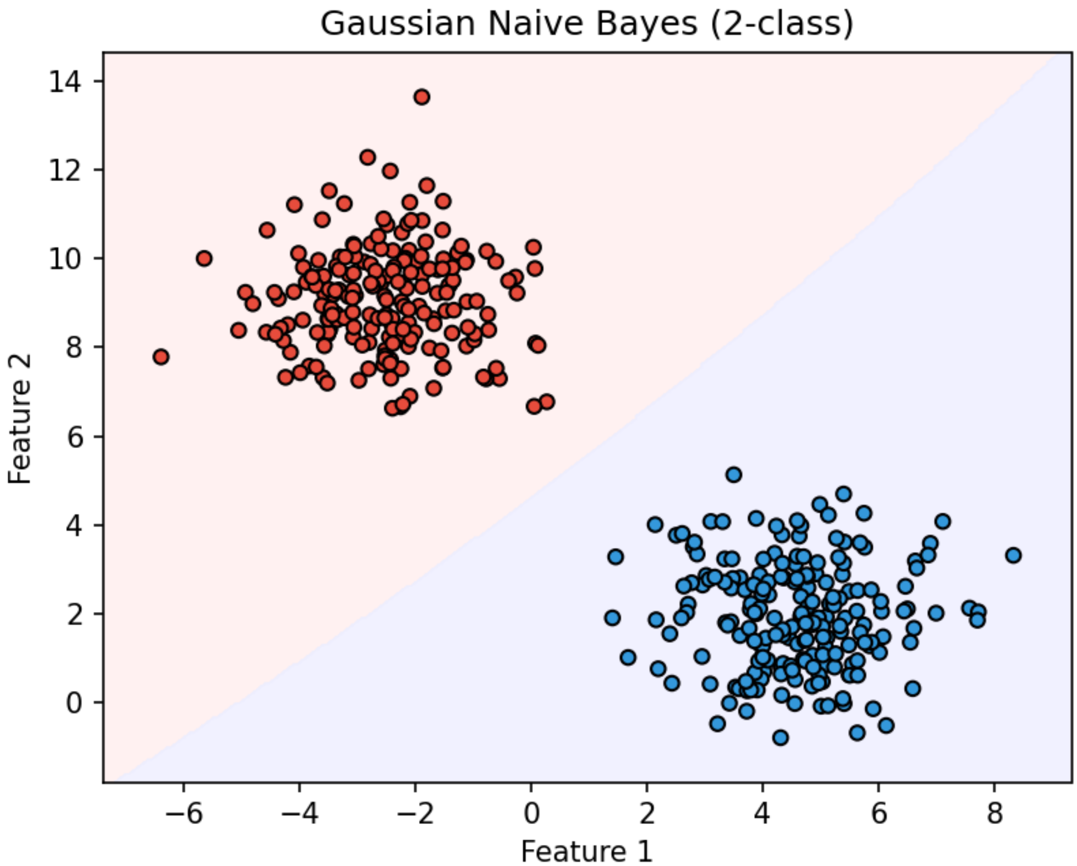

Figure 1: Gaussian NB decision boundary (2-class).

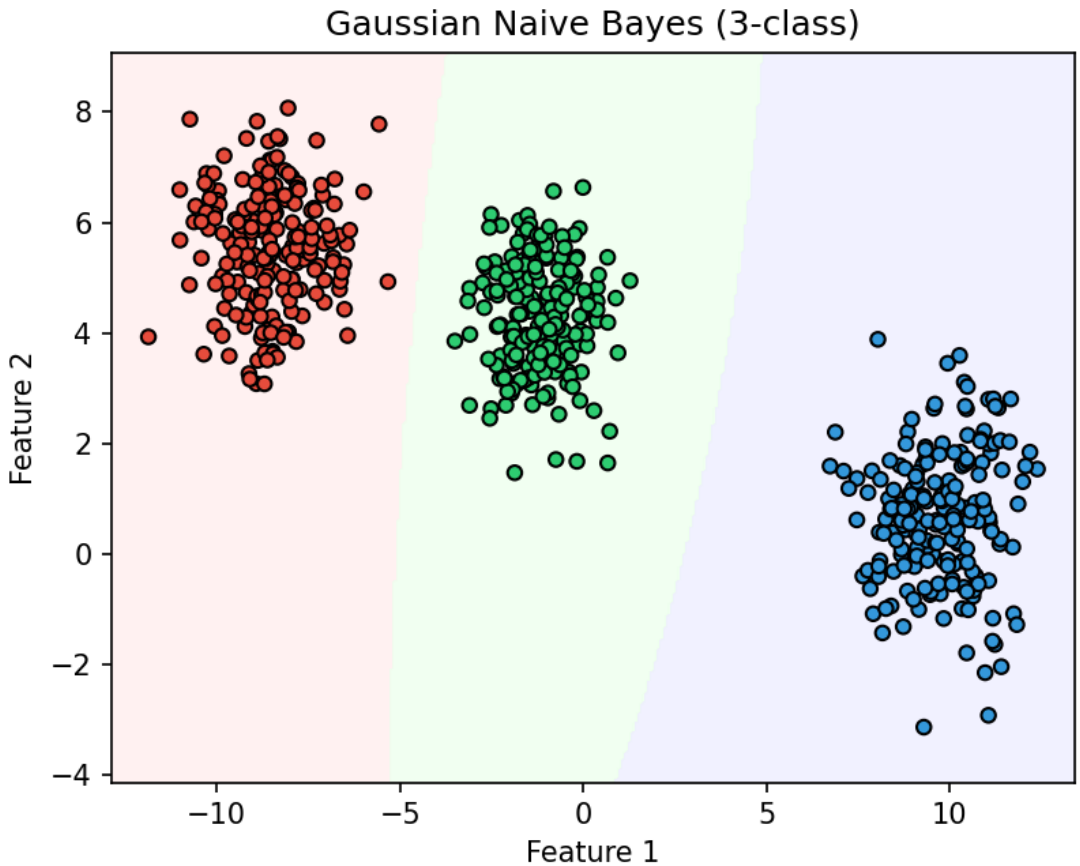

Figure 2: Gaussian NB decision regions (3-class).

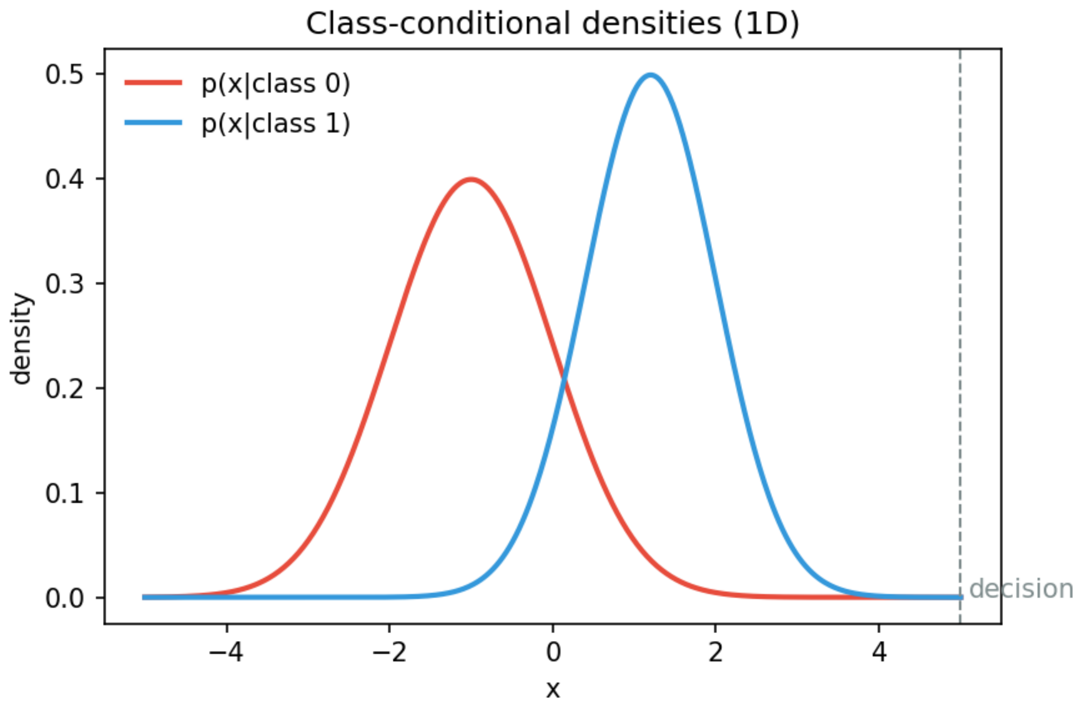

Figure 3: Class-conditional densities in 1D and a decision threshold.

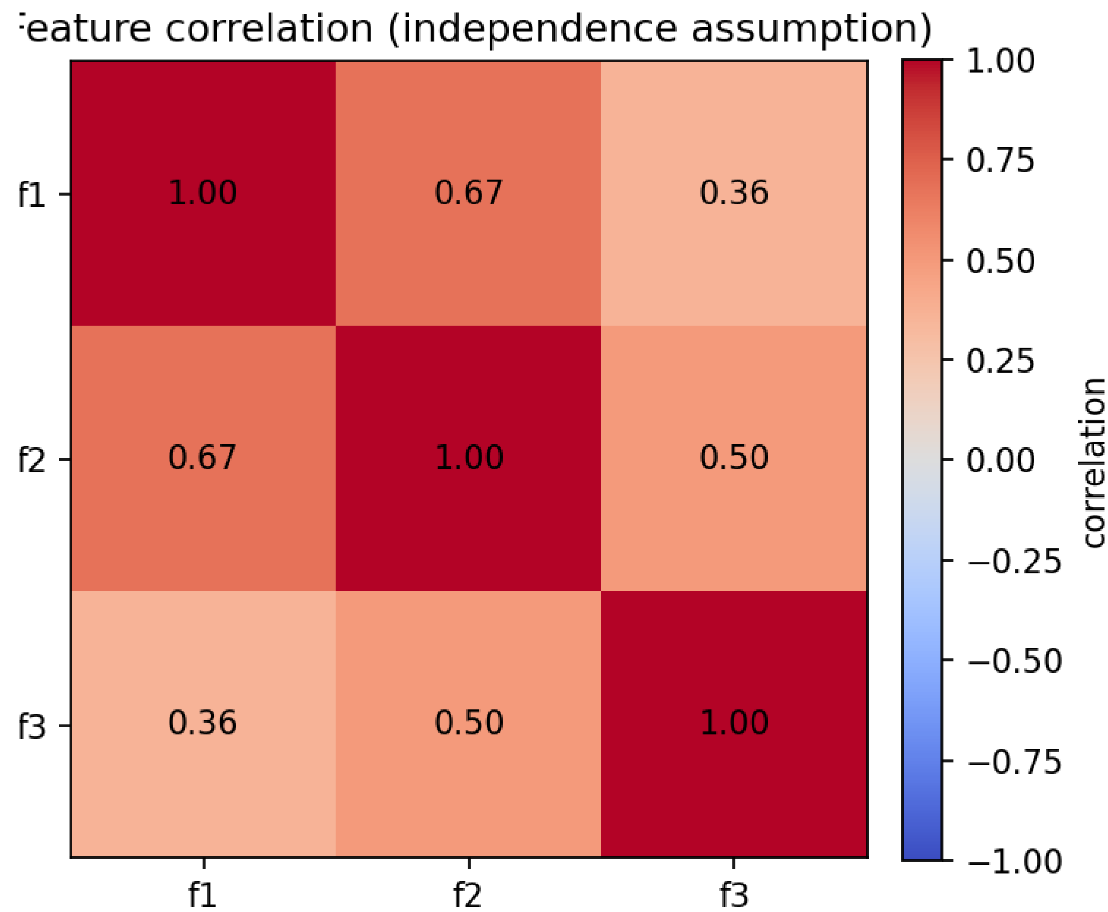

Figure 4: Feature correlation heatmap (independence assumption illustration).

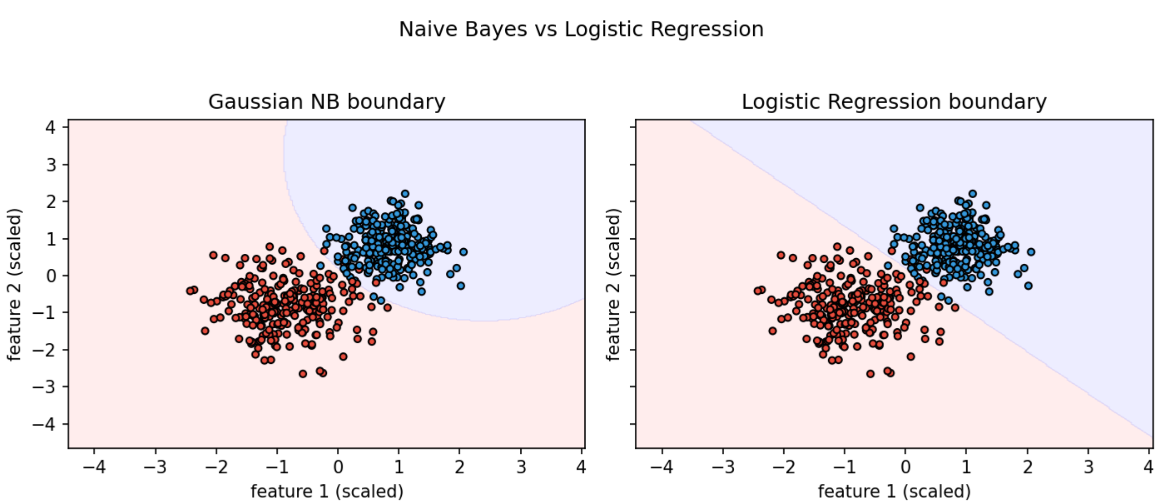

Figure 5: Decision boundary comparison: Gaussian NB vs Logistic Regression.

6 Summary

Naïve Bayes offers a fast, interpretable baseline. Its core idea is simple—combine class priors with per-feature likelihoods under conditional independence. While the assumption is often violated, NB remains competitive on certain problems and serves as a strong baseline against more flexible discriminative models.