

# 朴素贝叶斯：理论与实践

2025 年 9 月 9 日

## 目录

1	引言	1
2	原理与公式	1
3	应用场景与要点	2
4	Python 实战	2
5	结果	3
6	总结	6

## 1 引言

朴素贝叶斯（Naïve Bayes, NB）是在条件独立假设下建立的概率分类器族：

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

其中  $y$  为类别， $\mathbf{x} = (x_1, \dots, x_d)$  为特征。尽管独立性假设较强，NB 在高维稀疏特征（如文本）等任务中常表现稳健，且训练、预测效率较高。

## 2 原理与公式

以高斯朴素贝叶斯（Gaussian NB）为例，对于连续特征，假设对每个类别  $c \in \{1, \dots, C\}$  与每个特征  $j$  有：

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

则条件似然分解为  $p(\mathbf{x} | y = c) = \prod_j \mathcal{N}(x_j; \mu_{c,j}, \sigma_{c,j}^2)$ 。结合先验  $p(y = c)$ ，(未归一化的) 对数后验为：

$$\log p(y = c | \mathbf{x}) \propto \log p(y = c) + \sum_{j=1}^d \log \mathcal{N}(x_j; \mu_{c,j}, \sigma_{c,j}^2) \quad (3)$$

$$\propto \log p(y = c) - \sum_{j=1}^d \left[ \frac{1}{2} \log(2\pi\sigma_{c,j}^2) + \frac{(x_j - \mu_{c,j})^2}{2\sigma_{c,j}^2} \right]. \quad (4)$$

预测类别为  $\hat{y} = \arg \max_c \log p(y = c | \mathbf{x})$ 。参数估计可由各类别内样本的均值与方差直接得到。

**备注** 变体包括：连续特征的 Gaussian NB；计数/二值特征的 Multinomial/Bernoulli NB（常配合拉普拉斯平滑）。若需使用概率值做后续决策，建议做概率校准。

### 3 应用场景与要点

- **适用场景**：高维稀疏文本（BOW/TF-IDF）、简单传感器数据、作为强基线。
- **预处理**：Gaussian NB 建议对连续特征做标准化；文本常用计数或 TF-IDF（Multinomial NB）。
- **类别先验**：可用经验频率或领域知识设定。
- **独立性假设**：特征强相关时性能可能下降；建议与逻辑回归/线性 SVM 等对比。
- **评估**：采用交叉验证对比不同模型与超参数。

## 4 Python 实战

在章节目录内运行下述脚本，图片将保存到本目录下的 `figures/`：

Listing 1: 生成朴素贝叶斯配图

```
1 # 在 4_Naive Bayes 目录中执行：
2 python gen_naive_bayes_figures.py
```

Listing 2: `gen_naive_bayes_figures.py` 源码

```
1 """
2 Figure generator for the Naive Bayes chapter.
3
4 Generates illustrative figures and saves them into the local 'figures/'
   folder.
```

```
5
6 Requirements:
7 - Python 3.8+
8 - numpy, matplotlib, scikit-learn
9
10 Install (if needed):
11     pip install numpy matplotlib scikit-learn
12
13 This script avoids newer or experimental APIs to stay compatible with
14     older
15 versions of the dependencies.
16 """
17
18 from __future__ import annotations
19
20 import os
21 import math
22 import numpy as np
23 import matplotlib.pyplot as plt
24 from matplotlib.colors import ListedColormap
25
26 try:
27     from sklearn.datasets import make_blobs
28     from sklearn.naive_bayes import GaussianNB
29     from sklearn.linear_model import LogisticRegression
30     from sklearn.preprocessing import StandardScaler
31 except Exception as e:
32     raise SystemExit(
33         "Missing scikit-learn dependency. Please install with: pip
34         install scikit-learn"
35     )
36
37 def _ensure_figures_dir(path: str | None = None) -> str:
38     """Create figures directory under this chapter regardless of CWD.
39
40     If `path` is None, resolve to `<this_file_dir>/figures`.
41     """
42     if path is None:
43         base = os.path.dirname(os.path.abspath(__file__))
44         path = os.path.join(base, "figures")
45     os.makedirs(path, exist_ok=True)
46     return path
```

```

46
47 def _plot_decision_boundary(ax, clf, X, y, title: str, cmap_light,
    cmap_bold):
48     # Create a mesh grid for decision surface
49     x_min, x_max = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
50     y_min, y_max = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0
51     xx, yy = np.meshgrid(
52         np.linspace(x_min, x_max, 300), np.linspace(y_min, y_max, 300)
53     )
54     Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
55     Z = Z.reshape(xx.shape)
56
57     ax.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.8, levels=np.unique
        (Z).size)
58     # Training points
59     scatter = ax.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
        edgecolors="k", s=25)
60     ax.set_xlabel("Feature 1")
61     ax.set_ylabel("Feature 2")
62     ax.set_title(title)
63     return scatter
64
65
66 def fig_gnb_decision_boundary_2class(out_dir: str) -> str:
67     np.random.seed(42)
68     X, y = make_blobs(n_samples=400, centers=2, cluster_std=[1.2, 1.2],
        random_state=42)
69
70     clf = GaussianNB()
71     clf.fit(X, y)
72
73     cmap_light = ListedColormap(["#FFEEEE", "#EEEEFF"])
74     cmap_bold = ListedColormap(["#E74C3C", "#3498DB"])
75
76     fig, ax = plt.subplots(figsize=(5.5, 4.5), dpi=150)
77     _plot_decision_boundary(ax, clf, X, y, "Gaussian Naive Bayes (2-
        class)", cmap_light, cmap_bold)
78     out_path = os.path.join(out_dir, "gnb_decision_boundary_2class.png"
        )
79     fig.tight_layout()
80     fig.savefig(out_path)
81     plt.close(fig)
82     return out_path

```

```

83
84
85 def fig_gnb_decision_boundary_3class(out_dir: str) -> str:
86     np.random.seed(7)
87     X, y = make_blobs(
88         n_samples=600,
89         centers=3,
90         cluster_std=[1.1, 1.0, 1.2],
91         random_state=7,
92     )
93
94     clf = GaussianNB()
95     clf.fit(X, y)
96
97     cmap_light = ListedColormap(["#FFEEEE", "#EEFFEE", "#EEEEFF"])
98     cmap_bold = ListedColormap(["#E74C3C", "#2ECC71", "#3498DB"])
99
100    fig, ax = plt.subplots(figsize=(5.5, 4.5), dpi=150)
101    _plot_decision_boundary(ax, clf, X, y, "Gaussian Naive Bayes (3-
102        class)", cmap_light, cmap_bold)
103    out_path = os.path.join(out_dir, "gnb_decision_boundary_3class.png")
104
105    fig.tight_layout()
106    fig.savefig(out_path)
107    plt.close(fig)
108    return out_path
109
110 def _gaussian_pdf(x: np.ndarray, mu: float, sigma: float) -> np.ndarray
111 :
112     coef = 1.0 / (math.sqrt(2.0 * math.pi) * sigma)
113     return coef * np.exp(-0.5 * ((x - mu) / sigma) ** 2)
114
115 def fig_class_conditional_densities_1d(out_dir: str) -> str:
116     # Two 1D Gaussians with equal priors
117     mu0, sigma0 = -1.0, 1.0
118     mu1, sigma1 = 1.2, 0.8
119     xs = np.linspace(-5, 5, 500)
120     p_x_c0 = _gaussian_pdf(xs, mu0, sigma0)
121     p_x_c1 = _gaussian_pdf(xs, mu1, sigma1)
122
123     # Decision threshold where  $p(x|c0) = p(x|c1)$ 

```

```

123     # For illustration, compute numerically
124     idx = np.argmin(np.abs(p_x_c0 - p_x_c1))
125     x_star = xs[idx]
126
127     fig, ax = plt.subplots(figsize=(6, 4), dpi=150)
128     ax.plot(xs, p_x_c0, label="p(x|class 0)", color="#E74C3C", lw=2)
129     ax.plot(xs, p_x_c1, label="p(x|class 1)", color="#3498DB", lw=2)
130     ax.axvline(x_star, color="#7F8C8D", ls="--", lw=1)
131     ax.text(x_star + 0.1, max(p_x_c0[idx], p_x_c1[idx]) * 0.9, "
            decision", color="#7F8C8D")
132     ax.set_xlabel("x")
133     ax.set_ylabel("density")
134     ax.set_title("Class-conditional densities (1D)")
135     ax.legend(frameon=False)
136     out_path = os.path.join(out_dir, "class_conditional_densities_1d.
            png")
137     fig.tight_layout()
138     fig.savefig(out_path)
139     plt.close(fig)
140     return out_path
141
142
143 def fig_feature_independence_heatmap(out_dir: str) -> str:
144     # Create 3 correlated features to illustrate independence
145     # assumption violation
146     np.random.seed(123)
147     mean = np.array([0.0, 0.0, 0.0])
148     cov = np.array(
149         [
150             [1.0, 0.7, 0.4],
151             [0.7, 1.0, 0.5],
152             [0.4, 0.5, 1.0],
153         ]
154     )
155     X = np.random.multivariate_normal(mean, cov, size=1000)
156     # Empirical correlation matrix
157     C = np.corrcoef(X, rowvar=False)
158
159     fig, ax = plt.subplots(figsize=(4.8, 4.2), dpi=160)
160     im = ax.imshow(C, cmap="coolwarm", vmin=-1, vmax=1)
161     for i in range(C.shape[0]):
162         for j in range(C.shape[1]):
163             ax.text(j, i, f"{C[i, j]:.2f}", ha="center", va="center",

```

```

        color="black")
163 ax.set_xticks([0, 1, 2])
164 ax.set_yticks([0, 1, 2])
165 ax.set_xticklabels(["f1", "f2", "f3"])
166 ax.set_yticklabels(["f1", "f2", "f3"])
167 ax.set_title("Feature correlation (independence assumption)")
168 fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04, label="
    correlation")
169 out_path = os.path.join(out_dir, "feature_independence_heatmap.png"
    )
170 fig.tight_layout()
171 fig.savefig(out_path)
172 plt.close(fig)
173 return out_path
174
175
176 def fig_gnb_vs_logreg_boundary(out_dir: str) -> str:
177     # Dataset with partially overlapping Gaussians
178     np.random.seed(0)
179     X, y = make_blobs(n_samples=500, centers=[(-2, -2), (2.5, 2.0)],
        cluster_std=[1.6, 1.2], random_state=0)
180
181     scaler = StandardScaler()
182     Xs = scaler.fit_transform(X)
183
184     gnb = GaussianNB().fit(Xs, y)
185     # Use lbfgs which supports multinomial/binary and is widely
        available
186     lr = LogisticRegression(solver="lbfgs", max_iter=1000).fit(Xs, y)
187
188     x_min, x_max = Xs[:, 0].min() - 2.0, Xs[:, 0].max() + 2.0
189     y_min, y_max = Xs[:, 1].min() - 2.0, Xs[:, 1].max() + 2.0
190     xx, yy = np.meshgrid(np.linspace(x_min, x_max, 300), np.linspace(
        y_min, y_max, 300))
191     grid = np.c_[xx.ravel(), yy.ravel()]
192     Z_gnb = gnb.predict(grid).reshape(xx.shape)
193     Z_lr = lr.predict(grid).reshape(xx.shape)
194
195     fig, axes = plt.subplots(1, 2, figsize=(9.5, 4.2), dpi=150, sharex=
        True, sharey=True)
196     for ax, Z, title in [
197         (axes[0], Z_gnb, "Gaussian NB boundary"),
198         (axes[1], Z_lr, "Logistic Regression boundary"),

```

```

199     ]:
200         ax.contourf(xx, yy, Z, alpha=0.25, levels=np.unique(y).size,
201                    cmap=ListedColormap(["#FFBBBB", "#BBBBFF"]))
202         ax.scatter(Xs[:, 0], Xs[:, 1], c=y, s=15, cmap=ListedColormap([
203             "#E74C3C", "#3498DB"]), edgecolors="k")
204         ax.set_title(title)
205         ax.set_xlabel("feature 1 (scaled)")
206         ax.set_ylabel("feature 2 (scaled)")
207         fig.suptitle("Naive Bayes vs Logistic Regression")
208         out_path = os.path.join(out_dir, "gnb_vs_logreg_boundary.png")
209         fig.tight_layout(rect=[0, 0.03, 1, 0.95])
210         fig.savefig(out_path)
211         plt.close(fig)
212         return out_path
213
214 def main():
215     # Always save figures inside the current chapter directory
216     out_dir = _ensure_figures_dir(None)
217     generators = [
218         fig_gnb_decision_boundary_2class,
219         fig_gnb_decision_boundary_3class,
220         fig_class_conditional_densities_1d,
221         fig_feature_independence_heatmap,
222         fig_gnb_vs_logreg_boundary,
223     ]
224
225     print("Generating figures into:", os.path.abspath(out_dir))
226     for gen in generators:
227         try:
228             path = gen(out_dir)
229             print("Saved:", path)
230         except Exception as e:
231             print("Failed generating", gen.__name__, ":", e)
232
233 if __name__ == "__main__":
234     main()

```



## 5 结果

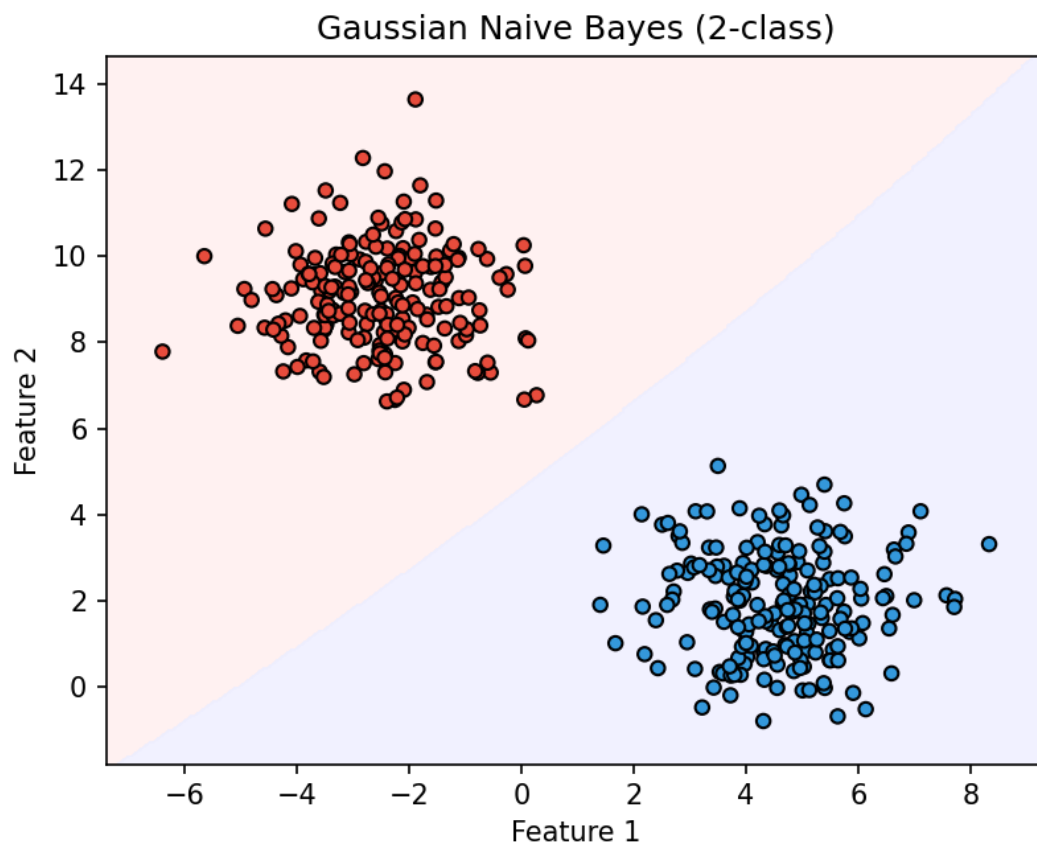


图 1: Gaussian NB 分类边界 (两类)。

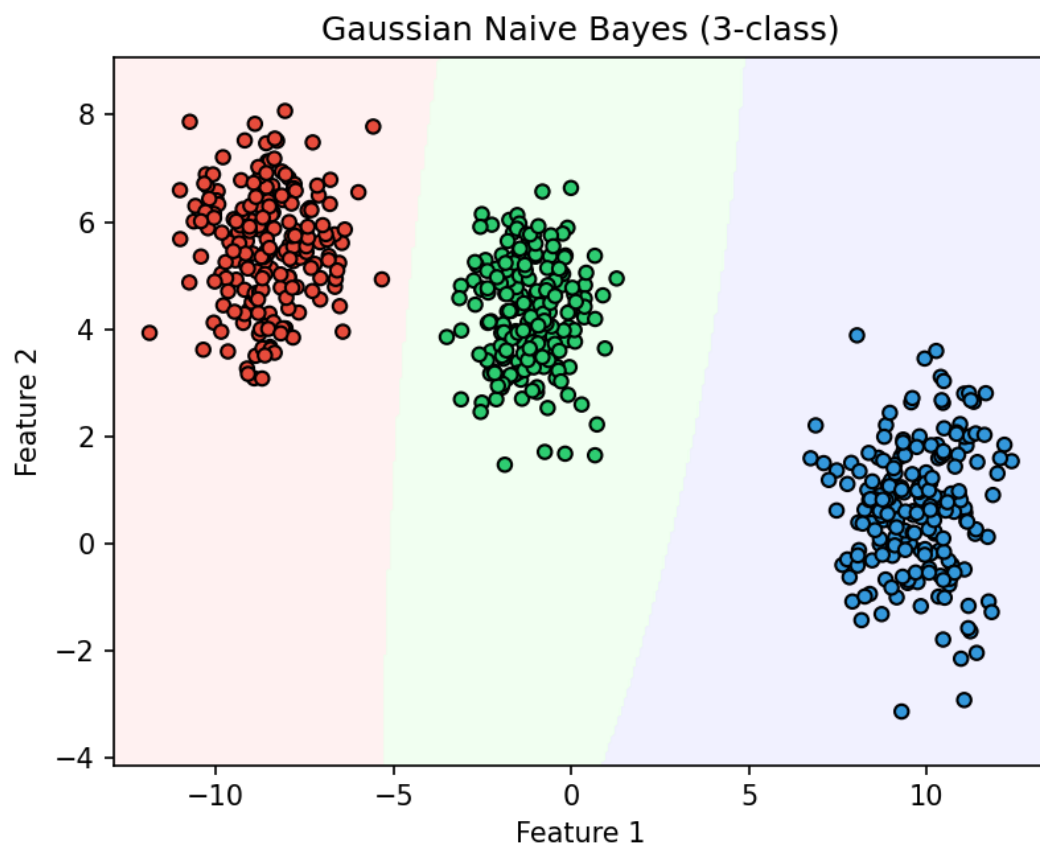


图 2: Gaussian NB 决策区域 (三类)。

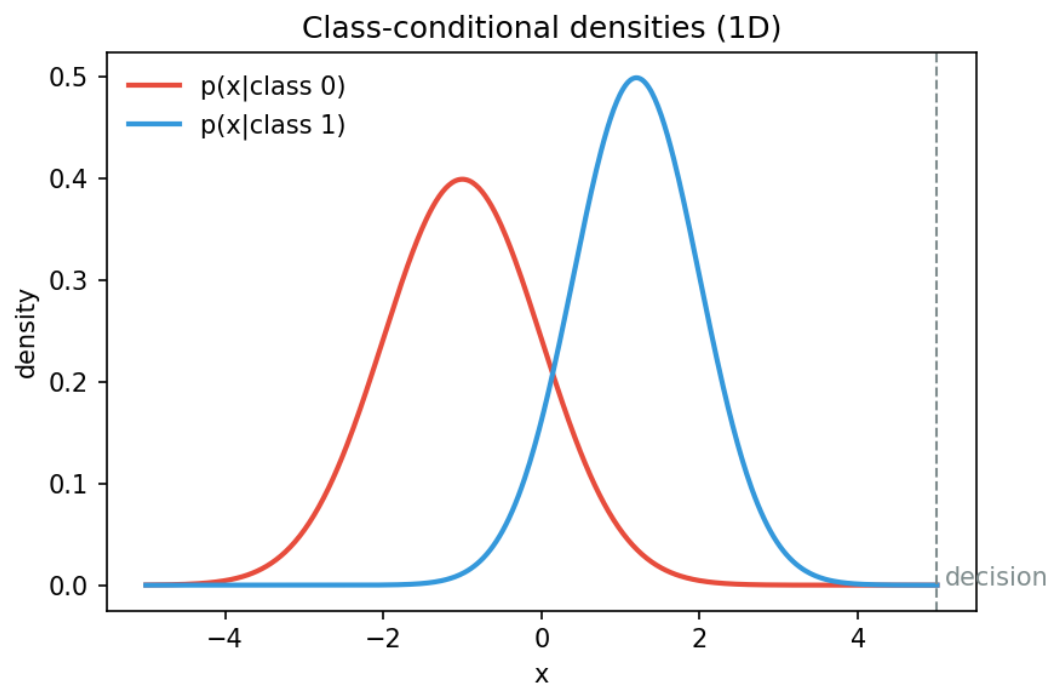


图 3: 一维类别条件密度与决策阈值。

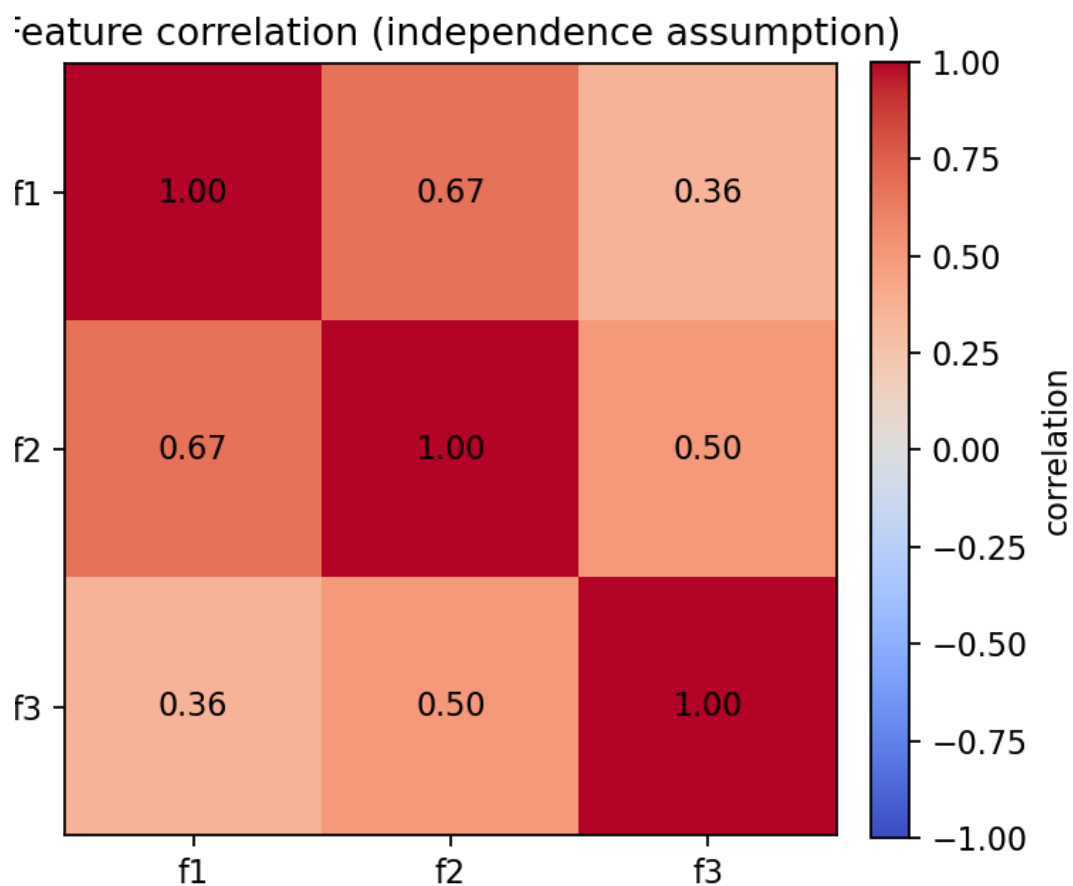


图 4: 特征相关性热力图（独立性假设示意）。

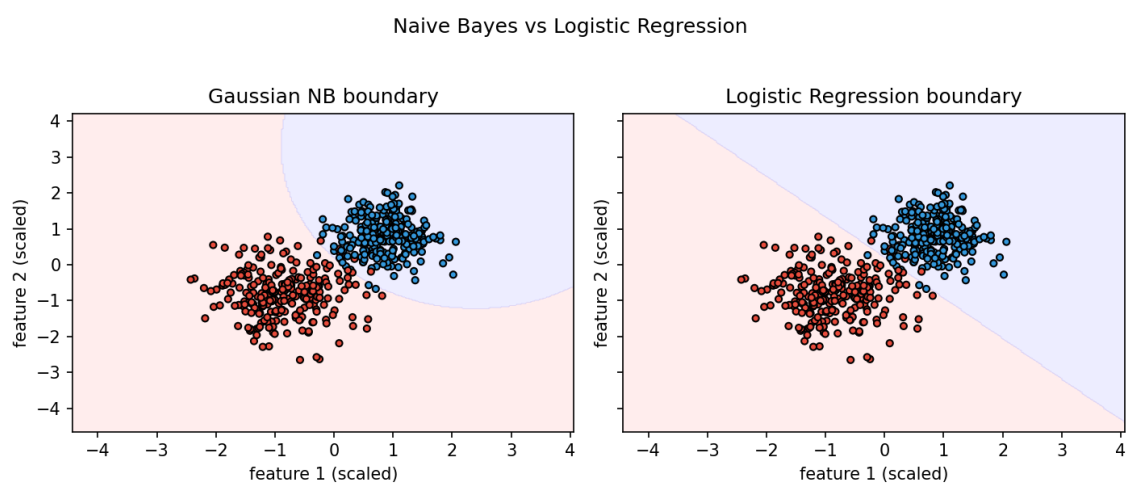


图 5: Gaussian NB 与逻辑回归的决策边界对比。

## 6 总结

朴素贝叶斯以简洁可解释、训练与预测高效为特点：核心是先验与逐特征似然的乘积（条件独立）。虽然假设并非总成立，它依然是可靠的基线模型，常用于与更强的判别式模型进行对比。