Naive Bayes Classifier – Complete Beginner-to-Advanced Revision Guide

---

1. Introduction

What is Naive Bayes?

The **Naive Bayes (NB) classifier** is a **supervised learning algorithm** based on **Bayes' Theorem**, used primarily for **classification**.

It is called *naive* because it assumes **conditional independence** between features given the class label — an assumption that is often unrealistic, yet works surprisingly well in practice.

Key Characteristics

- Probabilistic model
- Fast and scalable
- Performs well on high-dimensional data
- Widely used in **text classification**

Real-World Applications

- Spam filtering
- Sentiment analysis
- Document categorisation
- Medical diagnosis
- Recommendation systems

---

2. Mathematical Foundations

2.1 Bayes' Theorem

Bayes' theorem states:

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

Where: - $P(y)$: Prior probability of class - $P(x \mid y)$: Likelihood - $P(x)$: Evidence (normalising constant) - $P(y \mid x)$: Posterior probability

---

2.2 Classification Rule

For classification, the denominator is constant, so we compute:

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

2.3 Naive Conditional Independence Assumption

For feature vector:

$$ x = (x_1, x_2, \dots, x_n) $$

Naive Bayes assumes:

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

This simplifies computation drastically.

---

3. Types of Naive Bayes

3.1 Gaussian Naive Bayes (Continuous Features)

Assumes each feature follows a **normal distribution**:

$$ P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) $$

3.2 Multinomial Naive Bayes (Counts / Text)

Used for word counts:

$$ P(x_i \mid y) = \frac{N_{iy} + \alpha}{N_y + \alpha d} $$

Where: - α: Laplace smoothing - d: vocabulary size

3.3 Bernoulli Naive Bayes (Binary Features)

Used when features are binary:

$$ P(x_i \mid y) = p^{x_i}(1 - p)^{1 - x_i} $$

---

4. Model Training

4.1 Estimating Priors

$$ P(y) = \frac{N_y}{N} $$

4.2 Estimating Likelihoods

- Gaussian NB: estimate μ and σ^2
- Multinomial NB: estimate word probabilities

4.3 Log-Probability for Numerical Stability

Instead of multiplying probabilities:

$$ \log P(y \mid x) = \log P(y) + \sum_{i=1}^n \log P(x_i \mid y) $$

---

5. Step-by-Step Example

Example: Spam Classification

Given words: `free`, `win`, `offer`

1. Compute prior probabilities
2. Compute word likelihoods per class
3. Multiply (or sum logs)
4. Choose class with maximum posterior

---

6. Model Evaluation

Confusion Matrix

Actual \ Predicted	Spam	Not Spam
Spam	TP	FN
Not Spam	FP	TN

Evaluation Metrics

- Accuracy
- Precision
- Recall
- F1-score
- ROC-AUC

---

7. Interpretation

- High $P(x_i \mid y) \Rightarrow$ feature strongly associated with class
- Prior reflects class imbalance
- NB outputs probabilities (unlike SVM)

---

8. Assumptions

- Conditional independence
- Correct likelihood distribution
- Features equally important

---

9. Common Pitfalls & Misconceptions

- ❌ Assuming independence holds in reality
- ❌ Using Gaussian NB for categorical data
- ❌ Forgetting Laplace smoothing
- ❌ Zero probabilities without smoothing
- ❌ Misinterpreting posterior as confidence

---

10. Python Implementation

10.1 From Scratch (Gaussian NB)

```
import numpy as np

class GaussianNB:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.mean = {}
        self.var = {}
        self.prior = {}

        for c in self.classes:
            X_c = X[y == c]
            self.mean[c] = X_c.mean(axis=0)
            self.var[c] = X_c.var(axis=0)
            self.prior[c] = X_c.shape[0] / X.shape[0]

    def predict(self, X):
        y_pred = []
        for x in X:
            posteriors = []
            for c in self.classes:
                prior = np.log(self.prior[c])
                likelihood = -0.5 * np.sum(np.log(2 * np.pi * self.var[c]))
                             -0.5 * np.sum(((x - self.mean[c]) ** 2) /
self.var[c])
                posteriors.append(prior + likelihood)
            y_pred.append(self.classes[np.argmax(posteriors)])
        return np.array(y_pred)
```

10.2 Using scikit-learn

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = GaussianNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

10.3 Probability Visualization

```
import matplotlib.pyplot as plt

plt.hist(model.predict_proba(X_test)[:,1], bins=20)
plt.title("Predicted Probabilities")
plt.show()
```

---

11. When NOT to Use Naive Bayes

- Strong feature dependence
- Very small datasets
- Continuous non-Gaussian features (without preprocessing)

---

12. Best Practices

- Choose correct NB variant
- Use Laplace smoothing
- Scale continuous features
- Combine with TF-IDF for text

---

13. Summary

Naive Bayes is: - Simple - Fast - Probabilistic - Surprisingly powerful

It is especially strong for: - NLP - High-dimensional sparse data

---

End of Revision Guide