Naive Bayes: Regression & Classification Cheat Sheet

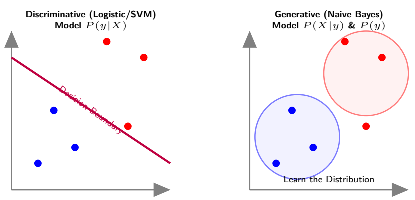
Probabilistic Machine Learning: Theory, Implementation, and Best Practices

1 Introduction

Definition: A family of probabilistic algorithms based on applying Bayes' theorem with a strong ("naive") assumption of independence between features.

Why "Naive"? It assumes all features are independent of each other given the class label. In reality, features are often correlated (e.g., "San" and "Francisco"), but NB ignores this to simplify computation.

Model Type:



- **Generative**: Models how the data is generated ($P(X|y)$). Uses prior probability.
- **Discriminative**: Models the boundary directly ($P(y|X)$).

2 Mathematical Foundation

$$\underbrace{P(y|X)}_{\text{Posterior}} = \frac{\overbrace{P(X|y)}^{\text{Likelihood}} \cdot \overbrace{P(y)}^{\text{Prior}}}{\underbrace{P(X)}_{\text{Evidence}}}$$

Core Formula:

$$P(y|x_1, \ldots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(X)}$$

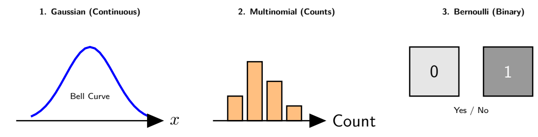
Components:

- **Prior** $P(y)$: Probability of class y before seeing data.
- **Likelihood** $P(x_i|y)$: Probability of feature x_i appearing in class y.
- **Posterior** $P(y|X)$: The probability we want to calculate.

Decision Rule:

$$\hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i|y)$$

3 Types of Naive Bayes



1. Gaussian NB:

- **Use Case**: Continuous features (e.g., Iris data, sensor readings).
- **Assumption**: Features follow a Normal (Gaussian) distribution.

2. Multinomial NB:

- **Use Case**: Discrete counts (e.g., Word counts in text).
- **Assumption**: Data follows a Multinomial distribution.

3. Bernoulli NB:

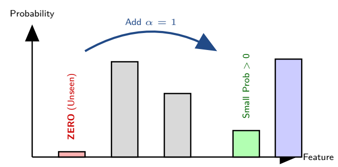
- **Use Case**: Binary/Boolean features (e.g., Word presence/absence).
- **Assumption**: Data is binary (0/1).

4 Data Requirements

Features:

- **Continuous**: Use GaussianNB.
- **Categorical**: Convert to counts/One-Hot and use MultinomialNB.
- **Text**: Bag of Words or TF-IDF.

Laplace Smoothing (Additive Smoothing): Solves the "Zero Probability" problem. If a word never appears in training, Likelihood = 0, killing the whole probability product.



$$P(x_i|y) = \frac{count(x_i, y) + \alpha}{count(y) + \alpha \cdot N_{features}}$$

where $\alpha = 1$ is standard Laplace smoothing.

5 Classification Implementation

Scenario: Spam Detection (Text).

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline

# 1. Data
X = ["offer is secret", "click secret link", "meeting today"]
y = [1, 1, 0] # 1=Spam, 0=Ham

# 2. Split
X_train, X_test, y_train, y_test = train_test_split(X, y)

# 3. Pipeline (Vectorize -> Model)
model = make_pipeline(
    CountVectorizer(), # Convert text to counts
    MultinomialNB(alpha=1.0) # Naive Bayes
)

# 4. Train & Predict
model.fit(X_train, y_train)
preds = model.predict(X_test)
print(preds)
```

6 Regression with Bayes

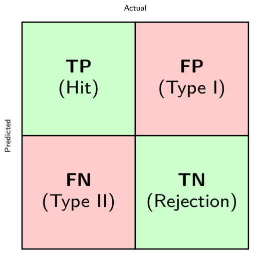
Standard Naive Bayes is a **Classifier**. For Regression, we use **Bayesian Ridge** or **Gaussian Process**. It assumes the target y comes from a probability distribution.

Bayesian Ridge: Assumes y is Gaussian distributed around Xw. It estimates a distribution over the weights w, not just point estimates.

```python
from sklearn.linear_model import BayesianRidge

# 1. Data
X = [[0, 0], [1, 1], [2, 2]]
y = [0, 1, 2]

# 2. Model
reg = BayesianRidge()

# 3. Train
reg.fit(X, y)

# 4. Predict (Returns Mean and Std Dev)
mean, std = reg.predict([[1, 0]], return_std=True)
print(f"Pred: {mean:.2f} +/- {std:.2f}")
```

7 Performance Metrics

Classification:



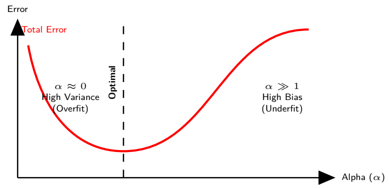
- **Accuracy**: % Correct.
- **Precision**: P(True Spam | Pred Spam). Critical for spam filters.
- **Recall**: P(Pred Spam | True Spam).
- **Log Loss**: Penalty for confident wrong answers.

Regression:

- **MSE**: Mean Squared Error.
- **R2 Score**: Variance explained.

8 Tuning

Smoothing (α):



- $\alpha = 0$: No smoothing (Risk of zero prob).
- $\alpha = 1$: Laplace smoothing (Default).
- $\alpha > 1$: High smoothing (High Bias, Low Variance).

Priors: Can explicitly set 'priors=[0.2, 0.8]' if you know the class balance differs from the training set.

9 Practical Use Cases

- **Spam Filtering**: The classic use case. Fast, handles high-dim text well.
- **Sentiment Analysis**: Positive/Negative classification.
- **Real-time Prediction**: Extremely fast inference speed.
- **Baseline Model**: Always run NB first to set a baseline for complex models.

10 Common Mistakes

- **Correlated Features**: "New" and "York" in "New York" are correlated. NB counts them twice, inflating confidence.
- **Zero Probability**: Forgetting smoothing (α) causes the model to crash on unseen words.
- **Wrong Distribution**: Using GaussianNB on sparse word counts (Use Multinomial instead).

11 Comparison

Model	vs Naive Bayes
Logistic Reg	Discriminative. Slower. Handles correlation better.
SVM	Discriminative. Max margin. Slower. Better acc.
Trees	Non-linear. Handles interactions. Much slower.

12 Summary

- **Choose When**: High dimensions (Text), small data, need speed, need baseline.
- **Avoid When**: Features are highly correlated, huge training data available (DL might be better).
- **Key Param**: Alpha (α) for smoothing.