Naive Bayes from Scratch

Bhanu Prasanna Koppolu

Table of contents

> **Note**
>
> Full implementation available at GitHub - ML from Scratch

Naive Bayes

For suppose we currently have 2 classes namely Spam vs Not Spam. We first compute the probabilities:

$$P(Y = \text{Spam}|X) = \text{some value}$$
$$P(Y = \text{Not Spam}|X) = \text{some value}$$

The sum of these 2 probabilities is 1, because there are only 2 classes.

Bayes Theorem

$$P(y = k|\mathbf{x}) = \frac{P(\mathbf{x}|y = k) \cdot P(y = k)}{P(\mathbf{x})}$$

But we ignore $P(\mathbf{x})$ because it doesn't change at all, so we approximate:

$$P(y = k|\mathbf{x}) \propto P(\mathbf{x}|y = k) \cdot P(y = k)$$

Breaking Down the Terms

Term	Name	Description
$P(y = k\|\mathbf{x})$	**Posterior**	Probability of class k given features $\mathbf{x}$ (what we want)
$P(\mathbf{x}\|y = k)$	**Likelihood**	Probability of seeing features $\mathbf{x}$ in class k
$P(y = k)$	**Prior**	Probability of class k (before seeing any features)
$P(\mathbf{x})$	**Evidence**	Probability of features $\mathbf{x}$ (across all classes)

$P(\mathbf{x})$ is the same for all classes, so for comparison we can ignore it.

Likelihood Computation

$$P(\mathbf{x}|y = k) = \prod_{i=1}^{n} \mathcal{N}(x_i; \mu_{ik}, \sigma_i)$$

So, the log posterior:

$$\log(P(y = k|\mathbf{x})) = \arg\max \left(\log(P(y = k)) + \sum_{i=1}^{n} \log(\mathcal{N}(x_i; \mu_{ik}, \sigma_i)) \right)$$

The Naive Assumption

We assume features are **conditionally independent** given the class.

But usually the Naive assumption is almost always wrong! In real data, features are almost never conditionally independent.

Why Naive Bayes Still Works

1. **We only need ranking, not exact probabilities**:

$$\hat{y} = \arg\max_k P(y = k|\mathbf{x})$$

2. **Errors can cancel out**: Two features are positively correlated in both classes. If class 0 and class 1 joint probability is overestimated by similar amounts, the ratio stays approximately correct.
3. **High-dimensional spaces and strong signal vs weak correlation**

Parameters

So, at the end there are only 3 parameters that come out of Gaussian Naive Bayes:

- $\pi_c = P(y = c)$ - prior probability for class c
- μ_{cj} - mean of class c for each feature j
- σ^2_{cj} - variance of class c for each feature j

Implementation

```python
def naive_bayes(X, y):
    # Gaussian Naive Bayes
    X = X.copy()
    y = y.copy()

    X_shape = X.shape
    class_storage = dict()
    unique_y = np.unique(y)

    for c in unique_y:
        subset_y_c = X[y == c]
        m_c = subset_y_c.shape[0]
        pi_c = m_c / X_shape[0]        # Prior
        u_c = np.mean(subset_y_c, axis=0)   # Mean
        var_c = np.var(subset_y_c, axis=0)  # Variance

        class_storage[c] = {}
        class_storage[c]['pi'] = pi_c
        class_storage[c]['mu'] = u_c
        class_storage[c]['var'] = var_c

    return class_storage
```

```python
def evaluate(X, y, params):
    # Evaluate Gaussian Naive Bayes
    X = X.copy()
    s_c_dict = dict()

    for c in params.keys():
        pi_c = params[c]['pi']
        u_c = params[c]['mu']
        var_c = params[c]['var']

        # Gaussian PDF in log space
        s_c = np.log((1 / np.sqrt(2 * np.pi * var_c)) *
                     np.power(np.e, (-1 / 2) * (((X - u_c) ** 2) / var_c)))
        s_c = np.log(params[c]['pi']) + np.sum(s_c, axis=1)

        s_c_dict[c] = s_c

    y_hat = np.column_stack([s_c_dict[i] for i in s_c_dict.keys()])
    y_hat = np.vstack(y_hat.argmax(axis=1))

    return y_hat
```

Extensions

We can extend this to **Bernoulli** and **Multinomial** cases as well:

- **Bernoulli**: Features have only 0 or 1 for every feature column (binary labels for features)
- **Multinomial**: Features can be 0, 1, 2, 3, …, n for every feature column (count data)