Pseudocode

```
Algorithm: NaiveBayesClassifier

Input:
    X  (training data)
    y  (class labels)

Output:
    Model parameters (class priors, feature likelihoods)

Training Phase:
    For each class c in y:
        P(c) = count(c) / total samples
        For each feature f:
            Compute P(f | c)

Prediction Phase:
    For each test sample x:
        For each class c:
            score(c) = P(c)
            For each feature f in x:
                score(c) = score(c) * P(f | c)
        Predict class with maximum score
```

Python Code

```python
import numpy as np

class NaiveBayes:
    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] = np.mean(X_c, axis=0)
            self.var[c] = np.var(X_c, axis=0)
            self.prior[c] = X_c.shape[0] / X.shape[0]
```

```
    def gaussian_prob(self, x, mean, var):
        eps = 1e-6
        coeff = 1 / np.sqrt(2 * np.pi * var + eps)
        exponent = np.exp(-(x - mean) ** 2 / (2 * var + eps))
        return coeff * exponent

    def predict(self, X):
        predictions = []

        for x in X:
            posteriors = []
            for c in self.classes:
                prior = np.log(self.prior[c])
                likelihood = np.sum(
                    np.log(self.gaussian_prob(x, self.mean[c], self.var[c]))
                )
                posteriors.append(prior + likelihood)
            predictions.append(self.classes[np.argmax(posteriors)])

        return np.array(predictions)
```