Naive Bayes Classifier

A notebook covering the theory, mathematics, and Python implementation of the Naive Bayes classification algorithm.

1. Brief Idea

Naive Bayes is a family of simple probabilistic classifiers based on Bayes' theorem with the "naive" assumption that features are conditionally independent given the class. Despite its simplicity, it performs surprisingly well in many real-world applications, especially text classification.

2. Context and Backstory

Naive Bayes classifiers have been used since the 1950s and are foundational in information retrieval and spam filtering. Their efficiency and low data requirements make them popular for high-dimensional data, such as text and document classification. Extensions include multinomial, Bernoulli, and Gaussian variants.

3. Mathematical Derivations and Formulae

Given features $x_1, \ldots, x_n$ and class y, Bayes' theorem gives:

$$p(y|x_1, \ldots, x_n) \propto p(y) \prod_{j=1}^{n} p(x_j|y)$$

The "naive" assumption is that x_j are conditionally independent given y. For the multinomial variant (text),

$$p(y|\mathbf{x}) \propto p(y) \prod_{j=1}^{n} p(x_j|y)^{x_j}$$

Parameters are estimated from data using MLE or MAP.

4. Diagrams and Visual Aids

Figure: Naive Bayes graphical model (class node with arrows to feature nodes).

(Insert a plot or image here if desired.)

```
In [ ]: from sklearn.datasets import fetch_20newsgroups
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.metrics import accuracy_score

        # Load data (subset for speed)
        data = fetch_20newsgroups(subset='train', categories=['sci.space', 'rec.sport.baseball'], rem
        X = data.data
        y = data.target

        # Vectorize text
```

```
vectorizer = CountVectorizer(stop_words='english', max_features=1000)
X_vec = vectorizer.fit_transform(X)

# Train Naive Bayes
clf = MultinomialNB()
clf.fit(X_vec, y)

# Predict on training data
preds = clf.predict(X_vec)
print('Training accuracy:', accuracy_score(y, preds))
```

6. Frequently Asked Questions (FAQ)

1. **Why is it called "naive"?** Because it assumes feature independence, which is rarely true in practice.
2. **When does Naive Bayes perform well?** When features are nearly independent or in high-dimensional sparse data.
3. **What are common variants?** Multinomial, Bernoulli, and Gaussian Naive Bayes.
4. **Can Naive Bayes handle continuous features?** Yes, with Gaussian or kernel density variants.

7. Higher-Order Thinking Questions (HOTS)

- Derive the MAP estimate for the parameters of a multinomial Naive Bayes classifier.
- Discuss the impact of correlated features on Naive Bayes performance.
- How would you adapt Naive Bayes for continuous, non-Gaussian features?

In [ ]:
```
import numpy as np
from collections import Counter, defaultdict

# Example toy dataset: 2 classes, 2 documents per class
X = [
    'space rocket launch',
    'astronaut moon mission',
    'baseball bat home run',
    'pitcher strikes batter'
]
y = [0, 0, 1, 1]  # 0: space, 1: baseball

# Build vocabulary
vocab = sorted(set(word for doc in X for word in doc.split()))
vocab_idx = {word: i for i, word in enumerate(vocab)}

# Count word occurrences per class
class_word_counts = [np.zeros(len(vocab), dtype=int) for _ in range(2)]
class_counts = [0, 0]
for doc, label in zip(X, y):
    counts = Counter(doc.split())
    for word, cnt in counts.items():
        class_word_counts[label][vocab_idx[word]] += cnt
    class_counts[label] += 1

# Compute class priors and likelihoods (Laplace smoothing)
priors = [class_counts[c] / len(y) for c in range(2)]
likelihoods = [
    (class_word_counts[c] + 1) / (class_word_counts[c].sum() + len(vocab))
    for c in range(2)
]

def predict(doc):
    words = doc.split()
    log_probs = []
    for c in range(2):
        log_prob = np.log(priors[c])
        for word in words:
            if word in vocab_idx:
                log_prob += np.log(likelihoods[c][vocab_idx[word]])
        log_probs.append(log_prob)
    return np.argmax(log_probs)

# Test prediction
for doc in X:
    print(f"Doc: '{doc}' => Predicted class:", predict(doc))
```