Naive Bayes — Weather Classification (Rain? Yes/No)

Code Walkthrough

1. Dataset Setup

```
import math
from collections import Counter, defaultdict

DATA = [
    {"Outlook":"Sunny","Temperature":"Hot","Humidity":"High","Windy":False,"Rain":"No"},
    {"Outlook":"Sunny","Temperature":"Hot","Humidity":"High","Windy":True, "Rain":"No"},
    {"Outlook":"Overcast","Temperature":"Hot","Humidity":"High","Windy":False,"Rain":"Yes"},
    {"Outlook":"Rain","Temperature":"Mild","Humidity":"High","Windy":False,"Rain":"Yes"},
    {"Outlook":"Rain","Temperature":"Cool","Humidity":"Normal","Windy":False,"Rain":"Yes"},
    {"Outlook":"Rain","Temperature":"Cool","Humidity":"Normal","Windy":True, "Rain":"No"},
    {"Outlook":"Overcast","Temperature":"Cool","Humidity":"Normal","Windy":True,"Rain":"Yes"},
    {"Outlook":"Sunny","Temperature":"Mild","Humidity":"High","Windy":False,"Rain":"No"},
    {"Outlook":"Sunny","Temperature":"Cool","Humidity":"Normal","Windy":False,"Rain":"Yes"},
    {"Outlook":"Rain","Temperature":"Mild","Humidity":"Normal","Windy":False,"Rain":"Yes"},
    {"Outlook":"Sunny","Temperature":"Mild","Humidity":"Normal","Windy":True,"Rain":"Yes"},
    {"Outlook":"Overcast","Temperature":"Mild","Humidity":"High","Windy":True,"Rain":"Yes"},
    {"Outlook":"Overcast","Temperature":"Hot","Humidity":"Normal","Windy":False,"Rain":"Yes"},
    {"Outlook":"Rain","Temperature":"Mild","Humidity":"High","Windy":True,"Rain":"No"},
]

FEATURES = ["Outlook","Temperature","Humidity","Windy"]
TARGET = "Rain"
CLASSES = ["Yes","No"]
```

This section defines the dataset used to train and test the Naive Bayes classifier. Each dictionary in the list represents one day's weather conditions and whether it rained or not. The model uses these examples to learn patterns between the features (Outlook, Temperature, Humidity, Windy) and the target variable "Rain." The FEATURES, TARGET, and CLASSES lists define what the model will analyze and predict.

2. Feature Domains Function

```
def feature_domains(rows):
    dom = {f:set() for f in FEATURES}
    for r in rows:
        for f in FEATURES:
            dom[f].add(r[f])
    return {f: sorted(list(v)) for f,v in dom.items()}
```

This function identifies all possible values each feature can take. For example, "Outlook" can be Sunny, Overcast, or Rain. This information is important when applying Laplace smoothing, because it determines how many possible outcomes each feature has. The function returns a dictionary mapping each feature name to its unique value set.

3. Training Function — Laplace Smoothing

```
def fit_naive_bayes(rows, alpha=1.0):
    n = len(rows)
    domains = feature_domains(rows)

    y_counts = Counter(r[TARGET] for r in rows)
    num_classes = len(CLASSES)

    priors = {y: (y_counts[y] + alpha) / (n + alpha * num_classes) for y in CLASSES}

    fv_counts = defaultdict(int)
    y_total_for_feature = defaultdict(int)
    for r in rows:
        y = r[TARGET]
        for f in FEATURES:
            v = r[f]
            fv_counts[(f, v, y)] += 1
            y_total_for_feature[(f, y)] += 1

    cond = {}
    for f in FEATURES:
        k = len(domains[f])
        for y in CLASSES:
            denom = y_total_for_feature[(f, y)] + alpha * k
            for v in domains[f]:
                num = fv_counts[(f, v, y)] + alpha
                cond[(f, v, y)] = num / denom

    return priors, cond, domains
```

This function is the core of the model's training process. It calculates: - **Class priors (P(Y))** — the base probability of each class (rain or no rain). - **Conditional probabilities (P(X|Y))** — how likely each feature value is given that class. Laplace smoothing (alpha=1) prevents any probability from becoming zero when certain combinations of feature values and classes don't appear in the dataset. It ensures the model remains stable and generalizable.

4. Prediction Logic

```
def predict_proba(sample, priors, cond):
    log_post = {}
    for y in CLASSES:
        s = math.log(priors[y])
        for f in FEATURES:
            v = sample[f]
            p = cond.get((f, v, y), 1e-12)
            s += math.log(p)
        log_post[y] = s
    m = max(log_post.values())
    exps = {y: math.exp(log_post[y] - m) for y in CLASSES}
    Z = sum(exps.values())
    return {y: exps[y]/Z for y in CLASSES}

def predict(sample, priors, cond):
    probs = predict_proba(sample, priors, cond)
    y_hat = max(probs.items(), key=lambda kv: kv[1])[0]
    return y_hat, probs
```

These functions perform classification. For a new day's weather data, the model multiplies the prior by the likelihoods for each feature value. Logarithms are used to prevent floating-point underflow from small probability multiplications. The function then normalizes these values into readable probabilities (softmax). Finally, the most likely class label (Yes or No) is returned as the prediction.

5. Leave-One-Out Evaluation

```
def leave_one_out_accuracy(rows, alpha=1.0):
    correct = 0
    for i in range(len(rows)):
        train = rows[:i] + rows[i+1:]
        test = rows[i]
        priors, cond, _ = fit_naive_bayes(train, alpha=alpha)
        y_hat, _ = predict({f:test[f] for f in FEATURES}, priors, cond)
        correct += int(y_hat == test[TARGET])
    return correct / len(rows)
```

This function evaluates model accuracy using Leave-One-Out Cross-Validation. It removes one data point at a time, trains on the remaining 13, and tests on the excluded one. This process repeats for all 14 records. The function then calculates the overall proportion of correct predictions. This method is ideal for very small datasets.

6. Main Execution and sklearn Comparison

```
if __name__ == "__main__":
    ALPHA = 1.0
    priors, cond, domains = fit_naive_bayes(DATA, alpha=ALPHA)

    y_counts = Counter(r[TARGET] for r in DATA)
    total = len(DATA)
    print("Empirical class priors (unsmoothed):")
    for y in CLASSES:
        print(f"  P({y}) ≈ {y_counts[y]/total:.4f}  ({y_counts[y]}/{total})")

    print("\nSmoothed priors used by the model:")
    for y in CLASSES:
        print(f"  P({y}) = {priors[y]:.4f}")

    print("\nExample likelihoods (with smoothing):")
    for (f, v) in [("Outlook","Sunny"), ("Humidity","High")]:
        for y in CLASSES:
            print(f"  P({f}={v} | Y={y}) = {cond[(f, v, y)]:.4f}")

    new_day = {"Outlook":"Sunny","Temperature":"Mild","Humidity":"High","Windy":False}
    y_hat, probs = predict(new_day, priors, cond)
    print("\nNew day:", new_day)
    print("Predicted:", y_hat)
    print("Posterior probabilities:", {k: round(v,4) for k,v in probs.items()})

    acc = leave_one_out_accuracy(DATA, alpha=ALPHA)
    print(f"\nLeave-One-Out Accuracy on tiny dataset: {acc:.3f}")

    try:
        from sklearn.feature_extraction import DictVectorizer
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.model_selection import LeaveOneOut
        from sklearn.metrics import accuracy_score
        import numpy as np
```

```
        X_dicts = []
        for row in DATA:
            d = {}
            for k, v in row.items():
                if k == "Windy":
                    d[k] = str(v)
                elif k != "Rain":
                    d[k] = v
            X_dicts.append(d)
        y = [row["Rain"] for row in DATA]

        vec = DictVectorizer(sparse=True)
        X = vec.fit_transform(X_dicts)

        desired_priors = {"No": (y_counts["No"] + ALPHA) / (total + ALPHA*2),
                          "Yes": (y_counts["Yes"] + ALPHA) / (total + ALPHA*2)}

        loo = LeaveOneOut()
        preds, truth = [], []
        for train_idx, test_idx in loo.split(X):
            clf = MultinomialNB(alpha=ALPHA)
            clf.fit(X[train_idx], np.array(y)[train_idx])
            order = list(clf.classes_)
            priors_ordered = [desired_priors[c] for c in order]
            clf = MultinomialNB(alpha=ALPHA, class_prior=priors_ordered, fit_prior=False)
            clf.fit(X[train_idx], np.array(y)[train_idx])

            preds.append(clf.predict(X[test_idx])[0])
            truth.append(y[test_idx[0]])

        print("sklearn LOO accuracy:", accuracy_score(truth, preds))

    except ImportError:
        print("\n[sklearn check skipped — scikit-learn not installed]")
```

This final section runs the program and prints all results. It first calculates and displays the empirical and smoothed class priors, then evaluates example likelihoods. It predicts rainfall for a new day and computes overall accuracy using Leave-One-Out validation. The final block compares results to scikit-learn’s MultinomialNB implementation, ensuring that the hand-coded model aligns with an industry-standard version.