

```

import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.utils import resample
from sklearn.base import clone

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 1. Bagging
class Bagging:
    def __init__(self, base_estimator, n_estimators=10):
        self.base_estimator = base_estimator
        self.n_estimators = n_estimators
        self.models = []

    def fit(self, X, y):
        for _ in range(self.n_estimators):
            X_resampled, y_resampled = resample(X, y)
            model = clone(self.base_estimator)
            model.fit(X_resampled, y_resampled)
            self.models.append(model)

    def predict(self, X):
        predictions = np.array([model.predict(X) for model in self.models])
        return np.apply_along_axis(lambda x: np.bincount(x).argmax(), axis=0, arr=predictions)

# Bagging example
bagging_model = Bagging(DecisionTreeClassifier(), n_estimators=10)
bagging_model.fit(X_train, y_train)
bagging_predictions = bagging_model.predict(X_test)
print(f"Bagging Accuracy: {accuracy_score(y_test, bagging_predictions):.2f}")

# 2. Boosting (AdaBoost)
class AdaBoost:
    def __init__(self, base_estimator, n_estimators=50):
        self.base_estimator = base_estimator
        self.n_estimators = n_estimators
        self.models = []
        self.alphas = []

    def fit(self, X, y):
        n_samples = X.shape[0]
        w = np.ones(n_samples) / n_samples

        for _ in range(self.n_estimators):
            model = clone(self.base_estimator)
            model.fit(X, y, sample_weight=w)
            predictions = model.predict(X)
            error = np.sum(w * (predictions != y)) / np.sum(w)
            alpha = np.log((1 - error) / error) / 2
            w = w * np.exp(-alpha * (predictions == y))
            w = w / np.sum(w)
            self.models.append(model)
            self.alphas.append(alpha)

    def predict(self, X):
        model_preds = np.array([alpha * model.predict(X) for model, alpha in zip(self.models, self.alphas)])
        return np.apply_along_axis(lambda x: np.bincount(x.astype(int)).argmax(), axis=0, arr=model_preds) # Changed line. Cast x to int

# AdaBoost example
boosting_model = AdaBoost(DecisionTreeClassifier(max_depth=1), n_estimators=50)
boosting_model.fit(X_train, y_train)
boosting_predictions = boosting_model.predict(X_test)
print(f"Boosting Accuracy: {accuracy_score(y_test, boosting_predictions):.2f}")

# 3. Stacking
class Stacking:
    def __init__(self, base_estimators, meta_estimator):
        self.base_estimators = base_estimators
        self.meta_estimator = meta_estimator

    def fit(self, X, y):
        meta_features = []
        for model in self.base_estimators:
            model.fit(X, y)

```


```

        meta_features.append(model.predict(X))
    meta_features = np.array(meta_features).T
    self.meta_estimator.fit(meta_features, y)

    def predict(self, X):
        meta_features = np.array([model.predict(X) for model in self.base_estimators]).T
        return self.meta_estimator.predict(meta_features)

# Stacking example
base_models = [DecisionTreeClassifier(), LogisticRegression(max_iter=200)]
meta_model = LogisticRegression()
stacking_model = Stacking(base_models, meta_model)
stacking_model.fit(X_train, y_train)
stacking_predictions = stacking_model.predict(X_test)
print(f"Stacking Accuracy: {accuracy_score(y_test, stacking_predictions):.2f}")

```

 Bagging Accuracy: 1.00
 Boosting Accuracy: 0.42
 Stacking Accuracy: 1.00

```

from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

```

```

# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.title(title)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()

```

```

# Generate and plot confusion matrix for Bagging
plot_confusion_matrix(y_test, bagging_predictions, "Bagging Confusion Matrix")
print(classification_report(y_test, bagging_predictions))

```

```

# Generate and plot confusion matrix for Boosting
plot_confusion_matrix(y_test, boosting_predictions, "Boosting Confusion Matrix")
print(classification_report(y_test, boosting_predictions))

```

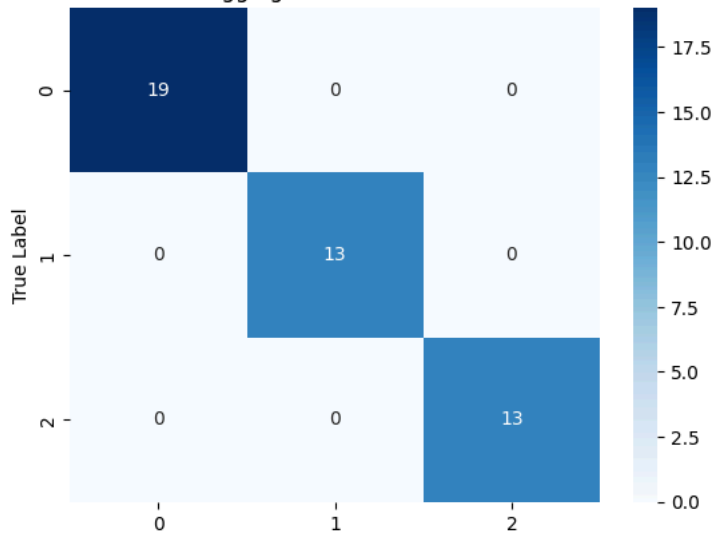
```

# Generate and plot confusion matrix for Stacking
plot_confusion_matrix(y_test, stacking_predictions, "Stacking Confusion Matrix")
print(classification_report(y_test, stacking_predictions))

```

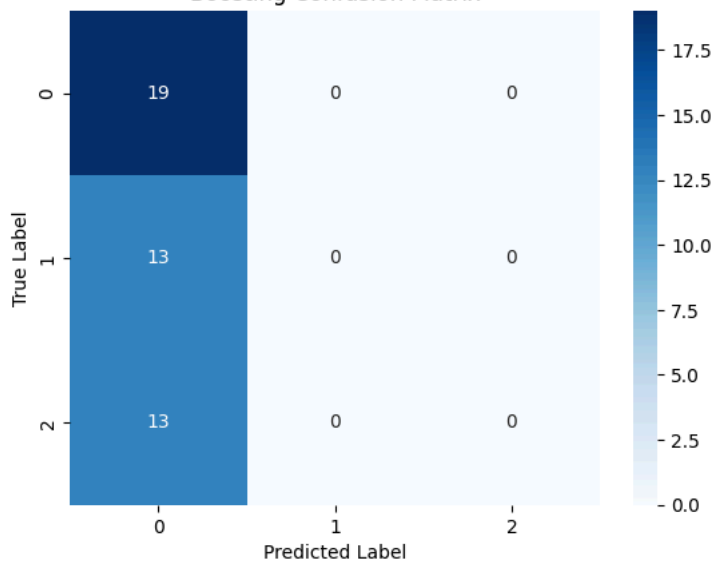


Bagging Confusion Matrix



	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

Boosting Confusion Matrix



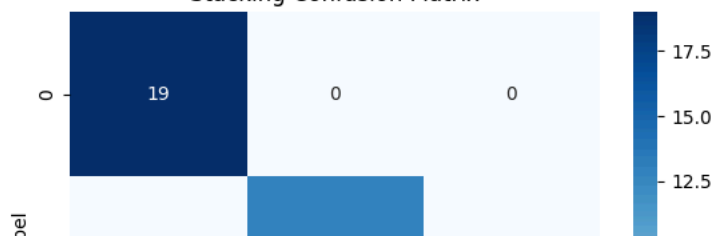
```

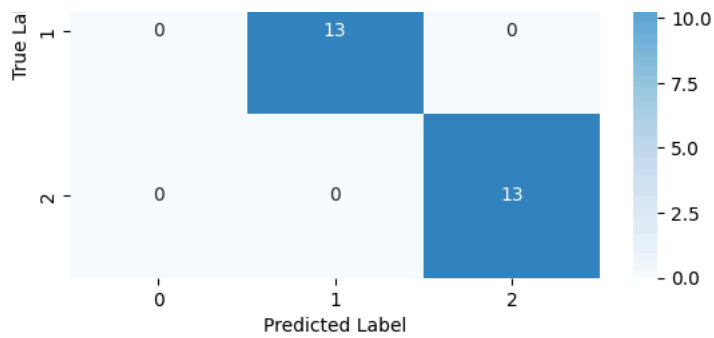
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score ar
_warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score ar
_warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score ar
_warn_prf(average, modifier, msg_start, len(result))

```

	precision	recall	f1-score	support
0	0.42	1.00	0.59	19
1	0.00	0.00	0.00	13
2	0.00	0.00	0.00	13
accuracy			0.42	45
macro avg	0.14	0.33	0.20	45
weighted avg	0.18	0.42	0.25	45

Stacking Confusion Matrix





	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	13
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

```
import pandas as pd
print("Bagging Predictions:", bagging_predictions)
print("Boosting Predictions:", boosting_predictions)
print("Stacking Predictions:", stacking_predictions)

# Compare predictions for each data point
comparison_df = pd.DataFrame({
    'y_test': y_test,
    'Bagging': bagging_predictions,
    'Boosting': boosting_predictions,
    'Stacking': stacking_predictions
})

print("\nComparison of Predictions:")
print(comparison_df)

# Calculate accuracy for each model
bagging_accuracy = accuracy_score(y_test, bagging_predictions)
boosting_accuracy = accuracy_score(y_test, boosting_predictions)
stacking_accuracy = accuracy_score(y_test, stacking_predictions)

print("\nAccuracy Scores:")
print(f"Bagging Accuracy: {bagging_accuracy:.2f}")
print(f"Boosting Accuracy: {boosting_accuracy:.2f}")
print(f"Stacking Accuracy: {stacking_accuracy:.2f}")
# Model names
model_names = ['Bagging', 'Boosting', 'Stacking']
accuracies = [bagging_accuracy, boosting_accuracy, stacking_accuracy]

# Create a bar plot
plt.figure(figsize=(8, 6))
plt.bar(model_names, accuracies, color=['blue', 'green', 'orange'])
plt.title('Comparison of Ensemble Methods on Iris Dataset')
plt.ylabel('Accuracy')
plt.xlabel('Ensemble Method')
plt.ylim(0.8, 1.0) # Adjust y-axis limits as needed
plt.show()
```



```

Bagging Predictions: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 0 0 0 1 0 0 2 1
0 0 0 2 1 1 0 0]
Boosting Predictions: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0]
Stacking Predictions: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 1 0 0 2 1
0 0 0 2 1 1 0 0]

```

Comparison of Predictions:

y_test	Bagging	Boosting	Stacking
0	1	1	0
1	0	0	0
2	2	2	0
3	1	1	0
4	1	1	0
5	0	0	0
6	1	1	0
7	2	2	0
8	1	1	0
9	1	1	0
10	2	2	0
11	0	0	0
12	0	0	0
13	0	0	0
14	0	0	0
15	1	1	0
16	2	2	0
17	1	1	0
18	1	1	0
19	2	2	0
20	0	0	0
21	2	2	0
22	0	0	0
23	2	2	0
24	2	2	0
25	2	2	0
26	2	2	0
27	2	2	0
28	0	0	0
29	0	0	0
30	0	0	0
31	0	0	0
32	1	1	0
33	0	0	0
34	0	0	0
35	2	2	0
36	1	1	0
37	0	0	0
38	0	0	0
39	0	0	0
40	2	2	0
41	1	1	0
42	1	1	0
43	0	0	0
44	0	0	0

Accuracy Scores:

Bagging Accuracy: 1.00

Boosting Accuracy: 0.42

Stacking Accuracy: 1.00

