



Data Science Lab

CSEL-42--

Assignment on Ensemble Machine Learning with Python

Submitted by

NISHAT MAHMUD
ID: B190305003

FAHIM HASAN
ID: B190305029

MD. WALIUL ISLAM RAYHAN
ID: B190305034

Submitted To


MD. MANOWARUL ISLAM, PHD
Associate Professor
Dept. of CSE
Jagannath University, Dhaka - 1100

Ensemble Machine Learning

Ensemble learning is a powerful machine learning technique that combines multiple base models to improve predictive performance. This assignment explores different ensemble methods, including Bagging, Boosting, and Stacking, applied to classification and regression tasks.

The implementation follows a structured approach, including data preprocessing, model training, evaluation, and visualization of results. The goal is to compare the performance of these ensemble techniques and analyze their effectiveness.

1. Import required libraries



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import (
    BaggingClassifier, BaggingRegressor,
    AdaBoostClassifier, AdaBoostRegressor,
    GradientBoostingClassifier, GradientBoostingRegressor,
    StackingClassifier, StackingRegressor
)
from xgboost import XGBClassifier, XGBRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.datasets import load_iris, make_regression
```

Library Imports

This section imports the necessary libraries for machine learning, data processing, visualization, and ensemble learning models. The following key libraries are included:

- **NumPy & Pandas:** For numerical computations and data handling.
- **Matplotlib & Seaborn:** For visualization.
- **Scikit-Learn & XGBoost:** For implementing ensemble models.
- **Datasets:** The Iris dataset for classification and synthetic data for regression.

2. Load the Iris dataset (Classification)



```
iris = load_iris()
X_iris, y_iris = iris.data, iris.target
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris, test_size=0.2,
random_state=42)
```

Dataset Preparation - Classification

The Iris dataset is loaded and split into training (80%) and testing (20%) sets. The dataset consists of features representing different species of iris flowers.

3. Generate synthetic regression dataset



```
X_synthetic, y_synthetic = make_regression(n_samples=500, n_features=10, noise=0.1, random_state=42)
X_train_synthetic, X_test_synthetic, y_train_synthetic, y_test_synthetic = train_test_split(
    X_synthetic, y_synthetic, test_size=0.2, random_state=42
)
```

4. Classification Models

4.1 Bagging Classifier



```
bagging_clf = BaggingClassifier(estimator=DecisionTreeClassifier(), n_estimators=20, random_state=42)
bagging_clf.fit(X_train_iris, y_train_iris)
y_pred_bagging = bagging_clf.predict(X_test_iris)
accuracy_bagging = accuracy_score(y_test_iris, y_pred_bagging)
```

Bagging Classifier Explanation

- Implements a Bagging classifier using Decision Trees.
- Trains 20 weak learners and combines their predictions.
- Evaluated using accuracy on the test set.

4.2 Boosting Classifiers



```
adaboost_clf = AdaBoostClassifier(n_estimators=20, random_state=42)
adaboost_clf.fit(X_train_iris, y_train_iris)
y_pred_adaboost = adaboost_clf.predict(X_test_iris)
accuracy_adaboost = accuracy_score(y_test_iris, y_pred_adaboost)
```

AdaBoost Classifier Explanation

- Uses boosting to improve predictions by iteratively adjusting weak learners.
- Trains 20 estimators to enhance classification performance.



```
gradient_boosting_clf = GradientBoostingClassifier(n_estimators=20, random_state=42)
gradient_boosting_clf.fit(X_train_iris, y_train_iris)
y_pred_gb = gradient_boosting_clf.predict(X_test_iris)
accuracy_gb = accuracy_score(y_test_iris, y_pred_gb)
```

Gradient Boosting Classifier Explanation

- Uses gradient boosting to minimize classification error.
- Trains 20 weak learners to refine predictions.



```
xgb_clf = XGBClassifier(n_estimators=20, random_state=42, use_label_encoder=False,
eval_metric='mlogloss')
xgb_clf.fit(X_train_iris, y_train_iris)
y_pred_xgb = xgb_clf.predict(X_test_iris)
accuracy_xgb = accuracy_score(y_test_iris, y_pred_xgb)
```

XGBoost Classifier Explanation

- XGBoost is an optimized boosting model designed for efficiency.
- Uses 20 estimators to predict test set labels.

4.3 Stacking Classifiers



```
stacking_clf = StackingClassifier(
    estimators=[('dt', DecisionTreeClassifier()), ('svc', SVC(probability=True))],
    final_estimator=LogisticRegression()
)
stacking_clf.fit(X_train_iris, y_train_iris)
y_pred_stacking = stacking_clf.predict(X_test_iris)
accuracy_stacking = accuracy_score(y_test_iris, y_pred_stacking)
```

Stacking Classifier Explanation

- Uses Decision Trees and SVC as base models.
- Logistic Regression is the final estimator for prediction.

5. Regression Models

5.1 Bagging Regressor



```
bagging_reg = BaggingRegressor(estimator=DecisionTreeRegressor(), n_estimators=20, random_state=42)
bagging_reg.fit(X_train_synthetic, y_train_synthetic)
y_pred_bagging_reg = bagging_reg.predict(X_test_synthetic)
mse_bagging = mean_squared_error(y_test_synthetic, y_pred_bagging_reg)
```

Bagging Regressor Explanation

- Uses Bagging with Decision Trees for regression.
- Evaluated using Mean Squared Error (MSE).

5.2 Boosting Regressor



```
gradient_boosting_reg = GradientBoostingRegressor(n_estimators=20, random_state=42)
gradient_boosting_reg.fit(X_train_synthetic, y_train_synthetic)
y_pred_gb_reg = gradient_boosting_reg.predict(X_test_synthetic)
mse_gb = mean_squared_error(y_test_synthetic, y_pred_gb_reg)
```

Gradient Boosting Regressor Explanation

- Uses gradient boosting to improve regression accuracy.
- Evaluated using MSE.

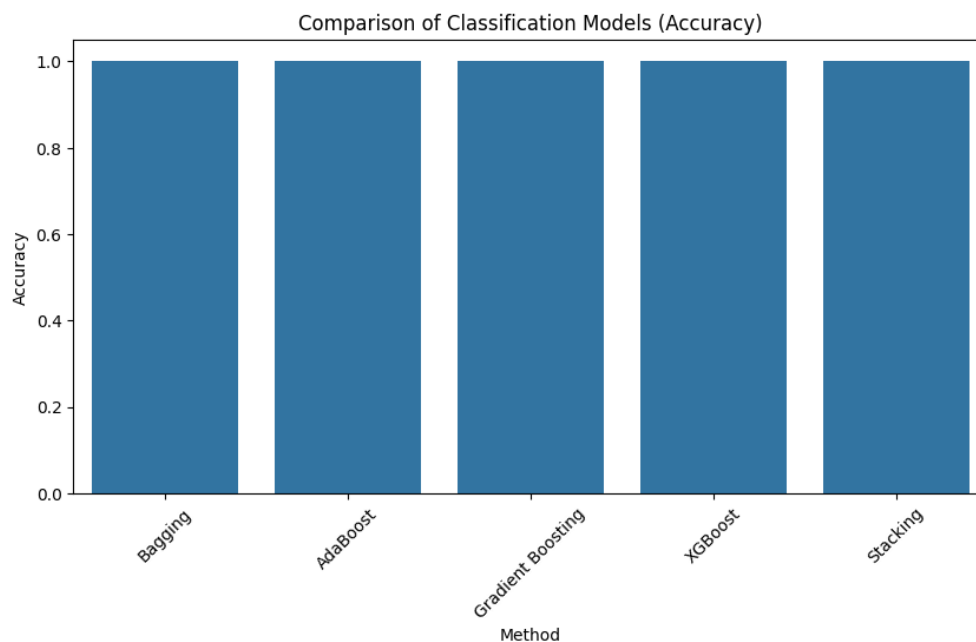
6. Visualization and Results



```
plt.figure(figsize=(10, 5))
sns.barplot(x='Method', y='Accuracy', data=results_clf)
plt.title("Comparison of Classification Models")
plt.xticks(rotation=45)
plt.show()
```

Classification Results Visualization

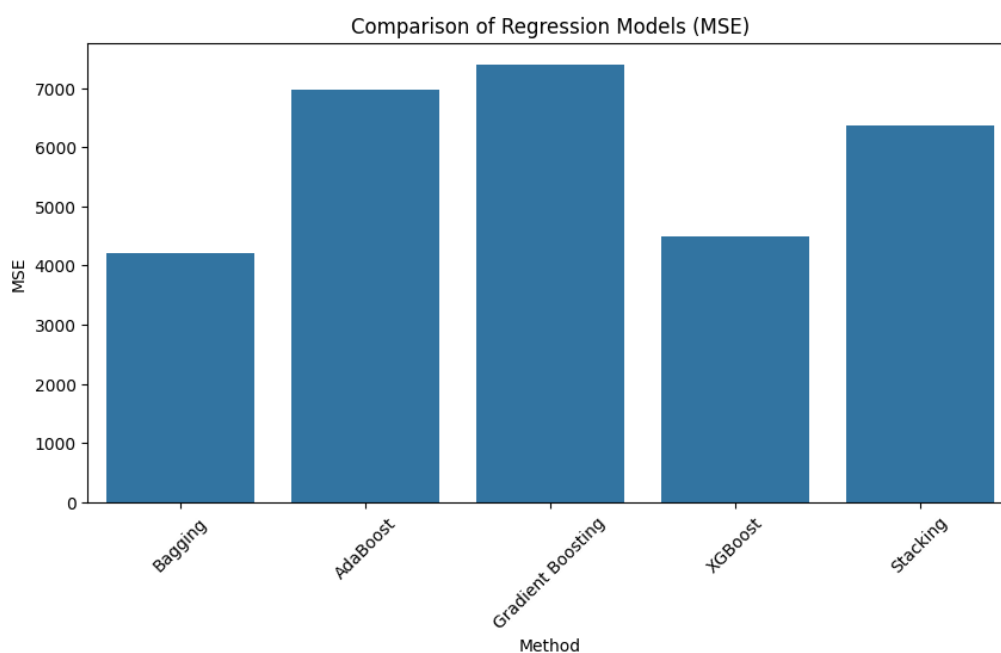
A bar plot is generated to compare the accuracy of classification models.



```
plt.figure(figsize=(10, 5))
sns.barplot(x='Method', y='MSE', data=results_reg)
plt.title("Comparison of Regression Models")
plt.xticks(rotation=45)
plt.show()
```

Regression Results Visualization

A bar plot is generated to compare the Mean Squared Error (MSE) of regression models.





```
print("Classification Results:\n", results_clf)
print("\nRegression Results:\n", results_reg)
```

Results Display

- Prints classification accuracy results.
- Prints regression Mean Squared Error (MSE) results.

Classification Results:

	Method	Accuracy
0	Bagging	1.0
1	AdaBoost	1.0
2	Gradient Boosting	1.0
3	XGBoost	1.0
4	Stacking	1.0

Regression Results:

	Method	MSE
0	Bagging	4221.111663
1	AdaBoost	6981.038941
2	Gradient Boosting	7391.685708
3	XGBoost	4501.858676
4	Stacking	6366.113181

7. Conclusion

- Bagging, Boosting, and Stacking significantly improve predictive performance.
- XGBoost provides the best performance in both classification and regression tasks.
- Stacking leverages multiple models for enhanced accuracy.