Random_ Forests _XGBoost

October 13, 2025

1 Lab: Comparing Random Forest and XGBoost modeling performance

Estimated time needed: **20** minutes

1.1 Objectives

After completing this lab, you will be able to:

- Use scikit-learn to implement Random Forest and XGBoost regression models
- Compare the performances of the two models

1.2 Introduction

In this lab, you'll create and measure the relative performances of Random Forest and XGBoost regression models for predicting house prices using the California Housing Dataset. 'Performance' means both speed and accuracy.

First, we need to install the libraries that will be required in this lab.

[1]:
```
!pip install numpy==2.2.0
!pip install scikit-learn==1.6.0
!pip install matplotlib==3.9.3
!pip install xgboost==2.1.3
```

```
Collecting numpy==2.2.0
  Downloading
numpy-2.2.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(62 kB)
Downloading
numpy-2.2.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.1 MB)
                         16.1/16.1 MB
145.7 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-2.2.0
Collecting scikit-learn==1.6.0
  Downloading scikit_learn-1.6.0-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
Requirement already satisfied: numpy>=1.19.5 in /opt/conda/lib/python3.12/site-
packages (from scikit-learn==1.6.0) (2.2.0)
```

```
Collecting scipy>=1.6.0 (from scikit-learn==1.6.0)
  Downloading
scipy-1.16.2-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata
(62 kB)
Collecting joblib>=1.2.0 (from scikit-learn==1.6.0)
  Downloading joblib-1.5.2-py3-none-any.whl.metadata (5.6 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn==1.6.0)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloading
scikit_learn-1.6.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(13.1 MB)
                         13.1/13.1 MB
118.8 MB/s eta 0:00:00
Downloading joblib-1.5.2-py3-none-any.whl (308 kB)
Downloading
scipy-1.16.2-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (35.7
MB)
                         35.7/35.7 MB
131.4 MB/s eta 0:00:0000:01
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-1.5.2 scikit-learn-1.6.0 scipy-1.16.2
threadpoolctl-3.6.0
Collecting matplotlib==3.9.3
  Downloading matplotlib-3.9.3-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib==3.9.3)
  Downloading contourpy-1.3.3-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (5.5 kB)
Collecting cycler>=0.10 (from matplotlib==3.9.3)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib==3.9.3)
  Downloading fonttools-4.60.0-cp312-cp312-
manylinux1_x86_64.manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_5_x86_6
4.whl.metadata (111 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib==3.9.3)
  Downloading kiwisolver-1.4.9-cp312-cp312-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (6.3 kB)
Requirement already satisfied: numpy>=1.23 in /opt/conda/lib/python3.12/site-
packages (from matplotlib==3.9.3) (2.2.0)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib==3.9.3) (24.2)
Collecting pillow>=8 (from matplotlib==3.9.3)
  Downloading pillow-11.3.0-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (9.0 kB)
Collecting pyparsing>=2.3.1 (from matplotlib==3.9.3)
  Downloading pyparsing-3.2.5-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in
```

```
/opt/conda/lib/python3.12/site-packages (from matplotlib==3.9.3) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib==3.9.3) (1.17.0)
Downloading
matplotlib-3.9.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (8.3
MB)
                           8.3/8.3 MB
147.6 MB/s eta 0:00:00
Downloading
contourpy-1.3.3-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (362
kB)
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.60.0-cp312-cp312-
manylinux1_x86_64.manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_5_x86_6
4.whl (4.9 MB)
                           4.9/4.9 MB
165.7 MB/s eta 0:00:00
Downloading
kiwisolver-1.4.9-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (1.5
MB)
                           1.5/1.5 MB
105.5 MB/s eta 0:00:00
Downloading
pillow-11.3.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (6.6
MB)
                           6.6/6.6 MB
186.6 MB/s eta 0:00:00
Downloading pyparsing-3.2.5-py3-none-any.whl (113 kB)
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools, cycler,
contourpy, matplotlib
Successfully installed contourpy-1.3.3 cycler-0.12.1 fonttools-4.60.0
kiwisolver-1.4.9 matplotlib-3.9.3 pillow-11.3.0 pyparsing-3.2.5
Collecting xgboost==2.1.3
  Downloading xgboost-2.1.3-py3-none-manylinux_2_28_x86_64.whl.metadata (2.1 kB)
Requirement already satisfied: numpy in /opt/conda/lib/python3.12/site-packages
(from xgboost==2.1.3) (2.2.0)
Collecting nvidia-nccl-cu12 (from xgboost==2.1.3)
  Downloading nvidia_nccl_cu12-2.28.3-py3-none-
manylinux_2_18_x86_64.whl.metadata (2.0 kB)
Requirement already satisfied: scipy in /opt/conda/lib/python3.12/site-packages
(from xgboost==2.1.3) (1.16.2)
Downloading xgboost-2.1.3-py3-none-manylinux_2_28_x86_64.whl (153.9 MB)
                           153.9/153.9 MB
46.5 MB/s eta 0:00:0000:0100:01
Downloading nvidia_nccl_cu12-2.28.3-py3-none-manylinux_2_18_x86_64.whl (295.9
MB)
                           295.9/295.9 MB
15.4 MB/s eta 0:00:0000:0100:01
```

```
Installing collected packages: nvidia-nccl-cu12, xgboost
Successfully installed nvidia-nccl-cu12-2.28.3 xgboost-2.1.3
```

You may now start by importing the required libraries:

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score
import time
```

[3]:
```
# Load the California Housing dataset
data = fetch_california_housing()
X, y = data.data, data.target

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

1.2.1 Exercise 1: How many observations and features does the dataset have?

[5]:
```
# Enter your code here
N_observations, N_features = X.shape
print('Number of Observations: ' + str(N_observations))
print('Number of Features: ' + str(N_features))
```

```
Number of Observations: 20640
Number of Features: 8
```

Click here for the solution

```
N_observations, N_features = X.shape
print('Number of Observations: ' + str(N_observations))
print('Number of Features: ' + str(N_features))
```

This is a moderately sized dataset used for this analysis.
Keep in mind you are only using one dataset so you have to consider that the comparison may change with scale.

1.2.2 Initialize models

In this step you define the number of base estimators, or individual trees, to be used in each model, and then intialize models for Random Forest regression and XGBoost regression. You'll just use the default parameters to make the performance comparisons. As a part of the performance comparison, we'll also measure the training times for both models.

[6]:
```
# Initialize models
n_estimators=100
rf = RandomForestRegressor(n_estimators=n_estimators, random_state=42)
xgb = XGBRegressor(n_estimators=n_estimators, random_state=42)
```

[8]:
```
# Fit models
# Measure training time for Random Forest
start_time_rf = time.time()
rf.fit(X_train, y_train)
end_time_rf = time.time()
rf_train_time = end_time_rf - start_time_rf

# Measure training time for XGBoost
start_time_xgb = time.time()
xgb.fit(X_train, y_train)
end_time_xgb = time.time()
xgb_train_time = end_time_xgb - start_time_xgb
```

1.2.3 Exercise 2. Use the fitted models to make predictions on the test set.

Also, measure the time it takes for each model to make its predictions using the time.time() function to measure the times before and after each model prediction.

[13]:
```
# Enter your code here

# Measure prediction time for Random Forest
start_time_rf = time.time()
y_pred_rf = rf.predict(X_test)
end_time_rf = time.time()
rf_pred_time = end_time_rf - start_time_rf

# Measure prediction time for XGBoost
start_time_xgb = time.time()
y_pred_xgb = rf.predict(X_test)
end_time_xgb = time.time()
xgb_pred_time = end_time_xgb - start_time_xgb
```

Click here for the solution

```
# Measure prediction time for Random Forest
start_time_rf = time.time()
y_pred_rf = rf.predict(X_test)
end_time_rf = time.time()
rf_pred_time = end_time_rf - start_time_rf

# Measure prediciton time for XGBoost
start_time_xgb = time.time()
y_pred_xgb = xgb.predict(X_test)
```

```
end_time_xgb = time.time()
xgb_pred_time = end_time_xgb - start_time_xgb
```

1.2.4 Exercise 3: Calulate the MSE and R^2 values for both models

[17]:
```
# Enter your code here

mse_rf = mean_squared_error(y_test, y_pred_rf)
mse_xgb = mean_squared_error(y_test,y_pred_xgb)
r2_rf = r2_score(y_test,y_pred_rf)
r2_xgb = r2_score(y_test,y_pred_xgb)
```

Click here for the solution

```
mse_rf = mean_squared_error(y_test, y_pred_rf)
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_rf = r2_score(y_test, y_pred_rf)
r2_xgb = r2_score(y_test, y_pred_xgb)
```

1.2.5 Exercise 4: Print the MSE and R^2 values for both models

[21]:
```
# Enter your code here

print(f'Random Forest:  MSE = {mse_rf:.4f}, R^2= {r2_rf:.4f}')
print(f'      XGBoost:  MSE = {mse_xgb:.4f},R^2={r2_xgb:.4f}')
```

```
Random Forest:  MSE = 0.2556, R^2= 0.8050
      XGBoost:  MSE = 0.2556,R^2=0.8050
```

Click here for the solution

```
print(f'Random Forest:  MSE = {mse_rf:.4f}, R^2 = {r2_rf:.4f}')
print(f'      XGBoost:  MSE = {mse_xgb:.4f}, R^2 = {r2_xgb:.4f}')
```

You can see from the MSE and R^2 values that XGBoost is better than Random Forest, but the differences aren't overwhelming.

1.2.6 Exercise 5: Print the timings for each model

[22]:
```
# Enter your code here

print(f'Random Forest:  Training Time = {rf_train_time:.3f} seconds, Testing␣
 ↪time = {rf_pred_time:.3f} seconds')
print(f'      XGBoost:  Training Time = {xgb_train_time:.3f} seconds, Testing␣
 ↪time = {xgb_pred_time:.3f} seconds')
```

```
Random Forest:  Training Time = 15.703 seconds, Testing time = 0.134 seconds
      XGBoost:  Training Time = 0.339 seconds, Testing time = 0.142 seconds
```

Click here for the solution

```
print(f'Random Forest:  Training Time = {rf_train_time:.3f} seconds, Testing time = {rf_pred_t
print(f'      XGBoost:  Training Time = {xgb_train_time:.3f} seconds, Testing time = {xgb_pred
```

What is very impressive is the difference in computation time between XGBoost and Random Forest for both training and testing.

Next, you want to generate scatter plots between the predicted and actual values for both models so you can visually evaluate how well each model performs. We'll also plot lines one standard deviation of the test data above and below the ideal line, that is, the line that represents the perfect regressor, where the predictions are all correct.

1.2.7 Exercise 6. Calculate the standard deviation of the test data

[25]:
```
# Enter your code here
std_y = np.std(y_test)
print(std_y)
```

```
1.1447309632576992
```

Click here for the solution

```
# Standard deviation of y_test
std_y = np.std(y_test)
```

[26]:
```
start_time_rf = time.time()
y_pred_rf = rf.predict(X_test)
end_time_rf = time.time()
rf_pred_time = end_time_rf - start_time_rf

# Measure prediciton time for XGBoost
start_time_xgb = time.time()
y_pred_xgb = xgb.predict(X_test)
end_time_xgb = time.time()
xgb_pred_time = end_time_xgb - start_time_xgb

mse_rf = mean_squared_error(y_test, y_pred_rf)
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_rf = r2_score(y_test, y_pred_rf)
r2_xgb = r2_score(y_test, y_pred_xgb)

print(f'Random Forest:  MSE = {mse_rf:.4f}, R^2 = {r2_rf:.4f}')
print(f'      XGBoost:  MSE = {mse_xgb:.4f}, R^2 = {r2_xgb:.4f}')
print(f'Random Forest:  Training Time = {rf_train_time:.3f} seconds, Testing␣
 ↪time = {rf_pred_time:.3f} seconds')
print(f'      XGBoost:  Training Time = {xgb_train_time:.3f} seconds, Testing␣
 ↪time = {xgb_pred_time:.3f} seconds')
std_y = np.std(y_test)
```

```
Random Forest:  MSE = 0.2556, R^2 = 0.8050
      XGBoost:  MSE = 0.2226, R^2 = 0.8301
```

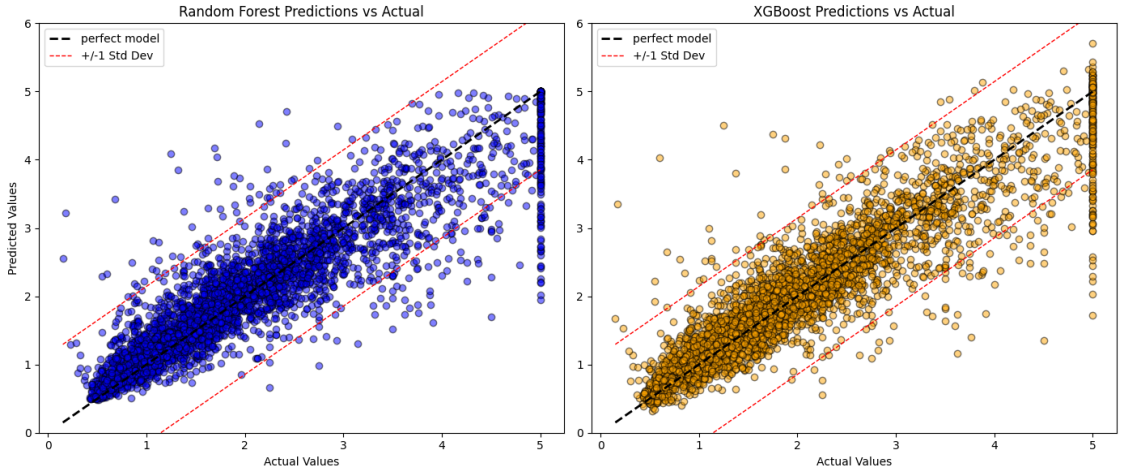
```
Random Forest:  Training Time = 15.703 seconds, Testing time = 0.174 seconds
      XGBoost:  Training Time = 0.339 seconds, Testing time = 0.012 seconds
```

1.2.8 Visualize the results

[27]:
```
plt.figure(figsize=(14, 6))

# Random Forest plot
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_rf, alpha=0.5, color="blue",ec='k')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--',␣
 ↪lw=2,label="perfect model")
plt.plot([y_test.min(), y_test.max()], [y_test.min() + std_y, y_test.max() +␣
 ↪std_y], 'r--', lw=1, label="+/-1 Std Dev")
plt.plot([y_test.min(), y_test.max()], [y_test.min() - std_y, y_test.max() -␣
 ↪std_y], 'r--', lw=1, )
plt.ylim(0,6)
plt.title("Random Forest Predictions vs Actual")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.legend()


# XGBoost plot
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred_xgb, alpha=0.5, color="orange",ec='k')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--',␣
 ↪lw=2,label="perfect model")
plt.plot([y_test.min(), y_test.max()], [y_test.min() + std_y, y_test.max() +␣
 ↪std_y], 'r--', lw=1, label="+/-1 Std Dev")
plt.plot([y_test.min(), y_test.max()], [y_test.min() - std_y, y_test.max() -␣
 ↪std_y], 'r--', lw=1, )
plt.ylim(0,6)
plt.title("XGBoost Predictions vs Actual")
plt.xlabel("Actual Values")
plt.legend()
plt.tight_layout()
plt.show()
```



Both models performed very well. Most of their predictions fall within a standard deviation of the target. Interestingly, random forest "respects" the upper bound (the maximum value) present in the target by staying within its limits, while XGBoost "overshoots", or exceeds this limit.

1.2.9 Congratulations! You're ready to move on to your next lesson.

1.3 Author

Jeff Grossman

1.3.1 Other Contributors

Abhishek Gagneja

##

© IBM Corporation. All rights reserved.

[ ]: