import pandas as pd

import numpy as np

import joblib

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.multioutput import MultiOutputRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.svm import SVR

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the dataset

filename = 'dataset\_combined\_small.xlsx' # Change this to your file path

data = pd.read\_excel(filename)

# Define the column numbers for features (x) and target features (t)

x\_columns = [0, 2, 3] # Column numbers for input features

t\_columns = [8, 9, 7] # Column numbers for target features

f = len(x\_columns) # Number of input features

o = len(t\_columns) # Number of target features

x = data.iloc[:, x\_columns] # Extract specified input features

t = data.iloc[:, t\_columns] # Extract specified target features

# Divide each column of y by its max absolute value

y\_max = np.max(np.abs(t), axis=0) # Calculate the maximum absolute value for each column

t /= y\_max # Divide each column by its max absolute value

# Convert column names to strings

x.columns = x.columns.astype(str)

t.columns = t.columns.astype(str)

# Split the data into training and testing sets

x\_train, x\_test, t\_train, t\_test = train\_test\_split(x, t, test\_size=0.25, random\_state=1)

# Define a list of base regressors

base\_regressors = [

LinearRegression(),

SVR(),

DecisionTreeRegressor(),

RandomForestRegressor(),

KNeighborsRegressor(n\_neighbors=5)

]

# Create a dictionary to store RMSE for each model

rmse\_dict = {}

r2\_dict = {}

# Train and evaluate each model

for regressor in base\_regressors:

model = MultiOutputRegressor(regressor)

model.fit(x\_train, t\_train)

y\_test = model.predict(x\_test)

rmse = np.sqrt(mean\_squared\_error(t\_test, y\_test, squared=False))

r2 = r2\_score(t\_test, y\_test)

model\_name = regressor.\_\_class\_\_.\_\_name\_\_

rmse\_dict[model\_name] = rmse

r2\_dict[model\_name] = r2

# Find the best-performing model

best\_model\_name = min(rmse\_dict, key=rmse\_dict.get)

# Create a new instance of the best-performing base regressor

best\_base\_regressor = [regressor for regressor in base\_regressors if regressor.\_\_class\_\_.\_\_name\_\_ == best\_model\_name][0]

# Create a MultiOutputRegressor with the best base regressor

best\_model = MultiOutputRegressor(best\_base\_regressor)

# Train the best model on the full training data

best\_model.fit(x, t)

# Predict on the test data

y\_test\_pred = best\_model.predict(x\_test)

# Initialize lists and dictionaries to store MSE, R^2, and percentage error deviation values

mse\_values = {f'Target\_{i}': [] for i in range(o)}

r2\_values = {f'Target\_{i}': [] for i in range(o)}

# Inside the loop after evaluating the model, calculate MSE, R^2, and percentage error deviation for each target feature

for target\_index in range(o):

target\_name = f'Target\_{target\_index}'

# Calculate MSE for the test set

test\_mse = mean\_squared\_error(t\_test.iloc[:, target\_index], y\_test\_pred[:, target\_index])

# Calculate MSE for the entire dataset (all data)

all\_data\_mse = mean\_squared\_error(t.iloc[:, target\_index], best\_model.predict(x)[:, target\_index])

# Calculate R^2 for the test set

test\_r2 = r2\_score(t\_test.iloc[:, target\_index], y\_test\_pred[:, target\_index])

# Calculate R^2 for the entire dataset (all data)

all\_data\_r2 = r2\_score(t.iloc[:, target\_index], best\_model.predict(x)[:, target\_index])

# Append the results to the lists

mse\_values[target\_name].append({'Test': test\_mse, 'All Data': all\_data\_mse})

r2\_values[target\_name].append({'Test': test\_r2, 'All Data': all\_data\_r2})

# Print the MSE, R^2, and percentage error deviation values for each target feature separately

for target\_name in mse\_values.keys():

print(f'Target Feature: {target\_name}')

for dataset\_name in ['Test', 'All Data']:

print(f'{dataset\_name} MSE: {np.mean([mse\_values[target\_name][i][dataset\_name] for i in range(len(mse\_values[target\_name]))]):.5f}')

print(f'{dataset\_name} R^2: {np.mean([r2\_values[target\_name][i][dataset\_name] for i in range(len(r2\_values[target\_name]))]):.5f}')

print()

# Multiply each target feature separately by its corresponding maximum value

for i, target\_name in enumerate(range(o)):

y\_test\_pred[:, i] \*= y\_max[i]

t\_test.iloc[:, i] \*= y\_max[i]

# Calculate the percentage error deviation for each target feature on the test data

percentage\_error\_deviation = np.abs((t\_test - y\_test\_pred) / t\_test) \* 100

# Create a scatter plot with different colors for each target

plt.figure(figsize=(8, 6))

for i in range(o):

plt.scatter(range(len(percentage\_error\_deviation)), percentage\_error\_deviation.iloc[:, i], s=30, alpha=0.6, label=f'Target {i+1}')

plt.xlabel('Data Point')

plt.ylabel('Percentage Error Deviation (%)')

plt.title('Percentage Error Deviation for Each Target Feature')

plt.legend()

plt.grid(True)

plt.show()

# Create a dictionary to store data for DataFrame creation

data\_dict = {f'Actual\_{target\_name}': t\_test.iloc[:, i].values for i, target\_name in enumerate(range(o))}

data\_dict.update({f'Predicted\_{target\_name}': y\_test\_pred[:, i] for i, target\_name in enumerate(range(o))})

# Convert the feature DataFrame to a dictionary

feature\_data\_dict = {f'Feature\_{i+1}': x\_test.iloc[:, i].values for i in range(f)}

# Add the feature data to the main data dictionary

data\_dict.update(feature\_data\_dict)

# Create a DataFrame from the data dictionary

actual\_predicted\_df = pd.DataFrame(data\_dict)

# Save the DataFrame to an Excel file

actual\_predicted\_df.to\_excel('Actual\_Predicted\_Test\_Data.xlsx', index=False)

# Predict on the entire dataset

y\_pred = best\_model.predict(x)

# Multiply each target feature separately by its corresponding maximum value

for i, target\_name in enumerate(range(o)):

y\_pred[:, i] \*= y\_max[i]

t.iloc[:, i] \*= y\_max[i]

# Calculate the percentage error deviation for each target feature on all data

percentage\_error\_deviation\_all = np.abs((t - y\_pred) / t) \* 100

# Create a dictionary to store data for DataFrame creation

data\_dict\_all = {f'Actual\_{target\_name}': t.iloc[:, i].values for i, target\_name in enumerate(range(o))}

data\_dict\_all.update({f'Predicted\_{target\_name}': y\_pred[:, i] for i, target\_name in enumerate(range(o))})

# Convert the feature DataFrame to a dictionary

feature\_data\_dict\_all = {f'Feature\_{i+1}': x.iloc[:, i].values for i in range(f)}

# Add the feature data to the main data dictionary

data\_dict\_all.update(feature\_data\_dict\_all)

# Add new columns for percentage error deviation to the DataFrame

for i, target\_name in enumerate(range(o)):

data\_dict\_all[f'%Error\_Deviation\_{target\_name}'] = percentage\_error\_deviation\_all.iloc[:, i]

# Create a DataFrame from the data dictionary

actual\_predicted\_df\_all = pd.DataFrame(data\_dict\_all)

# Save the DataFrame to an Excel file

actual\_predicted\_df\_all.to\_excel('Actual\_Predicted\_All\_Data.xlsx', index=False)

# Save the best model using joblib

best\_model\_filename = 'best\_multioutput\_regressor.pkl'

joblib.dump(best\_model, best\_model\_filename)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import joblib

# Load the dataset (assuming 'data' is your original dataset)

# filename = 'dataset\_pr\_constrained\_norm\_all\_data.csv' # Change this to your file path

data = pd.read\_excel(filename)

# Load the maximum absolute values for scaling

y\_max = np.max(np.abs(data.iloc[:, t\_columns].values), axis=0)

# Load the saved best model

loaded\_model = joblib.load('best\_multioutput\_regressor.pkl')

# Initialize a dictionary to store maximum %error deviation for each target feature

max\_error = {f'Max\_error\_for\_Target\_{i}': [] for i in range(o)}

# Number of samples taken

n = 10

# Perform the experiment n times

for experiment in range(1, n):

# Randomly select 100 samples from the dataset

sampled\_data = data.sample(n=408, random\_state=experiment)

x\_samples = sampled\_data.iloc[:, x\_columns]

t\_samples = sampled\_data.iloc[:, t\_columns]

x\_samples.columns = x\_samples.columns.astype(str)

# Predict using the loaded model

y\_pred\_samples = loaded\_model.predict(x\_samples)

y\_pred\_samples \*= y\_max

# Calculate percentage error deviation for each target feature

percentage\_error\_deviation = np.abs((t\_samples - y\_pred\_samples) / t\_samples) \* 100

# Find and store maximum %error deviation for each target feature

for i in range(o):

max\_error[f'Max\_error\_for\_Target\_{i}'].append(np.max(percentage\_error\_deviation.iloc[:, i]))

print(np.max(percentage\_error\_deviation.iloc[:, i]))

# Print information about the best model

print(f"Best Model: {best\_model\_name}")

print("Best Model Parameters:")

print(best\_base\_regressor)

# Plot the results

plt.figure(figsize=(10, 6))

for i in range(o):

plt.scatter(range(1, n), max\_error[f'Max\_error\_for\_Target\_{i}'], marker='o', label=f'Target {i+1}')

plt.xlabel('Experiment')

plt.ylabel('Maximum Error Deviation (%)')

# Set the x-axis ticks at 20 equally spaced points

x\_ticks = np.linspace(0, n-1, 20, dtype=int)

plt.xticks(x\_ticks)

plt.legend()

plt.show()

Target Feature: Target\_0

Test MSE: 0.0000160

Test R^2: 0.9986244

All Data MSE: 0.0000166

All Data R^2: 0.9985051

Target Feature: Target\_1

Test MSE: 0.0000031

Test R^2: 0.9987594

All Data MSE: 0.0000043

All Data R^2: 0.9980891

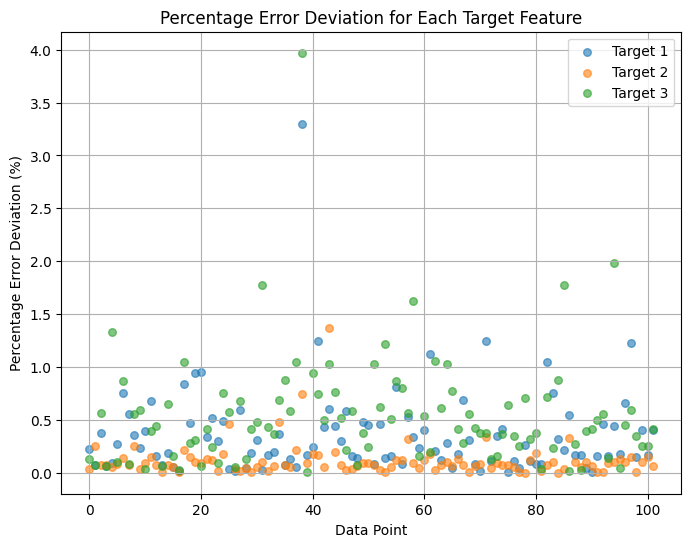
Target Feature: Target\_2

Test MSE: 0.0000209

Test R^2: 0.9993293

All Data MSE: 0.0000289

All Data R^2: 0.9990726



<ipython-input-99-4600a4f80e5d>:152: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation:<https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy>

t.iloc[:, i] \*= y\_max[i]

<ipython-input-99-4600a4f80e5d>:152: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation:<https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy>

t.iloc[:, i] \*= y\_max[i]

<ipython-input-99-4600a4f80e5d>:152: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation:<https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy>

t.iloc[:, i] \*= y\_max[i]

3.297740941731696

1.7064712879806248

4.797118847538699

3.297740941731696

1.7064712879806248

4.797118847538699

3.297740941731696

1.7064712879806248

4.797118847538699

3.297740941731696

1.7064712879806248

4.797118847538699

3.297740941731696

1.7064712879806248

4.797118847538699

3.297740941731696

1.7064712879806248

4.797118847538699

3.297740941731696

1.7064712879806248

4.797118847538699

3.297740941731696

1.7064712879806248

4.797118847538699

3.297740941731696

1.7064712879806248

4.797118847538699

Best Model: RandomForestRegressor

Best Model Parameters:

RandomForestRegressor()

