Linear_Regressor

May 15, 2024

1 This script hold necessary `classes` which implements the `Linear Regression` algorithm with `Backpropagation`

[28]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

2 Associated Formulae for the `LinearRegression` Class

Hypothesis function:

$$h_\theta(X) = \theta^T X + b = \theta_0 + \theta_1 \cdot X_1 + \theta_2 \cdot X_2 + \theta_3 \cdot X_3 + \theta_4 \cdot X_4 \tag{1}$$

Mean Squared Error (MSE) Loss function:

$$J(\theta, b) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(X^{(i)}) - y^{(i)})^2 \tag{2}$$

Gradient of the Loss function with respect to weights (θ) and bias (b):

$$\frac{\partial J}{\partial \theta} = \frac{1}{m} X^T (h_\theta(X) - y) \tag{3}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(X^{(i)}) - y^{(i)}) \tag{4}$$

Parameter Update (Gradient Descent):

$$\theta := \theta - \alpha \frac{\partial J}{\partial \theta} \tag{5}$$

$$b := b - \alpha \frac{\partial J}{\partial b} \tag{6}$$

Prediction:

$$y_{\text{pred}} = h_\theta(X) \tag{7}$$

Normalization of Features:

$$X_{\text{normalized}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{8}$$

Here, - $\theta_0 = b$ is the bias term, - θ_1 corresponds to the weight for Power Demand, X_1, - θ_2 corresponds to the weight for Month, X_2, - θ_3 corresponds to the weight for Day of the Week, X_3, - θ_4 corresponds to the weight for Time of the Day, X_4, - m is the number of training examples.

3 The class for `Linear Regression`

[29]:
```
class LinearRegression:
    def __init__(self, learning_rate=0.01, n_iterations=1000):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iterations):
            # Predictions
            y_pred = np.dot(X, self.weights) + self.bias

            # Compute gradients
            dw = (1 / n_samples) * np.dot(X.T, (y_pred - y))
            db = (1 / n_samples) * np.sum(y_pred - y)

            # Update weights and bias
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        return np.dot(X, self.weights) + self.bias
```

[30]:
```
csv_file = 'simulated_voltage_unbalance_data.csv'
```

[31]:
```
data = pd.read_csv(csv_file)
```

4 Training Our `LinearRegressor` model using the `csv_file` data

[32]:
```
# X contains the features and y contains the target variable (voltage unbalance)
X = np.array(data[['Power Demand (MW)', 'Month', 'Day of the Week', 'Time of␣
↪the Day (Hour)']])
```

```
y = np.array(data['Voltage Unbalance'])

# Normalize features
X_normalized = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

# Add bias term
X_normalized = np.hstack((X_normalized, np.ones((X_normalized.shape[0], 1))))

# Initialize and train the model
model = LinearRegression(learning_rate=0.01, n_iterations=1000)
model.fit(X_normalized, y)

# Print learned weights
print("Learned weights:", model.weights)

# Predict
predictions = model.predict(X_normalized)
```

```
Learned weights: [0.03532779 0.01283355 0.03982089 0.10873562 0.04940442]
```

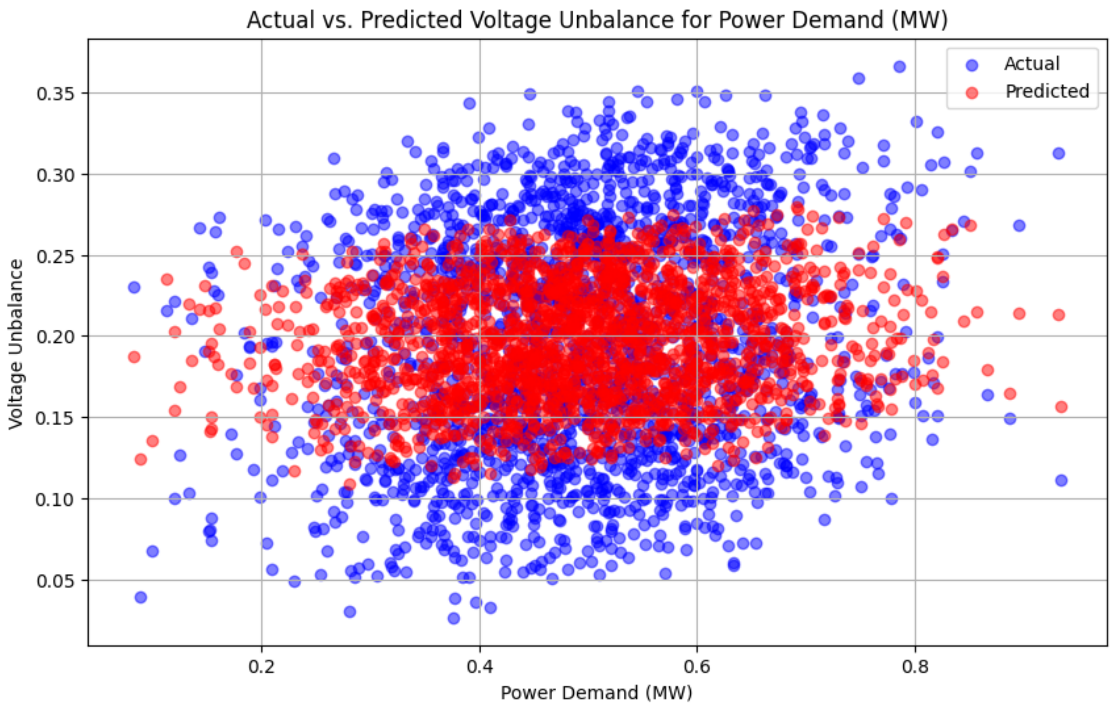
5 Model Evaluation

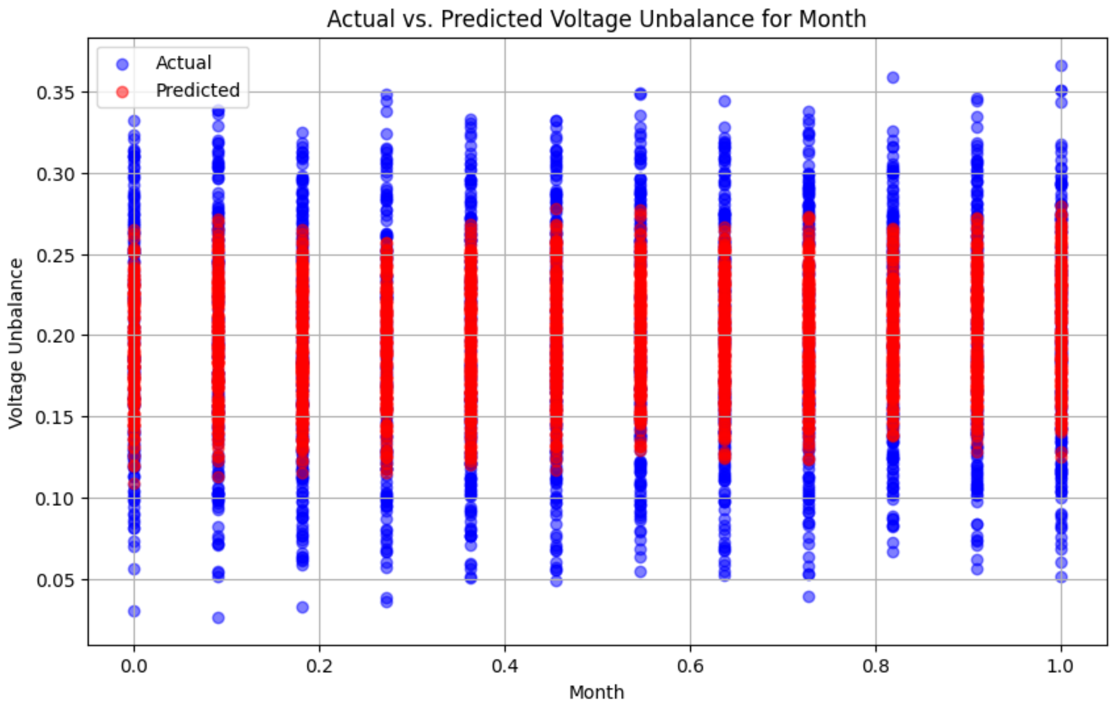
[33]:
```
# Split the data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y,␣
↪test_size=0.2, random_state=42)

# Initialize and train the model
model = LinearRegression(learning_rate=0.01, n_iterations=1000)
model.fit(X_train, y_train)

# Predict on the training and test sets
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)

# Calculate MSE on training and test sets
train_mse = mean_squared_error(y_train, train_predictions)
test_mse = mean_squared_error(y_test, test_predictions)

print("Train MSE:", train_mse)
print("Test MSE:", test_mse)
```

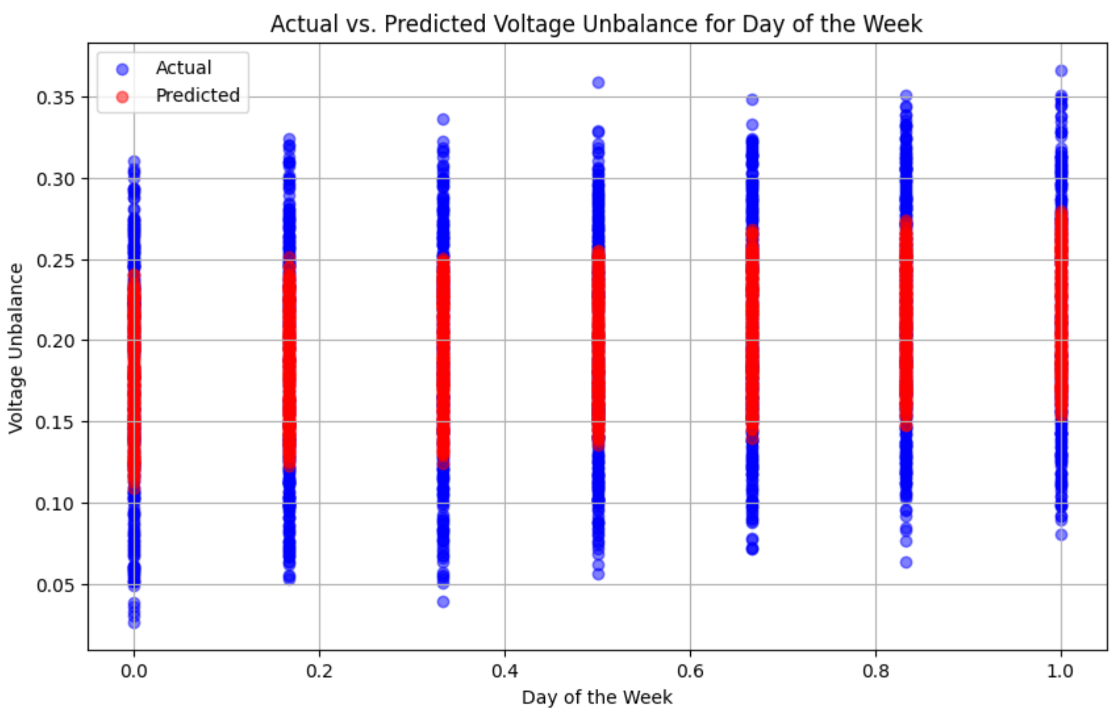
```
Train MSE: 0.0028690635696877642
Test MSE: 0.002726562557570657
```

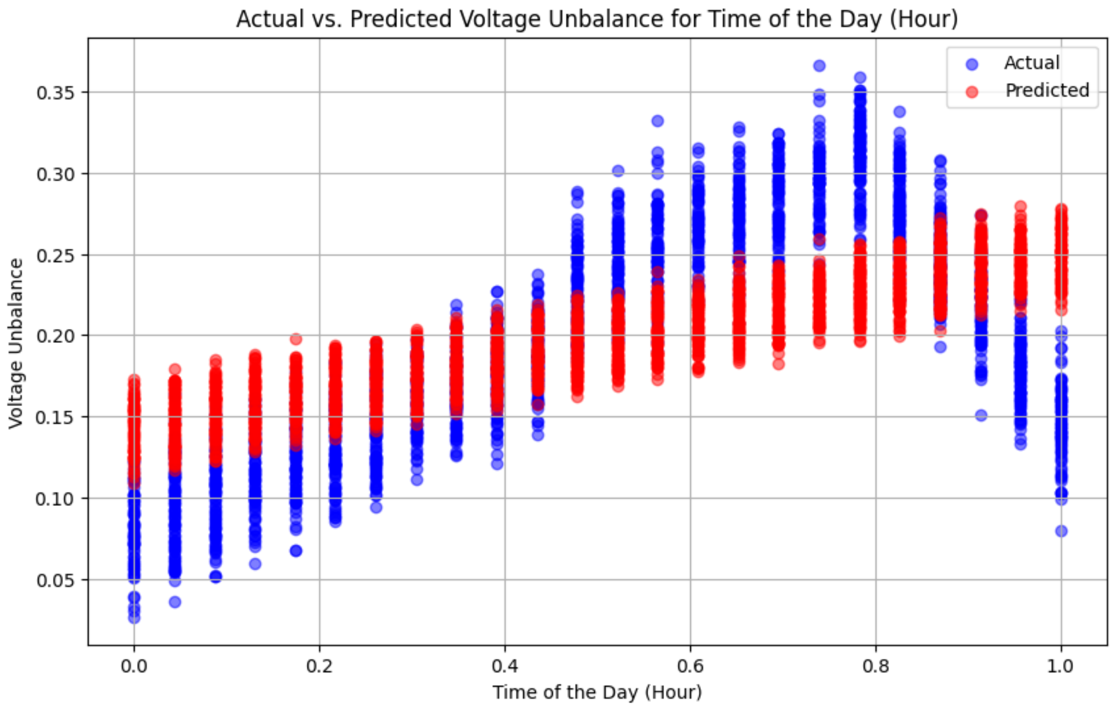
6 Visualization of our prediction against actual Measurements

[34]:
```
# Plot actual vs. predicted for each feature using the test set
for feature in features:
    plt.figure(figsize=(10, 6))

    # Plot actual vs. predicted
    plt.scatter(X_test[:, features.index(feature)], y_test, color='blue',␣
 ↪label='Actual', alpha=0.5)
    plt.scatter(X_test[:, features.index(feature)], test_predictions,␣
 ↪color='red', label='Predicted', alpha=0.5)

    plt.xlabel(feature)
    plt.ylabel('Voltage Unbalance')
    plt.title(f'Actual vs. Predicted Voltage Unbalance for {feature}')
    plt.legend()
    plt.grid(True)
    plt.show()
```




Actual vs. Predicted Voltage Unbalance for Month
Actual
Predicted
Voltage Unbalance
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.0
0.2
0.4
0.6
0.8
1.0
Month


Actual vs. Predicted Voltage Unbalance for Day of the Week
Actual
Predicted
Voltage Unbalance
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.0
0.2
0.4
0.6
0.8
1.0
Day of the Week


Actual vs. Predicted Voltage Unbalance for Time of the Day (Hour)
Actual
Predicted
0.35
0.30
0.25
0.20
0.15
0.10
0.05
Voltage Unbalance
0.0
0.2
0.4
0.6
0.8
1.0
Time of the Day (Hour)

[ ]: