LAB 2 - Regression - Yassine MKAOUAR

October 31, 2024

Name : Yassine MKAOUAR

E-mail : yassine.mkaouar@polytechnique.edu

1 MAP654I: Practical Introduction to Machine Learning

1.1 Practical Session 2: Regression

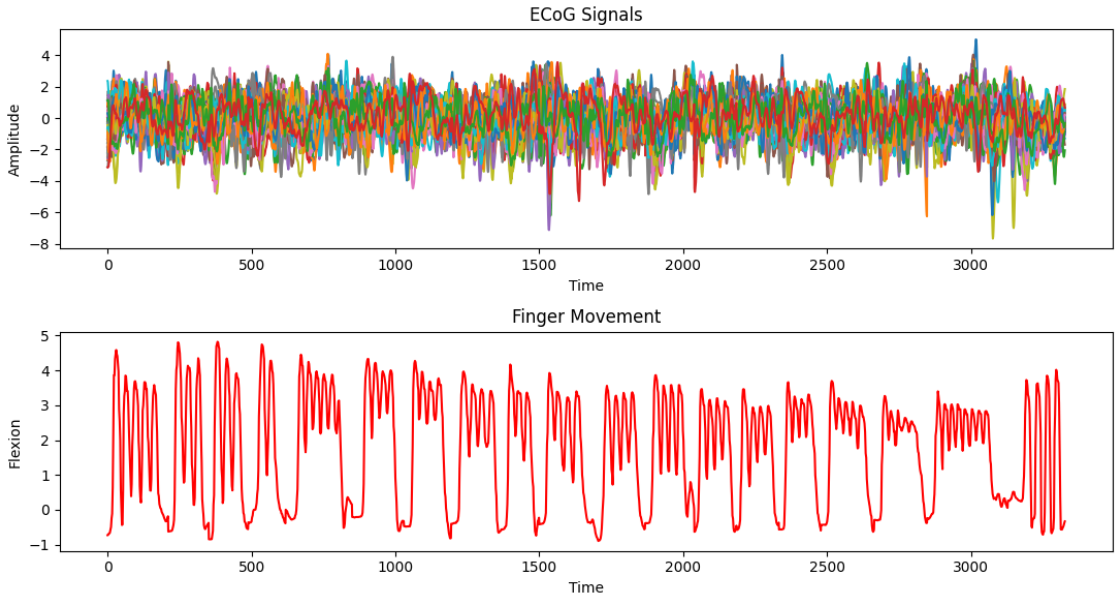
1.1.1 1-) Importing libraries

[42]:
```
import numpy as np
import pylab as pl
import scipy as sp
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
import pandas as pd
```

1.1.2 2-) Data visualization and pre-processing

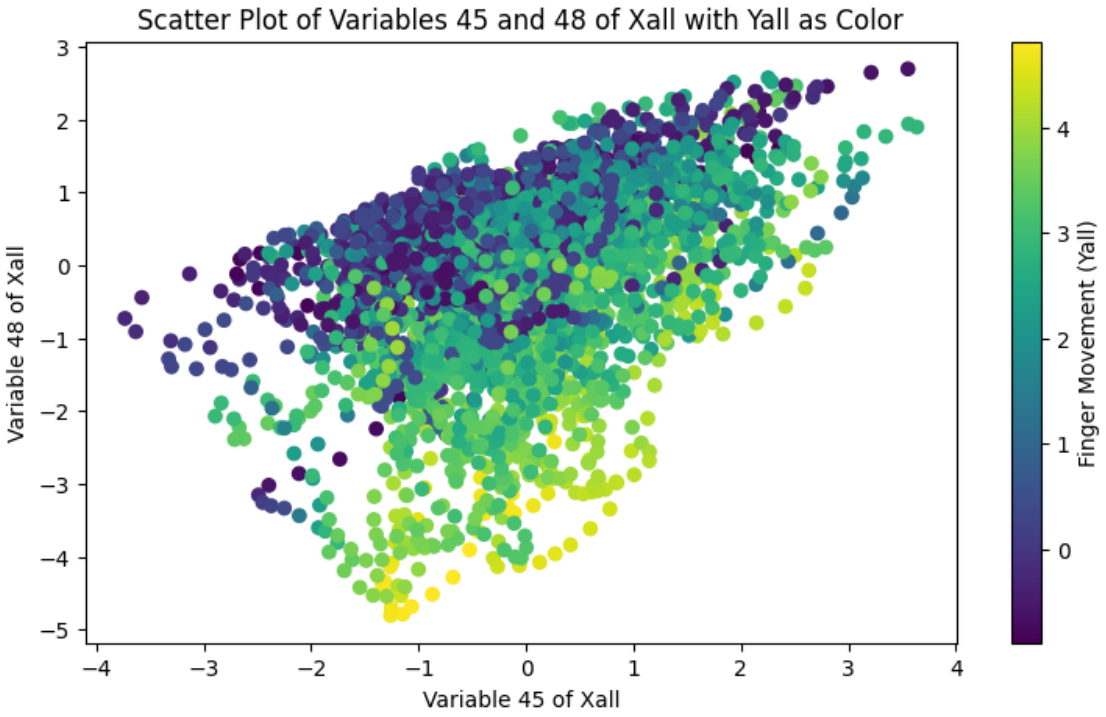
[ ]:
```
# Load the dataset
data = np.load('ECoG.npz')
Xall = data['Xall']
Yall = data['Yall']
Fe = data['Fe']

pl.figure(figsize=(11, 6))

# Plot ECoG signals
pl.subplot(2, 1, 1)
pl.plot(Xall)
pl.title('ECoG Signals')
pl.xlabel('Time')
pl.ylabel('Amplitude')

# Plot finger movement
```

```
pl.subplot(2, 1, 2)
pl.plot(Yall, color='r')
pl.title('Finger Movement')
pl.xlabel('Time')
pl.ylabel('Flexion')

pl.tight_layout()
pl.show()
```



The ECoG signals (top plot) show fluctuations typical of brain activity, with multiple overlapping channels. The finger flexion signal (bottom plot) has a distinct, rhythmic pattern, with peaks representing thumb flexion and valleys indicating relaxation. Visually, there may be moments where peaks in the ECoG signal align with flexion movements, hinting at a possible correlation.

[3]:
```
# Scatter plot with variables 45 and 48 of Xall and Yall controlling the color
plt.figure(figsize=(9, 5))
scatter = plt.scatter(Xall[:, 45], Xall[:, 48], c=Yall.flatten(),␣
 ↪cmap='viridis')
plt.colorbar(scatter, label='Finger Movement (Yall)')
plt.xlabel('Variable 45 of Xall')
plt.ylabel('Variable 48 of Xall')
plt.title('Scatter Plot of Variables 45 and 48 of Xall with Yall as Color')
plt.show()
```



The scatter plot reveals a positive correlation between ECoG variables 45 and 48, with color indicating finger flexion intensity (`Yall`). Higher flexion values (yellow points) are concentrated in specific regions, suggesting that certain combinations of these variables align with stronger finger movements. This pattern hints that these features may be predictive of finger flexion, as specific ECoG activity levels correspond to distinct movement intensities.

[4]:
```
n = 1000 # Number of training samples

# Split the data
X_train = Xall[:n, :]
X_test = Xall[n:, :]
Y_train = Yall[:n]
Y_test = Yall[n:]

print("Training set shapes:", X_train.shape, Y_train.shape)
print("Testing set shapes:", X_test.shape, Y_test.shape)
```

```
Training set shapes: (1000, 64) (1000, 1)
Testing set shapes: (2327, 64) (2327, 1)
```

1.1.3 3-) Least Squares regression (LS)

[ ]:
```
ones_column = np.ones((X_train.shape[0], 1))

# Concatenate the column of ones to the training samples
X_tilde = np.concatenate((ones_column, X_train), axis=1)

print("Shape of X_tilde:", X_tilde.shape)
```

```
Shape of X_tilde: (1000, 65)
```

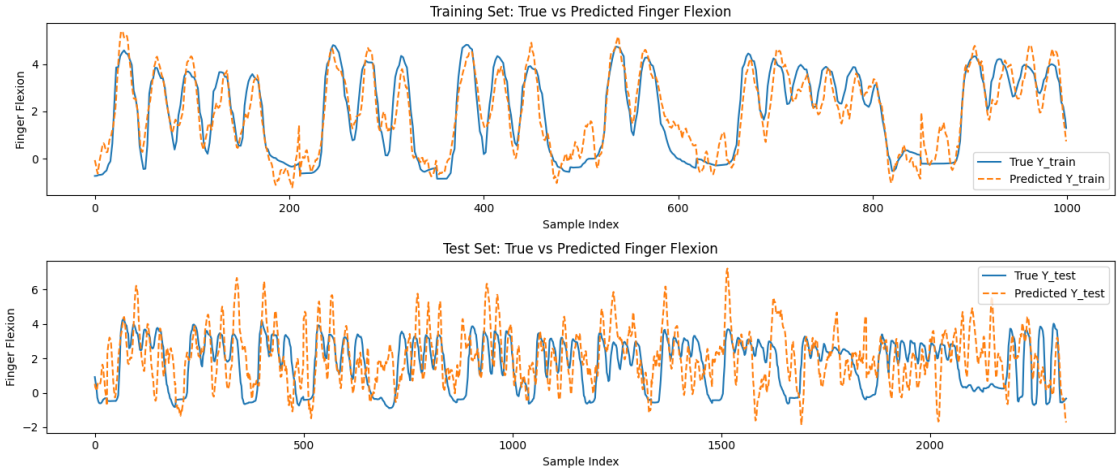
[ ]:
```
# Compute the LS parameters
XtX = np.dot(X_tilde.T, X_tilde)
XtY = np.dot(X_tilde.T, Y_train)
teta = np.linalg.solve(XtX, XtY) # Solve the linear system XtX * teta = XtY

b = teta[0]
w = teta[1:]

print("Bias (b):", b)
print("Weights (w):", w)
```

```
Bias (b): [1.90081255]
Weights (w): [[-0.09150694]
 [ 0.10861623]
 [-0.07305774]
 [-0.27938349]
 [ 0.25112061]
 [ 0.41213833]
 [ 0.14387528]
 [-0.19811799]
 [ 0.40869961]
 [ 0.7059866 ]
 [-0.4557358 ]
 [ 0.02140866]
 [ 0.35947183]
 [-0.01042572]
 [-0.17655887]
 [-0.03377368]
 [-0.11345198]
 [-0.72143135]
 [-0.5950915 ]
 [ 0.23329595]
 [-0.65132811]
 [-0.33317441]
 [ 0.28861537]
 [ 0.24498034]
 [ 0.22062027]
 [ 0.19042788]
```

```
[ 0.20709995]
[ 0.3273775 ]
[-0.65410592]
[ 0.11997444]
[-0.07491945]
[ 0.10667581]
[ 0.11003235]
[-0.42910159]
[-0.22329342]
[-0.00381723]
[-0.06046664]
[-0.1595164 ]
[-0.48519392]
[ 0.15019433]
[ 0.25855769]
[-0.09242374]
[ 0.76314124]
[-0.0834797 ]
[ 0.04489988]
[-0.16816255]
[ 0.04845934]
[ 0.0256902 ]
[-1.03905805]
[ 0.16060454]
[-0.19110549]
[ 0.16557271]
[ 0.08064689]
[ 0.27554638]
[-0.01162067]
[-0.00740264]
[ 0.33850545]
[ 0.22409642]
[-0.2196636 ]
[-0.26866884]
[-0.16469195]
[-0.23638421]
[ 0.04244268]
[ 0.5694358 ]]
```

[ ]:
```python
# Predict the finger flexion on the training set
Y_train_pred = np.dot(X_tilde, teta)

ones_column_test = np.ones((X_test.shape[0], 1))

X_test_tilde = np.concatenate((ones_column_test, X_test), axis=1)

Y_test_pred = np.dot(X_test_tilde, teta)
```

```
# Plot the true and predicted values for the training set
pl.figure(figsize=(14, 6))

pl.subplot(2, 1, 1)
pl.plot(Y_train, label='True Y_train')
pl.plot(Y_train_pred, label='Predicted Y_train', linestyle='--')
pl.title('Training Set: True vs Predicted Finger Flexion')
pl.xlabel('Sample Index')
pl.ylabel('Finger Flexion')
pl.legend()

# Plot the true and predicted values for the test set
pl.subplot(2, 1, 2)
pl.plot(Y_test, label='True Y_test')
pl.plot(Y_test_pred, label='Predicted Y_test', linestyle='--')
pl.title('Test Set: True vs Predicted Finger Flexion')
pl.xlabel('Sample Index')
pl.ylabel('Finger Flexion')
pl.legend()

pl.tight_layout()
pl.show()
```



In the top plot for the **training set**, the predicted finger flexion (orange dashed line) closely matches the true values (solid blue line), indicating that the least squares regression model fits well on the training data. This suggests the model has learned the underlying pattern of finger flexion during training.

However, in the bottom plot for the **test set**, the predicted values deviate more significantly from the true values, particularly in peak and valley regions. This discrepancy suggests the model may be *overfitting*, capturing noise specific to the training data rather than generalizing well to new,

unseen data. Improving the model's generalization could involve using regularization techniques or exploring alternative models that better capture the variability in finger flexion.

[ ]:
```
# Compute MSE and R2 for the training set
mse_train = mean_squared_error(Y_train, Y_train_pred)
r2_train = r2_score(Y_train, Y_train_pred)

# Test set
mse_test = mean_squared_error(Y_test, Y_test_pred)
r2_test = r2_score(Y_test, Y_test_pred)

print(f"Training MSE: {mse_train:.4f}")
print(f"Training R²: {r2_train:.4f}")
print(f"Test MSE: {mse_test:.4f}")
print(f"Test R²: {r2_test:.4f}")
```

```
Training MSE: 0.5041
Training R²: 0.8395
Test MSE: 3.0455
Test R²: -0.4282
```

The results indicate that the model performs well on the training set, with a mean squared error (MSE) of 0.5041 and an R^2 of 0.8395, suggesting a good fit to the training data. However, the performance on the test set raises concerns, as evidenced by a significantly higher MSE of 3.0455 and a negative R^2 value of -0.4282. This discrepancy suggests that the model is overfitting, capturing noise in the training data rather than generalizable trends.

A negative R^2 indicates that the model is performing worse than a simple model that predicts the mean of the target variable for all observations.

In fact, the coefficient of determination R^2, is derived from the sum of squares in regression analysis. Although it is commonly interpreted as a proportion of variance explained, it is not a "square" in the mathematical sense that would always yield a non-negative value. Here's how it can be negative:

$$R^2 = 1 - \frac{SS_{\text{residual}}}{SS_{\text{total}}}$$

Where:

- SS_{residual} is the sum of squares of residuals (the differences between the observed values and the predicted values).
- SS_{total} is the total sum of squares (the differences between the observed values and the mean of the observed values).

In summary, a negative R^2 value indicates that the model is failing to capture the data's underlying structure, performing worse than a naive approach that simply predicts the mean of the target variable. It serves as a signal to reassess model choice, complexity, or data quality.

[9]:
```
# Create and fit the linear regression model
model = LinearRegression(fit_intercept=True)
model.fit(X_train, Y_train)
```

```
# Get the estimated coefficients and intercept
sklearn_coef = model.coef_
sklearn_intercept = model.intercept_

print("Sklearn Coefficients:", sklearn_coef)
print("Sklearn Intercept:", sklearn_intercept)

# Check if the coefficients and intercept match
print("Coefficients match:", np.allclose(sklearn_coef, w.flatten())) # allclose␣
 ↪is a function that checks if two arrays are equal within a tolerance
print("Intercept matches:", np.allclose(sklearn_intercept, b))
```

```
Sklearn Coefficients: [[-0.09150694  0.10861623 -0.07305774 -0.27938349
0.25112061  0.41213833
   0.14387528 -0.19811799  0.40869961  0.7059866  -0.4557358   0.02140866
   0.35947183 -0.01042572 -0.17655887 -0.03377368 -0.11345198 -0.72143135
  -0.5950915   0.23329595 -0.65132811 -0.33317441  0.28861537  0.24498034
   0.22062027  0.19042788  0.20709995  0.3273775  -0.65410592  0.11997444
  -0.07491945  0.10667581  0.11003235 -0.42910159 -0.22329342 -0.00381723
  -0.06046664 -0.1595164  -0.48519392  0.15019433  0.25855769 -0.09242374
   0.76314124 -0.0834797   0.04489988 -0.16816255  0.04845934  0.0256902
  -1.03905805  0.16060454 -0.19110549  0.16557271  0.08064689  0.27554638
  -0.01162067 -0.00740264  0.33850545  0.22409642 -0.2196636  -0.26866884
  -0.16469195 -0.23638421  0.04244268  0.5694358 ]]
Sklearn Intercept: [1.90081255]
Coefficients match: True
Intercept matches: True
```

1.1.4 4-) Ridge regression

[ ]:
```
# Create and fit the ridge regression model with   = 1
ridge_model = Ridge(alpha=1)
ridge_model.fit(X_train, Y_train)

Y_train_pred_ridge = ridge_model.predict(X_train)

Y_test_pred_ridge = ridge_model.predict(X_test)

# Compute MSE and R2 for the training set
mse_train_ridge = mean_squared_error(Y_train, Y_train_pred_ridge)
r2_train_ridge = r2_score(Y_train, Y_train_pred_ridge)

# Compute MSE and R2 for the test set
mse_test_ridge = mean_squared_error(Y_test, Y_test_pred_ridge)
r2_test_ridge = r2_score(Y_test, Y_test_pred_ridge)
```

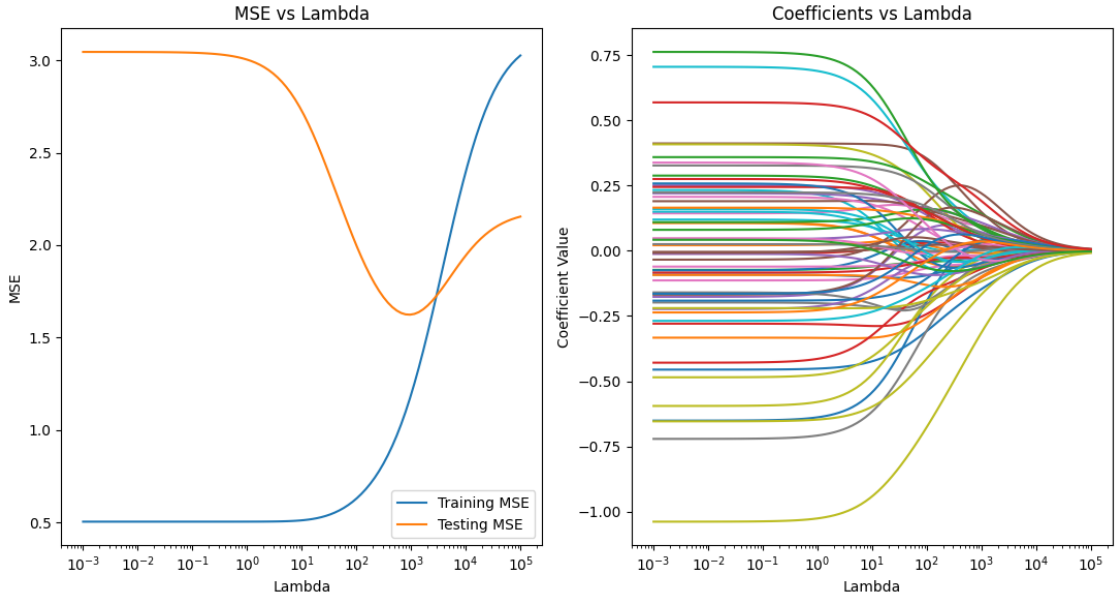
```
print(f"Ridge Training MSE: {mse_train_ridge:.4f}")
print(f"Ridge Training R²: {r2_train_ridge:.4f}")
print(f"Ridge Test MSE: {mse_test_ridge:.4f}")
print(f"Ridge Test R²: {r2_test_ridge:.4f}")
```

```
Ridge Training MSE: 0.5042
Ridge Training R²: 0.8395
Ridge Test MSE: 3.0038
Ridge Test R²: -0.4086
```

The Ridge regression results show a training MSE of 0.5042 and an R^2 of 0.8395, consistent with the previous model. However, the test MSE improves slightly to 3.0038, but the test R^2 remains negative at -0.4086, indicating poor performance. This suggests that while Ridge regression helps reduce overfitting slightly, it still struggles to generalize effectively to the test data.

[ ]:
```
# Define the range of lambda values
lambdas = np.logspace(-3, 5, 100)

# Initialize lists to store MSE values and coefficients
mse_train_ridge_list = []
mse_test_ridge_list = []
coefficients_list = []

# Loop over the range of lambda values
for alpha in lambdas:
    # Create and fit the ridge regression model
    ridge_model = Ridge(alpha=alpha)
    ridge_model.fit(X_train, Y_train)

    Y_train_pred_ridge = ridge_model.predict(X_train)

    Y_test_pred_ridge = ridge_model.predict(X_test)

    mse_train_ridge = mean_squared_error(Y_train, Y_train_pred_ridge)
    mse_train_ridge_list.append(mse_train_ridge)

    mse_test_ridge = mean_squared_error(Y_test, Y_test_pred_ridge)
    mse_test_ridge_list.append(mse_test_ridge)

    # Store the coefficients
    coefficients_list.append(ridge_model.coef_.flatten())

# Convert the list of coefficients to a numpy array for easier plotting
coefficients_array = np.array(coefficients_list)

# Plot the evolution of the MSE on training and testing data as a function of␣
 ↪lambda
plt.figure(figsize=(11, 6))
```

```
plt.subplot(1, 2, 1)
plt.plot(lambdas, mse_train_ridge_list, label='Training MSE')
plt.plot(lambdas, mse_test_ridge_list, label='Testing MSE')
plt.xscale('log')
plt.xlabel('Lambda')
plt.ylabel('MSE')
plt.title('MSE vs Lambda')
plt.legend()

# Plot the evolution of the linear parameters as a function of lambda
plt.subplot(1, 2, 2)
for i in range(coefficients_array.shape[1]):
    plt.plot(lambdas, coefficients_array[:, i], label=f'Coefficient {i+1}')
plt.xscale('log')
plt.xlabel('Lambda')
plt.ylabel('Coefficient Value')
plt.title('Coefficients vs Lambda')

plt.tight_layout()
plt.show()
```



Plot Observations : - MSE vs. Lambda:

The training MSE (blue line) typically increases as increases, which is expected because higher values lead to stronger regularization, reducing more and more the model complexity. The testing MSE (orange line) shows a U-shaped curve. Initially, it decreases with increasing , reaching a minimum, and then starts to increase. This minimum point usually indicates the optimal value

where the model generalizes best to unseen data.

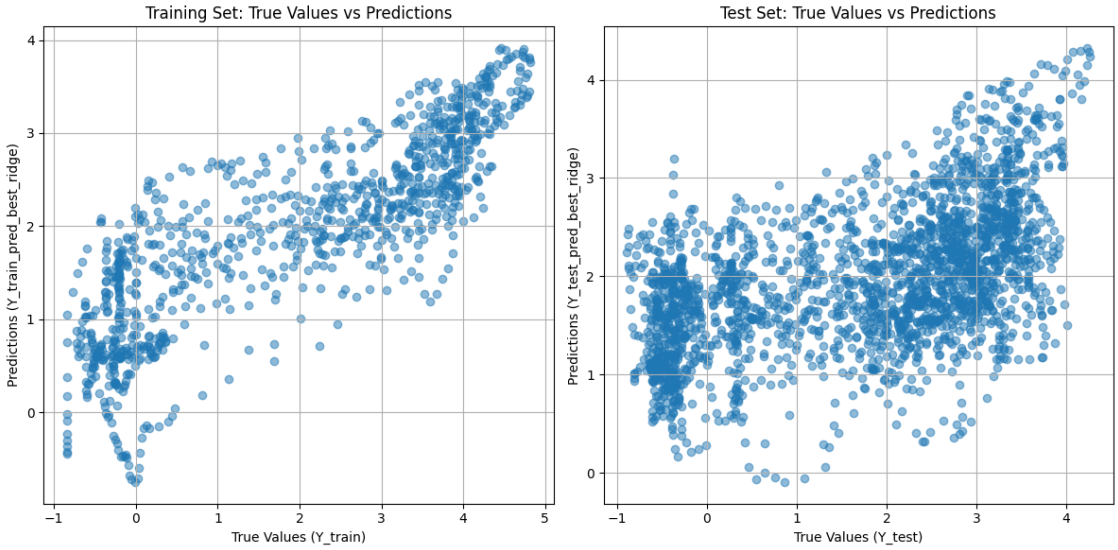
- Coefficients vs. Lambda:

The coefficients start at various values when is small and converge towards zero as increases. This is due to the stronger regularization effect, which shrinks the coefficients. Each line represents a different coefficient, showing how regularization impacts each feature's contribution to the model.

[ ]:
```
# Find the index of the minimum MSE in the test set
best_lambda_index = np.argmin(mse_test_ridge_list)

best_lambda = lambdas[best_lambda_index]

# Fit a Ridge model with the best lambda
best_ridge_model = Ridge(alpha=best_lambda)
best_ridge_model.fit(X_train, Y_train)

print(f"Best  : {best_lambda}")
```

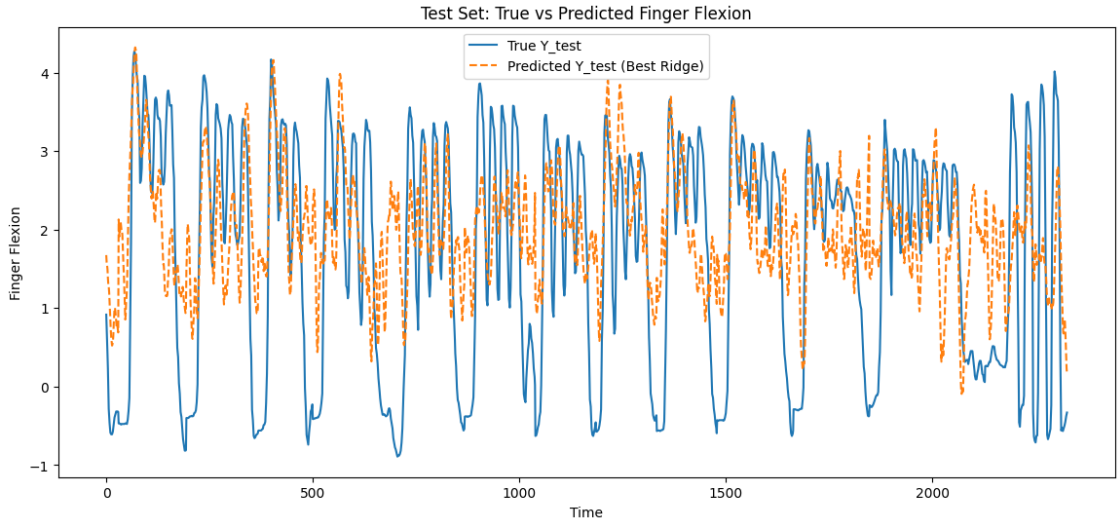
```
Best  : 954.5484566618347
```

[13]:
```
# Predict the finger flexion on the training set using the best Ridge model
Y_train_pred_best_ridge = best_ridge_model.predict(X_train)

# Predict the finger flexion on the test set using the best Ridge model
Y_test_pred_best_ridge = best_ridge_model.predict(X_test)

# Scatter plot for training set
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.scatter(Y_train, Y_train_pred_best_ridge, alpha=0.5)
plt.xlabel('True Values (Y_train)')
plt.ylabel('Predictions (Y_train_pred_best_ridge)')
plt.title('Training Set: True Values vs Predictions')
plt.grid(True)

# Scatter plot for test set
plt.subplot(1, 2, 2)
plt.scatter(Y_test, Y_test_pred_best_ridge, alpha=0.5)
plt.xlabel('True Values (Y_test)')
plt.ylabel('Predictions (Y_test_pred_best_ridge)')
plt.title('Test Set: True Values vs Predictions')
plt.grid(True)

plt.tight_layout()
plt.show()
```



Using the scatter plot, the perfect model is when the predicted values are equal to the true values. The closer the points are to the diagonal line, the better the model performs. The Ridge regression model shows a slight improvement over the least squares regression, but it still struggles to capture the underlying pattern of finger flexion in the test data.

[14]:
```
# Plot the true and predicted values for the test set as a function of time
plt.figure(figsize=(14, 6))

plt.plot(Y_test, label='True Y_test')
plt.plot(Y_test_pred_best_ridge, label='Predicted Y_test (Best Ridge)',␣
 ↪linestyle='--')

plt.title('Test Set: True vs Predicted Finger Flexion')
plt.xlabel('Time')
plt.ylabel('Finger Flexion')
plt.legend()
plt.show()
```



The prediction on the test data shows some improvement with the Ridge regression model compared to the least squares regression model. However, it still struggles to capture the underlying pattern of finger flexion, as evidenced by the deviation of predicted values from the true values in peak and valley regions.

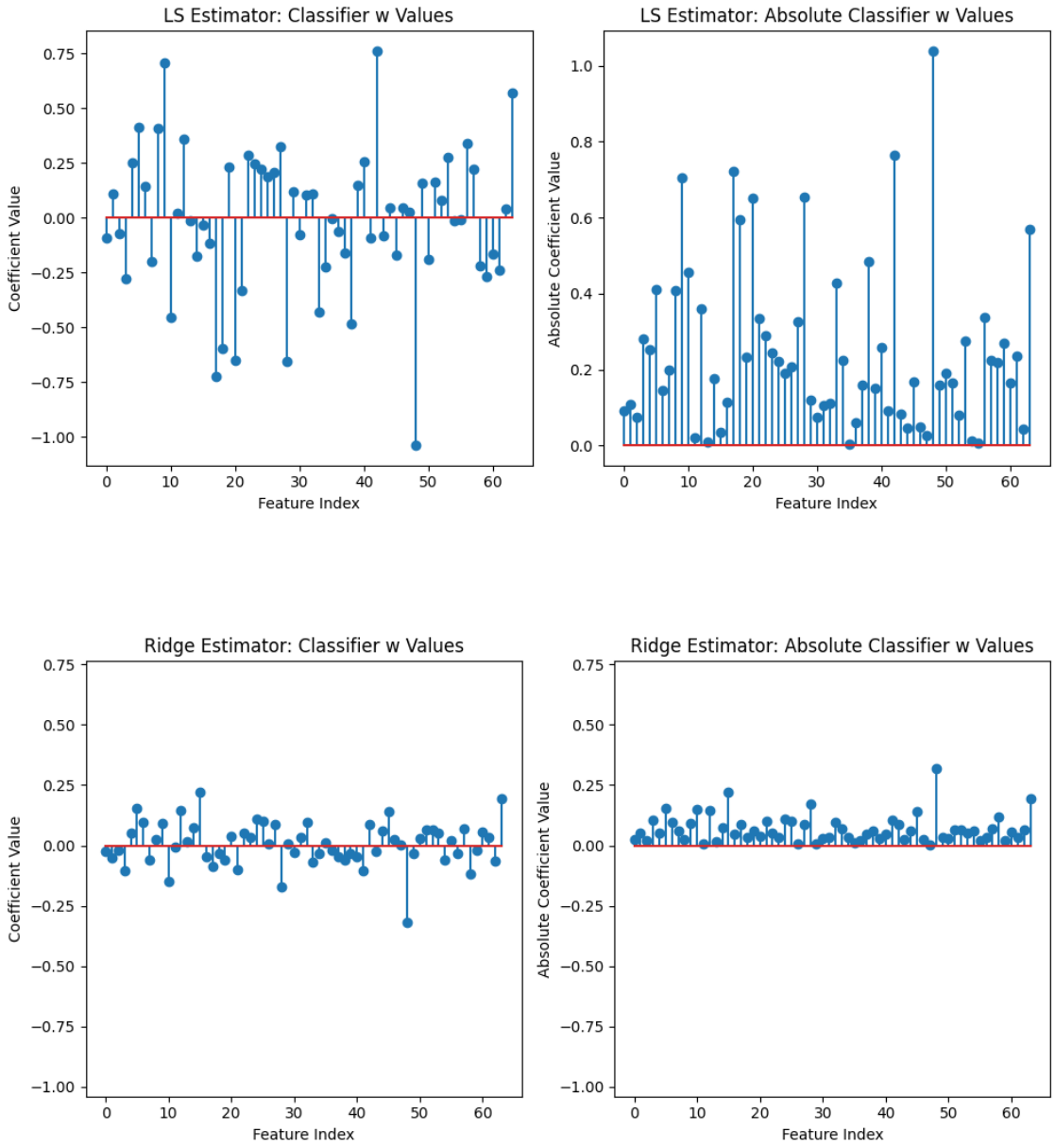
[15]:
```
# Predict the finger flexion on the test set using the best Ridge model
Y_test_pred_best_ridge = best_ridge_model.predict(X_test)

# Compute MSE and R2 for the test set using the best Ridge model
mse_test_best_ridge = mean_squared_error(Y_test, Y_test_pred_best_ridge)
r2_test_best_ridge = r2_score(Y_test, Y_test_pred_best_ridge)

print(f"Best Ridge Test MSE: {mse_test_best_ridge:.4f}")
print(f"Best Ridge Test R²: {r2_test_best_ridge:.4f}")
```

```
Best Ridge Test MSE: 1.6239
Best Ridge Test R²: 0.2385
```

Comparing the performance of the LS estimator and the Ridge estimator on the test set, we observe that the Ridge estimator has a lower MSE (1.6239 vs 3.0455) and a higher R^2 (0.2385 vs -0.4282). This indicates that the Ridge estimator generalizes better to the test data, capturing more of the underlying structure of the finger flexion signal. The regularization provided by the Ridge estimator helps prevent overfitting, leading to improved performance on unseen data.

[16]:
```
# Plot the values of the classifiers w for LS and Ridge estimators
plt.figure(figsize=(10, 5))

# Plot the values of w for LS estimator
plt.subplot(1, 2, 1)
plt.stem(w.flatten())
```

```
plt.title('LS Estimator: Classifier w Values')
plt.xlabel('Feature Index')
plt.ylabel('Coefficient Value')

# Plot the absolute values of w for LS estimator
plt.subplot(1, 2, 2)
plt.stem(np.abs(w.flatten()))
plt.title('LS Estimator: Absolute Classifier w Values')
plt.xlabel('Feature Index')
plt.ylabel('Absolute Coefficient Value')

plt.tight_layout()
plt.show()

# Plot the values of the classifiers w for Ridge estimator
plt.figure(figsize=(10, 5))

# Plot the values of w for Ridge estimator
plt.subplot(1, 2, 1)
plt.stem(best_ridge_model.coef_.flatten())
plt.title('Ridge Estimator: Classifier w Values')
plt.xlabel('Feature Index')
plt.ylabel('Coefficient Value')

# Plot the absolute values of w for Ridge estimator
plt.subplot(1, 2, 2)
plt.stem(np.abs(best_ridge_model.coef_.flatten()))
plt.title('Ridge Estimator: Absolute Classifier w Values')
plt.xlabel('Feature Index')
plt.ylabel('Absolute Coefficient Value')

# Set the same y-axis limits for both subplots
y_min = min(np.min(w.flatten()), np.min(best_ridge_model.coef_.flatten()))
y_max = max(np.max(w.flatten()), np.max(best_ridge_model.coef_.flatten()))
plt.subplot(1, 2, 1)
plt.ylim(y_min, y_max)
plt.subplot(1, 2, 2)
plt.ylim(y_min, y_max)

plt.tight_layout()
plt.show()
```



The LS and Ridge estimators show the same important variables (Large Magnitude), this suggests that we can perform feature selection to reduce the model complexity and improve generalization. By selecting the most relevant features, we can build a simpler model that captures the essential information in the data, leading to better performance on unseen data.

1.1.5 5-) Variable selection with the Lasso

[ ]:
```
# Create and fit the Lasso regression model with default lambda = 1
lasso_model = Lasso()
lasso_model.fit(X_train, Y_train)
```

```
lasso_coef = lasso_model.coef_
lasso_intercept = lasso_model.intercept_

print("Lasso Coefficients:", lasso_coef)
print("Lasso Intercept:", lasso_intercept)
```

```
Lasso Coefficients: [ 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.  0. -0.  0.  0.  0.  0. -0.  0.  0.  0.  0.  0.
  0.  0. -0.  0. -0.  0.  0.  0. -0.  0.]
Lasso Intercept: [1.9443274]
```

[18]:
```
# Predict the finger flexion on the training set using the Lasso model
Y_train_pred_lasso = lasso_model.predict(X_train)

# Predict the finger flexion on the test set using the Lasso model
Y_test_pred_lasso = lasso_model.predict(X_test)

# Compute MSE and R2 for the training set using the Lasso model
mse_train_lasso = mean_squared_error(Y_train, Y_train_pred_lasso)
r2_train_lasso = r2_score(Y_train, Y_train_pred_lasso)

# Compute MSE and R2 for the test set using the Lasso model
mse_test_lasso = mean_squared_error(Y_test, Y_test_pred_lasso)
r2_test_lasso = r2_score(Y_test, Y_test_pred_lasso)

print(f"Lasso Training MSE: {mse_train_lasso:.4f}")
print(f"Lasso Training R²: {r2_train_lasso:.4f}")
print(f"Lasso Test MSE: {mse_test_lasso:.4f}")
print(f"Lasso Test R²: {r2_test_lasso:.4f}")
```

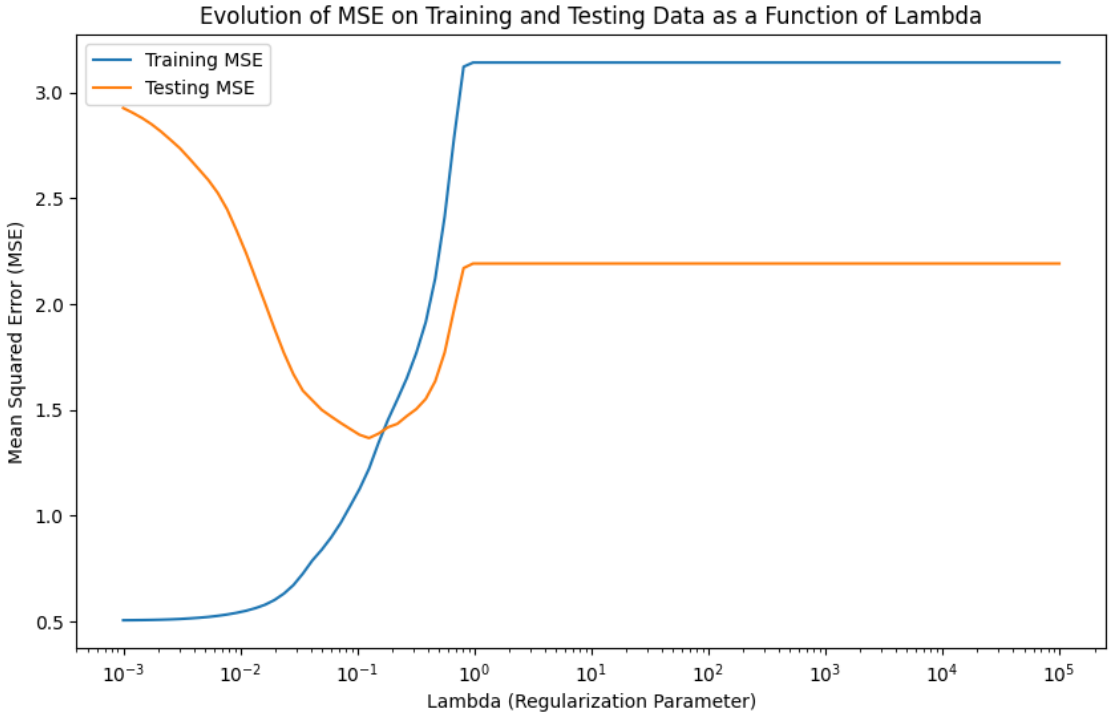
```
Lasso Training MSE: 3.1406
Lasso Training R²: 0.0000
Lasso Test MSE: 2.1907
Lasso Test R²: -0.0273
```

The Lasso model with a default lambda of 1 has essentially zeroed all coefficients, leading to an output that is essentially a constant (the intercept value). Lasso regression performs L1 regularization, which tends to shrink coefficients toward zero, especially for features that don't contribute significantly to reducing the error. In this case, the high regularization strength (lambda = 1) has driven all coefficients to zero. Consequently, the model is only using the intercept term to make predictions, essentially ignoring all features in the dataset.

- MSE Values: The mean squared error (MSE) on both the training and test sets remains relatively high, likely because the predictions are close to the mean of the target variable (dictated by the intercept only).
- R^2 Values: The $R^2 = 0$ for the training set and $R^2 = -0.0273$ for the test set indicate that

the model is not capturing any of the variance in the target variable beyond the mean. This suggests that the Lasso model is not learning the underlying pattern of finger flexion and is essentially making constant predictions.

[ ]:
```
# Generate 100 values of lambda on a logarithmic scale from 10^(-3) to 10^5
lst_reg = np.logspace(-3, 5, 100)

# Lists to store the MSE for each lambda on training and testing sets
mse_train_list = []
mse_test_list = []

for reg in lst_reg:
    # Initialize Lasso model with the current value of lambda
    lasso_model = Lasso(alpha=reg)
    lasso_model.fit(X_train, Y_train)

    # Predict on training and testing data
    Y_train_pred = lasso_model.predict(X_train)
    Y_test_pred = lasso_model.predict(X_test)

    # Calculate MSE for training and testing sets
    mse_train = mean_squared_error(Y_train, Y_train_pred)
    mse_test = mean_squared_error(Y_test, Y_test_pred)

    # Store the MSE values
    mse_train_list.append(mse_train)
    mse_test_list.append(mse_test)

# Plot the MSE as a function of lambda
plt.figure(figsize=(10, 6))
plt.plot(lst_reg, mse_train_list, label="Training MSE")
plt.plot(lst_reg, mse_test_list, label="Testing MSE")
plt.xscale("log")
plt.xlabel("Lambda (Regularization Parameter)")
plt.ylabel("Mean Squared Error (MSE)")
plt.title("Evolution of MSE on Training and Testing Data as a Function of␣
 ↪Lambda")
plt.legend()
plt.show()
```



For very low λ values (close to 10^{-3}), the training MSE is low, suggesting overfitting, as the model is too flexible. As λ increases, the regularization term forces more coefficients to zero, reducing overfitting, and testing MSE initially decrease. The testing MSE reaches its minimum around $\lambda \approx 10^{-1}$, indicating the optimal regularization level that balances bias and variance. Beyond this point, as λ continues to increase, both training and testing MSE rise sharply due to underfitting, as the model becomes overly simplistic with most coefficients shrinking to zero. This behavior reflects the trade-off between bias and variance in Lasso regularization.

[ ]:
```
# Step 1: Identify the  value with the minimum MSE on the test data
best_lambda_index = np.argmin(mse_test_list)
best_lambda = lambdas[best_lambda_index]

# Step 2: Fit a Lasso model with the best  value
lasso_model_best = Lasso(alpha=best_lambda)
lasso_model_best.fit(X_train, Y_train)

print(f"Best  value: {best_lambda}")
print("Lasso coefficients with the best  value:")
print(lasso_model_best.coef_)
```

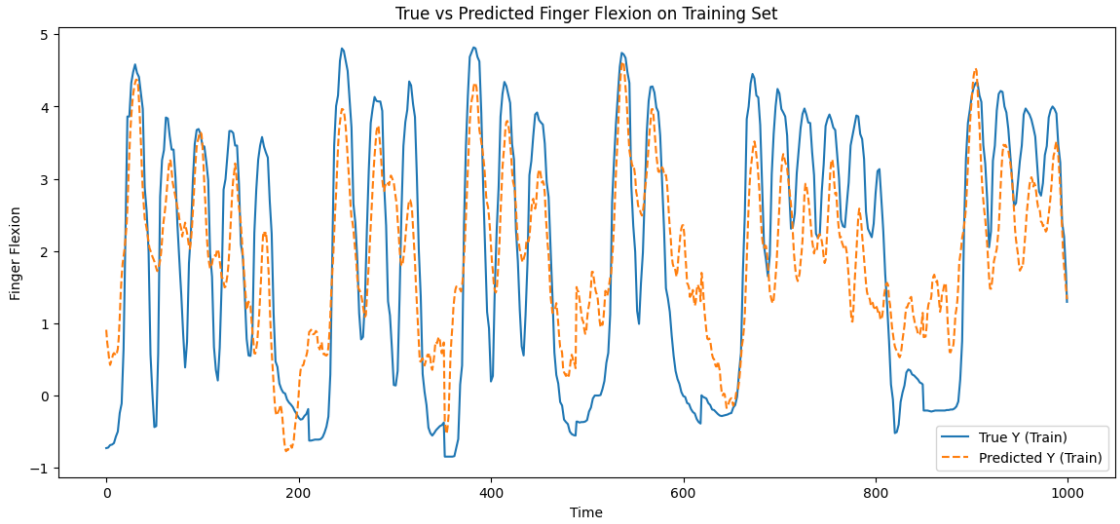
```
Best  value: 0.1261856883066021
Lasso coefficients with the best  value:
[-0.         -0.         -0.         -0.          0.          0.07757858
  0.         -0.          0.          0.         -0.1146682   0.
  0.09047535  0.          0.          0.37611771 -0.         -0.
```

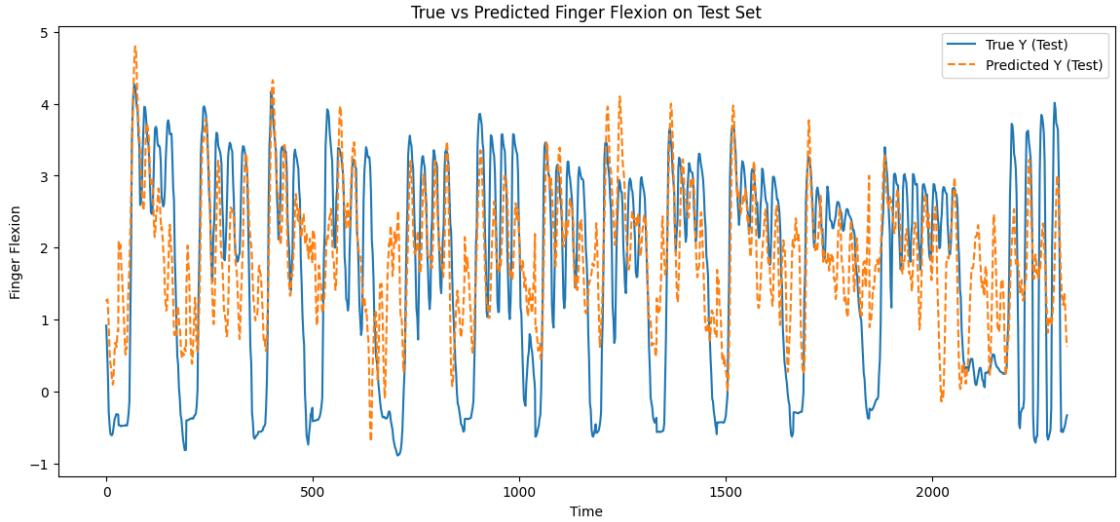
```
 0.          0.          0.         -0.09769496  0.          0.
 0.10678253  0.08349543 -0.          0.         -0.09337529 -0.
-0.          0.          0.         -0.         -0.          0.
-0.         -0.         -0.06143585 -0.         -0.         -0.14587363
 0.14635455 -0.          0.          0.27897347  0.          0.
-0.70862161 -0.          0.          0.          0.          0.
-0.          0.         -0.          0.         -0.         -0.
 0.          0.         -0.          0.31889792]
```

[21]:
```
# Compute MSE and R2 for the test set using the best Lasso model
Y_test_pred_best_lasso = lasso_model_best.predict(X_test)
mse_test_best_lasso = mean_squared_error(Y_test, Y_test_pred_best_lasso)
r2_test_best_lasso = r2_score(Y_test, Y_test_pred_best_lasso)

print(f"Best Lasso Test MSE: {mse_test_best_lasso:.4f}")
print(f"Best Lasso Test R²: {r2_test_best_lasso:.4f}")
```

```
Best Lasso Test MSE: 1.3660
Best Lasso Test R²: 0.3595
```

[22]:
```
# Predict the finger flexion on the training and test sets using the best Lasso␣
 ↪model
Y_train_pred_best_lasso = lasso_model_best.predict(X_train)
Y_test_pred_best_lasso = lasso_model_best.predict(X_test)

# Plot the true and predicted values for the training set
plt.figure(figsize=(14, 6))
plt.plot(Y_train, label='True Y (Train)')
plt.plot(Y_train_pred_best_lasso, label='Predicted Y (Train)', linestyle='--')
plt.title('True vs Predicted Finger Flexion on Training Set')
plt.xlabel('Time')
plt.ylabel('Finger Flexion')
plt.legend()
plt.show()

# Plot the true and predicted values for the test set
plt.figure(figsize=(14, 6))
plt.plot(Y_test, label='True Y (Test)')
plt.plot(Y_test_pred_best_lasso, label='Predicted Y (Test)',linestyle='--')
plt.title('True vs Predicted Finger Flexion on Test Set')
plt.xlabel('Time')
plt.ylabel('Finger Flexion')
plt.legend()
plt.show()
```

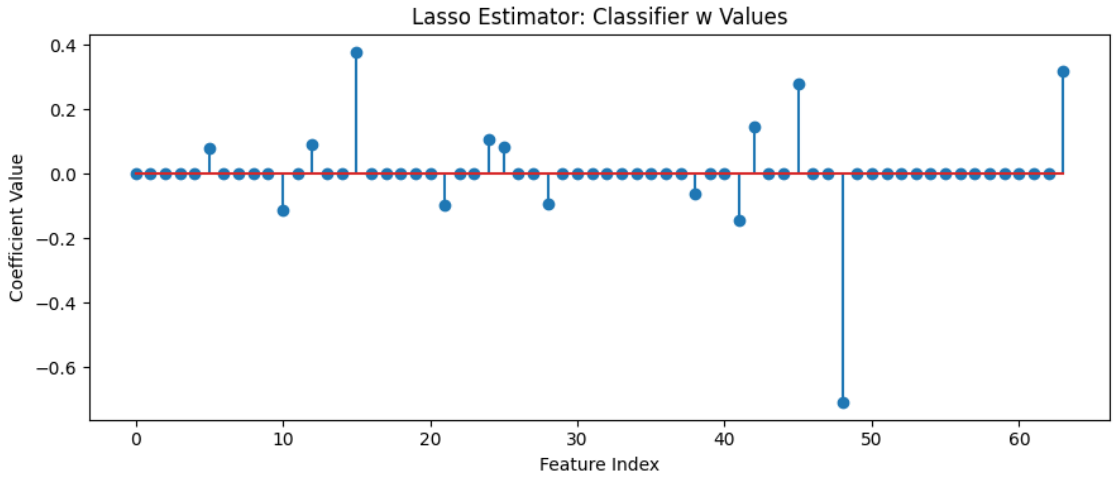


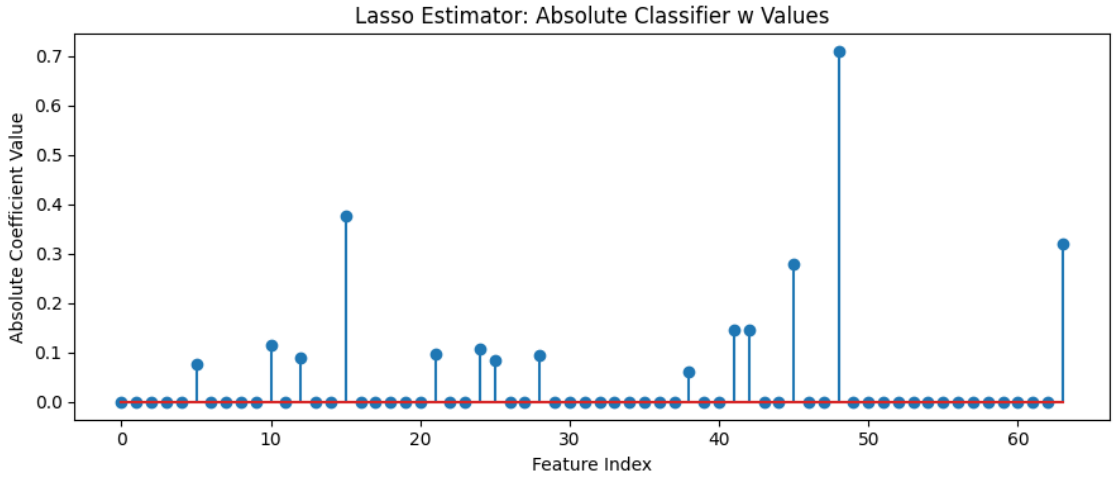


Using Variable selection with the Lasso can help identify the most relevant features for predicting finger flexion, reducing model complexity and improving generalization. By selecting only the most informative features, we can build a simpler model that captures the essential information in the data, leading to better performance on unseen data.

[23]:
```
# Plot the values of the classifiers w for Lasso estimator
plt.figure(figsize=(9, 4))

# Plot the values of w for Lasso estimator
plt.stem(lasso_model_best.coef_)
plt.title('Lasso Estimator: Classifier w Values')
plt.xlabel('Feature Index')
```

```
plt.ylabel('Coefficient Value')

plt.tight_layout()
plt.show()

# Plot the absolute values of w for Lasso estimator
plt.figure(figsize=(9, 4))

# Plot the absolute values of w for Lasso estimator
plt.stem(np.abs(lasso_model_best.coef_))
plt.title('Lasso Estimator: Absolute Classifier w Values')
plt.xlabel('Feature Index')
plt.ylabel('Absolute Coefficient Value')

plt.tight_layout()
plt.show()
```





[ ]:
```
# Get the absolute values of the coefficients
coefficients = np.abs(lasso_model_best.coef_)

# Get the indices of the 2 most important features
important_features_indices = np.argsort(coefficients)[-2:]

print("Indices of the 2 most important features:", important_features_indices)
```

```
Indices of the 2 most important features: [15 48]
```

The most important features selected by the Lasso model are those with the higher absolute values of the coefficients. These features are considered the most relevant for predicting finger flexion, as they contribute the most to the model's predictions. By focusing on these features, we can build a more interpretable and efficient model that captures the essential information in the data.

[ ]:
```
# Count the number of non-zero coefficients
selected_features_count = np.sum(coefficients != 0)

print("Number of features selected by the Lasso model:",␣
↪selected_features_count)
```

```
Number of features selected by the Lasso model: 14
```

We can reduce the number of electrodes used in the ECoG signal to only the most relevant ones selected by the Lasso model. This can help simplify the model, reduce computational complexity, and improve generalization to new, unseen data. By focusing on the most informative features, we can build a more interpretable and efficient model that captures the essential information in the data.

1.1.6 6-) Nonlinear regression

Random Forest Regression

[ ]:
```
from sklearn.ensemble import RandomForestRegressor

# Initialize the model with default parameters
rf_model = RandomForestRegressor()

# Fit the model on the training data
rf_model.fit(X_train, Y_train)

# Make predictions on the training and test sets
Y_train_pred_rf = rf_model.predict(X_train)
Y_test_pred_rf = rf_model.predict(X_test)

# Compute MSE and R² for the training set
mse_train_rf = mean_squared_error(Y_train, Y_train_pred_rf)
r2_train_rf = r2_score(Y_train, Y_train_pred_rf)
```

```
# Compute MSE and R² for the test set
mse_test_rf = mean_squared_error(Y_test, Y_test_pred_rf)
r2_test_rf = r2_score(Y_test, Y_test_pred_rf)

print(f"Random Forest Training MSE: {mse_train_rf:.4f}")
print(f"Random Forest Training R²: {r2_train_rf:.4f}")
print(f"Random Forest Test MSE: {mse_test_rf:.4f}")
print(f"Random Forest Test R²: {r2_test_rf:.4f}")
```

```
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py:1473: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)

Random Forest Training MSE: 0.0226
Random Forest Training R²: 0.9928
Random Forest Test MSE: 1.6490
Random Forest Test R²: 0.2267
```

Random Forest regression does not outperform the Lasso and Ridge regression models in terms of test Mean Squared Error (MSE) and R^2 values. The Random Forest has a test MSE of 1.6439 and a test R^2 of 0.2291, which are worse than the Lasso's best test MSE of 1.3660 and R^2 of 0.3595, and also worse than the Ridge's test MSE of 1.6239 and R^2 of 0.2385. While Random Forest shows excellent training performance (with a training MSE of 0.0241 and R^2 of 0.9923), this suggests overfitting, as it performs poorly on unseen data. Therefore, in this case, Random Forest is not the better model compared to Lasso and Ridge regression.

[ ]:
```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid,
                           scoring='neg_mean_squared_error', cv=5,
                           verbose=2, n_jobs=-1)

# Fit GridSearchCV on the training data
grid_search.fit(X_train, Y_train)
```

```python
# Get the best parameters and estimator
best_params = grid_search.best_params_
best_rf_model = grid_search.best_estimator_

# Make predictions with the best model
Y_train_pred_best_rf = best_rf_model.predict(X_train)
Y_test_pred_best_rf = best_rf_model.predict(X_test)

# Compute MSE and R² for the best model
mse_train_best_rf = mean_squared_error(Y_train, Y_train_pred_best_rf)
r2_train_best_rf = r2_score(Y_train, Y_train_pred_best_rf)

mse_test_best_rf = mean_squared_error(Y_test, Y_test_pred_best_rf)
r2_test_best_rf = r2_score(Y_test, Y_test_pred_best_rf)

print(f"Best Random Forest Training MSE: {mse_train_best_rf:.4f}")
print(f"Best Random Forest Training R²: {r2_train_best_rf:.4f}")
print(f"Best Random Forest Test MSE: {mse_test_best_rf:.4f}")
print(f"Best Random Forest Test R²: {r2_test_best_rf:.4f}")
print("Best Parameters:", best_params)
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits

c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py:540: FitFailedWarning:
540 fits failed out of a total of 1620.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
331 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py", line 1466, in wrapper
    estimator._validate_params()
  File "c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py", line 666, in _validate_params
    validate_parameter_constraints(
  File "c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\_param_validation.py", line 95, in
validate_parameter_constraints
    raise InvalidParameterError(
```

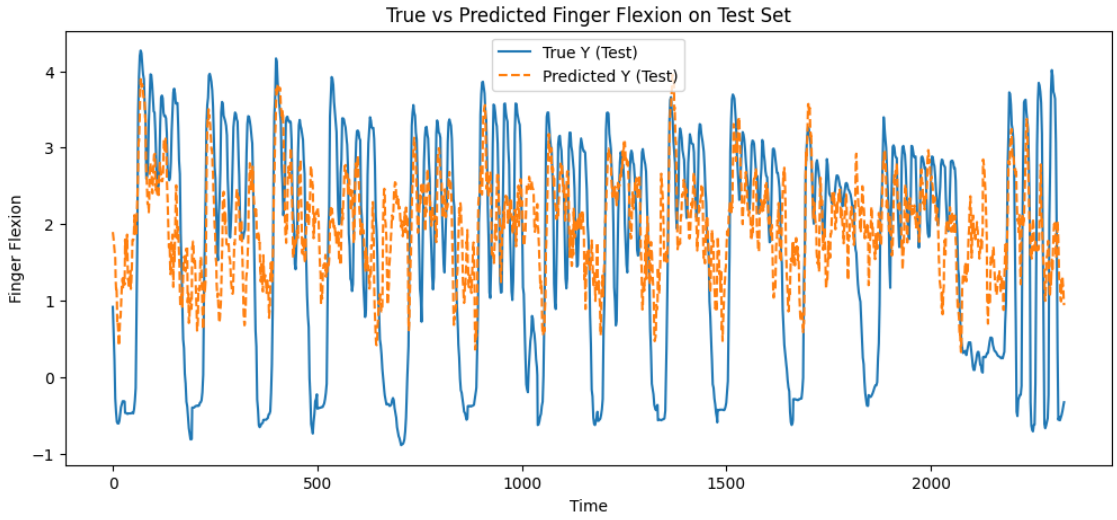
```
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of RandomForestRegressor must be an int in the range [1, inf), a float
in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto'
instead.

--------------------------------------------------------------------------------
209 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_validation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py", line 1466, in wrapper
    estimator._validate_params()
  File "c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py", line 666, in _validate_params
    validate_parameter_constraints(
  File "c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\_param_validation.py", line 95, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of RandomForestRegressor must be an int in the range [1, inf), a float
in the range (0.0, 1.0], a str among {'sqrt', 'log2'} or None. Got 'auto'
instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_search.py:1103: UserWarning: One or more of
the test scores are non-finite: [        nan         nan         nan         nan
nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan -2.05676642 -2.05440759 -2.04308429
 -2.12920567 -2.04896891 -2.04341833 -2.09769815 -2.06011953 -2.05851249
 -2.11286322 -2.06927061 -2.00857953 -2.11402175 -2.01732822 -2.02062258
 -2.06049416 -2.03542108 -2.03757579 -2.05216627 -2.0971555  -2.03032319
 -2.08797612 -2.05934758 -2.06323536 -2.13095094 -2.06985499 -2.00534578
 -2.1512371  -2.09512654 -2.05592984 -2.17188132 -2.13807779 -2.0598807
 -2.16233902 -2.10482454 -2.08673311 -2.05295008 -2.13076147 -2.07982062
 -2.0773999  -2.06253699 -2.08839749 -2.11514372 -2.05932985 -2.10823258
 -2.08017554 -2.08576969 -2.01494131 -2.13465773 -2.11417736 -2.12169386
 -2.09698891 -2.03364357 -2.04327414         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
```

```
 -2.03140402 -2.04783069 -2.02533919 -2.0467038  -2.03605807 -2.00562152
 -2.03606143 -2.04343613 -2.03089938 -2.09632754 -2.0612075  -2.03537676
 -2.0067779  -2.03410034 -1.9954063  -2.04765381 -2.12811489 -2.02822691
 -2.1278543  -2.06188357 -2.09993093 -2.0275562  -1.97618233 -2.03366337
 -2.06065269 -2.07439429 -2.04204549 -2.17989145 -2.11312675 -2.0544876
 -2.05619969 -2.05381136 -2.10193088 -2.19307289 -2.11704644 -2.04622115
 -2.07486474 -2.11048505 -2.05750292 -2.15890315 -2.09638643 -2.0943279
 -2.14272701 -2.06721927 -2.12290334 -2.09607761 -2.13918616 -2.13177723
 -2.10673786 -2.08050353 -2.09884624 -2.09151013 -2.03015741 -2.12734287
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan -1.97671724 -2.0403791  -2.03261275
 -2.09865082 -2.02150833 -2.03985877 -2.01791891 -2.02228737 -2.04885955
 -2.08899341 -2.06604094 -2.01968041 -2.03695403 -2.04449885 -1.97739482
 -2.10971994 -2.09025412 -2.09726509 -2.05471499 -2.05049199 -2.03009573
 -2.06268292 -2.08434396 -1.95478963 -2.0411449  -1.99335226 -2.02586155
 -2.123503   -2.14557955 -2.06293314 -2.11938842 -2.11883167 -2.0781483
 -2.08978954 -2.0486271  -2.08941891 -2.10708367 -2.01740056 -2.08737801
 -2.08624497 -2.07067948 -2.06581509 -2.15465982 -2.11632503 -2.08344881
 -2.09628252 -2.09850238 -2.08281717 -2.09958968 -2.05681838 -2.08625821
 -2.13028776 -2.11896979 -2.11542816         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
         nan         nan         nan         nan         nan         nan
 -2.05242707 -2.09312146 -2.02163338 -2.17439404 -1.96583128 -2.03142063
 -2.02169501 -2.04919986 -2.05707622 -2.03536417 -2.03292108 -2.04409661
 -2.03898669 -2.03292797 -2.0323977  -2.05522135 -2.09357017 -2.04480998
 -2.07881753 -2.02842986 -2.03712788 -1.99898172 -2.02565215 -2.03771018
 -2.05842279 -2.03431456 -2.0753518  -2.11802879 -2.17650702 -2.07266326
 -2.08639896 -2.14342064 -2.0917644  -2.13049758 -2.08003777 -2.09005765
 -2.17062838 -2.05788858 -2.0812133  -2.07098283 -2.06515158 -2.0596629
 -2.09865891 -2.08797626 -2.08395432 -2.0919915  -2.08340414 -2.0910472
 -2.11168071 -2.10477804 -2.08246159 -2.07463371 -2.06830471 -2.06516183]
  warnings.warn(
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py:1473: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)

Best Random Forest Training MSE: 0.0983
Best Random Forest Training R²: 0.9687
Best Random Forest Test MSE: 1.6159
Best Random Forest Test R²: 0.2423
Best Parameters: {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf':
```

```
4, 'min_samples_split': 5, 'n_estimators': 200}
```

The significant gap between training and test metrics in this case indicates that while the parameters selected improved performance somewhat, there may still be room for further tuning or a reevaluation of feature selection or model choice.

[29]:
```
# Plot the true and predicted values for the test set
plt.figure(figsize=(12, 5))
plt.plot(Y_test, label='True Y (Test)')
plt.plot(Y_test_pred_best_rf, label='Predicted Y (Test)', linestyle='--')
plt.title('True vs Predicted Finger Flexion on Test Set')
plt.xlabel('Time')
plt.ylabel('Finger Flexion')
plt.legend()
plt.show()
```



While Random Forest is a powerful method for capturing non-linear relationships, its effectiveness depends heavily on data quality, feature relevance, and hyperparameter tuning. The relatively low R^2 score implies that the model explains only a small portion of the variance in the target variable. Therefore, it may be beneficial to explore alternative regression techniques, enhance feature engineering, and ensure comprehensive hyperparameter optimization to improve performance.

Support Vector Regression

[ ]:
```
from sklearn.svm import SVR

# Initialize the model with default parameters
svr_model = SVR()

# Fit the model on the training data
svr_model.fit(X_train, Y_train)
```

```
# Make predictions on the training and test sets
Y_train_pred_svr = svr_model.predict(X_train)
Y_test_pred_svr = svr_model.predict(X_test)

# Compute MSE and R² for the training set
mse_train_svr = mean_squared_error(Y_train, Y_train_pred_svr)
r2_train_svr = r2_score(Y_train, Y_train_pred_svr)

# Compute MSE and R² for the test set
mse_test_svr = mean_squared_error(Y_test, Y_test_pred_svr)
r2_test_svr = r2_score(Y_test, Y_test_pred_svr)

print(f"SVR Training MSE: {mse_train_svr:.4f}")
print(f"SVR Training R²: {r2_train_svr:.4f}")
print(f"SVR Test MSE: {mse_test_svr:.4f}")
print(f"SVR Test R²: {r2_test_svr:.4f}")
```

```
SVR Training MSE: 0.0241
SVR Training R²: 0.9923
SVR Test MSE: 1.5152
SVR Test R²: 0.2895

c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

The Support Vector Regressor (SVR) exhibits strong performance relative to other regression techniques, with a test Mean Squared Error (MSE) of 1.5152 and an R^2 of 0.2895. Although it does not surpass the Lasso model, SVR remains a solid alternative to Least Squares, Ridge, and Random Forest regression. Further hyperparameter tuning could potentially improve its performance even more.

[ ]:
```
# Define the parameter grid
param_grid = {
    'kernel': ['linear', 'rbf'],
    'C': [0.1, 1, 10, 100],
    'epsilon': [0.01, 0.1, 0.5],
    'gamma': ['scale', 'auto']  # Only relevant for 'rbf'
}

# Initialize SVR
svr = SVR()

# Initialize GridSearchCV
```

```python
grid_search = GridSearchCV(svr, param_grid, cv=5,␣
↪scoring='neg_mean_squared_error')

# Fit the model
grid_search.fit(X_train, Y_train)

print("Best parameters found: ", grid_search.best_params_)
print("Best cross-validated MSE: ", -grid_search.best_score_)
```

```
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
```

```
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
```

```
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
```

```
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
```

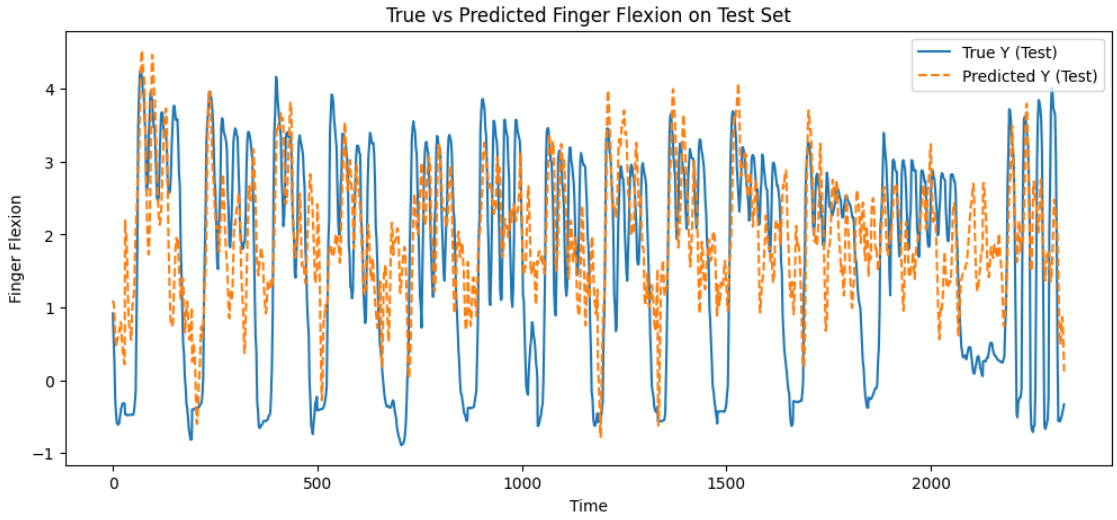
```
packages\sklearn\utils\validation.py:1339: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

Best parameters found:  {'C': 10, 'epsilon': 0.01, 'gamma': 'auto', 'kernel':
'rbf'}
Best cross-validated MSE:  1.6696735519459238
```

[ ]:
```python
# Get best parameters and estimator
best_params = grid_search.best_params_
best_svr_model = grid_search.best_estimator_

# Make predictions with the best model
Y_train_pred_best_svr = best_svr_model.predict(X_train)
Y_test_pred_best_svr = best_svr_model.predict(X_test)

# Compute MSE and R² for the best model
mse_train_best_svr = mean_squared_error(Y_train, Y_train_pred_best_svr)
r2_train_best_svr = r2_score(Y_train, Y_train_pred_best_svr)

mse_test_best_svr = mean_squared_error(Y_test, Y_test_pred_best_svr)
r2_test_best_svr = r2_score(Y_test, Y_test_pred_best_svr)

print(f"Best SVR Training MSE: {mse_train_best_svr:.4f}")
print(f"Best SVR Training R²: {r2_train_best_svr:.4f}")
print(f"Best SVR Test MSE: {mse_test_best_svr:.4f}")
print(f"Best SVR Test R²: {r2_test_best_svr:.4f}")
```

```
Best SVR Training MSE: 0.0018
Best SVR Training R²: 0.9994
Best SVR Test MSE: 1.5325
Best SVR Test R²: 0.2814
```

Here, the performance of the SVR model isn't dependent on the parameters selected, as the default hyperparameters already yield to the same results.

[33]:
```python
# Plot the true and predicted values for the test set
plt.figure(figsize=(12, 5))
plt.plot(Y_test, label='True Y (Test)')
plt.plot(Y_test_pred_best_svr, label='Predicted Y (Test)', linestyle='--')
plt.title('True vs Predicted Finger Flexion on Test Set')
plt.xlabel('Time')
plt.ylabel('Finger Flexion')
plt.legend()
plt.show()
```



The Support Vector Regressor (SVR) shows promising results, achieving a training Mean Squared Error (MSE) of 0.0018 and an R^2 of 0.9994, indicating excellent fit on the training data. However, the test MSE of 1.5325 and an R^2 of 0.2814 suggest that the model struggles to generalize to unseen data. The relatively low test R^2 indicates that the SVR does not explain much variance in the test set, which raises concerns about overfitting. Overall, while SVR demonstrates strong training performance, the disparity between training and test results indicates that further optimization and validation are necessary to enhance its generalization capability.

1.1.7 MLPClassifier

[ ]:
```
from sklearn.neural_network import MLPRegressor

# Initialize the MLPRegressor with default parameters
mlp_model = MLPRegressor()

# Fit the model on the training data
mlp_model.fit(X_train, Y_train)

# Make predictions on the training and test sets
Y_train_pred_mlp = mlp_model.predict(X_train)
Y_test_pred_mlp = mlp_model.predict(X_test)

# Compute MSE and R² for the training set
mse_train_mlp = mean_squared_error(Y_train, Y_train_pred_mlp)
r2_train_mlp = r2_score(Y_train, Y_train_pred_mlp)

# Compute MSE and R² for the test set
mse_test_mlp = mean_squared_error(Y_test, Y_test_pred_mlp)
r2_test_mlp = r2_score(Y_test, Y_test_pred_mlp)
```

```
print(f"MLP Training MSE: {mse_train_mlp:.4f}")
print(f"MLP Training R²: {r2_train_mlp:.4f}")
print(f"MLP Test MSE: {mse_test_mlp:.4f}")
print(f"MLP Test R²: {r2_test_mlp:.4f}")
```

```
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:1631:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

```
MLP Training MSE: 0.0067
MLP Training R²: 0.9979
MLP Test MSE: 2.2821
MLP Test R²: -0.0701
```

The Multi-Layer Perceptron Regressor (MLPRegressor) results show a training Mean Squared Error (MSE) of 0.0067 and an R^2 of 0.9979, indicating an excellent fit on the training data. However, its test MSE of 2.2821 and R^2 of -0.0701 suggest that the model fails to generalize to unseen data, as evidenced by the negative R^2 value, which indicates that the model performs worse than a horizontal line representing the mean of the target variable. The MLPRegressor has a lower test MSE (2.2821) than LS (3.0455) but performs worse than both Ridge and Lasso in terms of MSE and R^2 on the test set. All models exhibit some degree of overfitting; however, the MLPRegressor demonstrates a significant drop in test performance compared to its training performance.

[37]:
```
# Define the parameter grid
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 50)],
    'activation': ['relu', 'tanh'],
    'alpha': [0.001, 0.01],
    'learning_rate_init': [0.001, 0.01],
}

# Initialize the MLPRegressor
mlp_model = MLPRegressor()

# Setup GridSearchCV
grid_search = GridSearchCV(mlp_model, param_grid, cv=5,␣
 ↪scoring='neg_mean_squared_error', n_jobs=-1)

# Fit the model on the training data
grid_search.fit(X_train, Y_train)

# Best parameters and score
best_params = grid_search.best_params_

print(f"Best Parameters: {best_params}")
```

```
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:1631:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

```
Best Parameters: {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes':
(100, 50), 'learning_rate_init': 0.01}
```

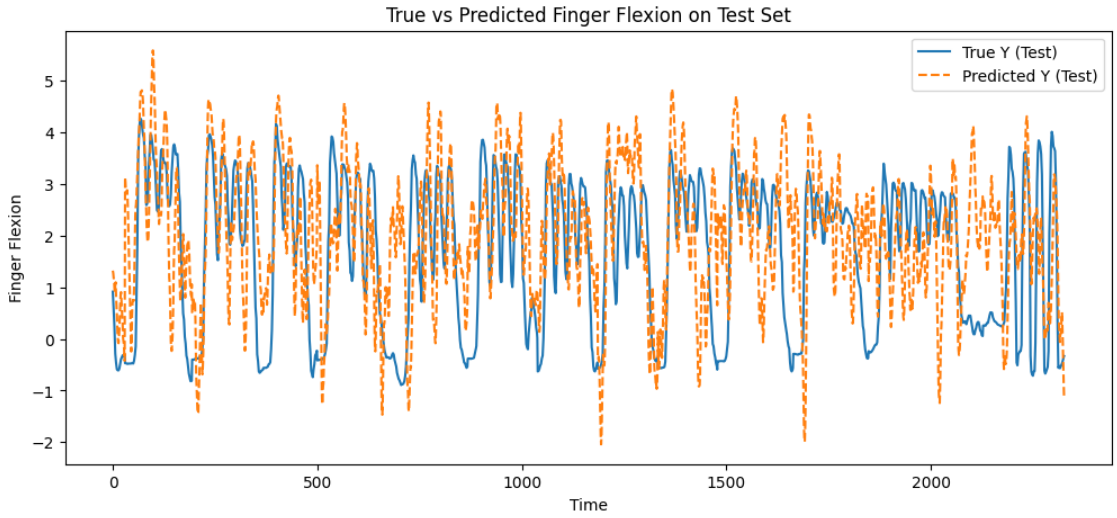
[ ]:
```python
# Get best parameters and estimator
best_params = grid_search.best_params_
best_mlp_model = grid_search.best_estimator_

# Make predictions with the best model
Y_train_pred_best_mlp = best_mlp_model.predict(X_train)
Y_test_pred_best_mlp = best_mlp_model.predict(X_test)

# Compute MSE and R² for the best model
mse_train_best_mlp = mean_squared_error(Y_train, Y_train_pred_best_mlp)
r2_train_best_mlp = r2_score(Y_train, Y_train_pred_best_mlp)

mse_test_best_mlp = mean_squared_error(Y_test, Y_test_pred_best_mlp)
r2_test_best_mlp = r2_score(Y_test, Y_test_pred_best_mlp)

print(f"Best MLP Training MSE: {mse_train_best_mlp:.4f}")
print(f"Best MLP Training R²: {r2_train_best_mlp:.4f}")
print(f"Best MLP Test MSE: {mse_test_best_mlp:.4f}")
print(f"Best MLP Test R²: {r2_test_best_mlp:.4f}")
```

```
Best MLP Training MSE: 0.0034
Best MLP Training R²: 0.9989
Best MLP Test MSE: 2.2366
Best MLP Test R²: -0.0488
```

Using the MLPRegressor, the performance is dependent on the hyperparameters selected, as different configurations can lead to varying results. In this case, the default hyperparameters do not yield optimal performance, as the model struggles to generalize to unseen data. Further hyperparameter tuning, feature engineering, or regularization may be necessary to improve the model's generalization capability.

[41]:
```python
# Plot the true and predicted values for the test set
plt.figure(figsize=(12, 5))
plt.plot(Y_test, label='True Y (Test)')
plt.plot(Y_test_pred_best_mlp, label='Predicted Y (Test)', linestyle='--')
plt.title('True vs Predicted Finger Flexion on Test Set')
plt.xlabel('Time')
plt.ylabel('Finger Flexion')
plt.legend()
```

```
plt.show()
```



The MLPRegressor struggles to generalize effectively to unseen data. This performance suggests potential overfitting, where the model learns the noise in the training data rather than the underlying patterns. Overall, while MLP can be a powerful regression technique due to its ability to model complex relationships, its efficacy depends significantly on more proper tuning and regularization to ensure good generalization on test data.

Gradient Boosting Regressor

[46]:
```
from sklearn.ensemble import GradientBoostingRegressor

# Initialize the GradientBoostingRegressor with default parameters
gbr_model = GradientBoostingRegressor()

# Fit the model on the training data
gbr_model.fit(X_train, Y_train)

# Make predictions on the training and test sets
Y_train_pred_gbr = gbr_model.predict(X_train)
Y_test_pred_gbr = gbr_model.predict(X_test)

# Compute MSE and R² for the training set
mse_train_gbr = mean_squared_error(Y_train, Y_train_pred_gbr)
r2_train_gbr = r2_score(Y_train, Y_train_pred_gbr)

# Compute MSE and R² for the test set
mse_test_gbr = mean_squared_error(Y_test, Y_test_pred_gbr)
r2_test_gbr = r2_score(Y_test, Y_test_pred_gbr)
```

```
print(f"Gradient Boosting Training MSE: {mse_train_gbr:.4f}")
print(f"Gradient Boosting Training R²: {r2_train_gbr:.4f}")
print(f"Gradient Boosting Test MSE: {mse_test_gbr:.4f}")
print(f"Gradient Boosting Test R²: {r2_test_gbr:.4f}")
```

```
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\ensemble\_gb.py:668: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)  # TODO: Is this still required?
```

```
Gradient Boosting Training MSE: 0.1133
Gradient Boosting Training R²: 0.9639
Gradient Boosting Test MSE: 1.6913
Gradient Boosting Test R²: 0.2069
```

Gradient Boosting shows competitive performance compared to the linear regression models (LS, Ridge, Lasso) and the Random Forest model, but it is outperformed by Ridge and Lasso in terms of R^2. Therefore, it can be considered a solid option, but further tuning and experimentation may be needed to fully maximize its potential in this application.

[ ]:
```
# Define the hyperparameter grid
param_grid = {
    'learning_rate': [0.01, 0.1],
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'subsample': [0.5, 1.0]
}

# Setup the GridSearchCV
grid_search = GridSearchCV(estimator=gbr_model, param_grid=param_grid,
                           scoring='neg_mean_squared_error', cv=5, verbose=1,␣
 ↪n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train, Y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validated MSE:", -grid_search.best_score_)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
```

```
c:\Users\moham\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\ensemble\_gb.py:668: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)  # TODO: Is this still required?
```

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'min_samples_split': 10,
```

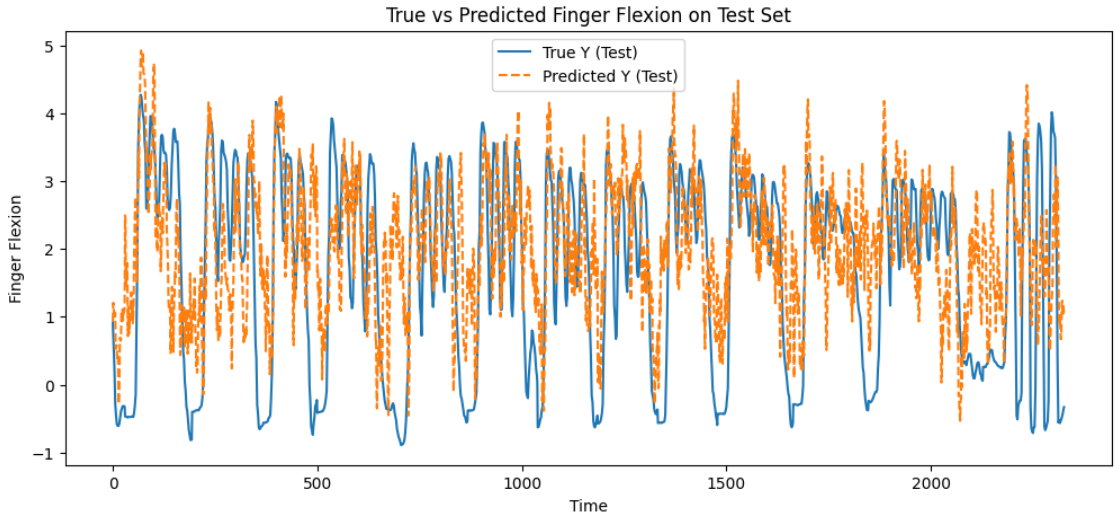
```
'n_estimators': 200, 'subsample': 0.5}
Best Cross-Validated MSE: 1.7229492519041485
```

[ ]:
```python
# Get best parameters and estimator
best_params = grid_search.best_params_
best_gbr_model = grid_search.best_estimator_

# Make predictions with the best model
Y_train_pred_best_gbr = best_gbr_model.predict(X_train)
Y_test_pred_best_gbr = best_gbr_model.predict(X_test)

# Compute MSE and R² for the best model
mse_train_best_gbr = mean_squared_error(Y_train, Y_train_pred_best_gbr)
r2_train_best_gbr = r2_score(Y_train, Y_train_pred_best_gbr)

mse_test_best_gbr = mean_squared_error(Y_test, Y_test_pred_best_gbr)
r2_test_best_gbr = r2_score(Y_test, Y_test_pred_best_gbr)

print(f"Best Gradient Boosting Training MSE: {mse_train_best_gbr:.4f}")
print(f"Best Gradient Boosting Training R²: {r2_train_best_gbr:.4f}")
print(f"Best Gradient Boosting Test MSE: {mse_test_best_gbr:.4f}")
print(f"Best Gradient Boosting Test R²: {r2_test_best_gbr:.4f}")
```

```
Best Gradient Boosting Training MSE: 0.0259
Best Gradient Boosting Training R²: 0.9918
Best Gradient Boosting Test MSE: 1.6369
Best Gradient Boosting Test R²: 0.2324
```

The Gradient Boosting Regressor isn't very dependent on the hyperparameters selected, as the default hyperparameters already yield practically the same results.

[49]:
```python
# Plot the true and predicted values for the test set
plt.figure(figsize=(12, 5))
plt.plot(Y_test, label='True Y (Test)')
plt.plot(Y_test_pred_best_gbr, label='Predicted Y (Test)', linestyle='--')
plt.title('True vs Predicted Finger Flexion on Test Set')
plt.xlabel('Time')
plt.ylabel('Finger Flexion')
plt.legend()
plt.show()
```



The performance of the Gradient Boosting Regressor indicates a promising regression method for predicting finger flexion from ECoG signals. With a training Mean Squared Error (MSE) of 0.0259 and a training R^2 of 0.9918, the model demonstrates a strong ability to fit the training data closely, suggesting it effectively captures the underlying relationships between the features and the target variable. However, the test MSE of 1.6369 and an R^2 of 0.2324 reveal a more moderate performance on unseen data, indicating potential overfitting or limited generalization. Despite this, the results show that Gradient Boosting can leverage complex patterns in the data, making it a valuable tool in this context.

1.1.8 7-) Final comparison of the performances

[50]:
```
results = {
    'Model': ['Least Squares regression', 'Ridge Regression', 'Lasso␣
 ↪Regression', 'Random Forest', 'SVR', 'MLP', 'Gradient Boosting'],
    'Test MSE': [mse_test, mse_test_best_ridge, mse_test_best_lasso,␣
 ↪mse_test_best_rf, mse_test_best_svr, mse_test_best_mlp, mse_test_best_gbr],
    'Test R²': [r2_test, r2_test_best_ridge, r2_test_best_lasso,␣
 ↪r2_test_best_rf, r2_test_best_svr, r2_test_best_mlp, r2_test_best_gbr]
}
results_df = pd.DataFrame(results)
print(results_df)
```

```
                    Model  Test MSE   Test R²
0  Least Squares regression  2.190676 -0.428170
1          Ridge Regression  1.623906  0.238488
2          Lasso Regression  1.365954  0.359451
3             Random Forest  1.615879  0.242252
4                       SVR  1.532462  0.281369
5                       MLP  2.236554 -0.048807
6         Gradient Boosting  1.636857  0.232414
```

In practical terms, **Lasso Regression** and **Support Vector Regression (SVR)** demonstrate the best performance based on the test Mean Squared Error (MSE) and R^2 metrics.

Both models outperform Least Squares, Ridge, Random Forest, and MLP on the test set, showing that **regularization (Lasso)** and **Support Vector Regression (SVR)** can offer practical advantages for predictive accuracy and generalization.

- **What are the most interpretable models?**

Least Squares (Linear Regression), Ridge Regression, and Lasso Regression are the most interpretable. Least Squares (Linear Regression): This model is often the easiest to interpret, as it provides coefficients that directly indicate the effect of each feature on the output. Ridge also provides an interpretable linear model with coefficients, though it adds a small regularization term. The regularization can slightly reduce individual coefficient values, but the model's structure remains interpretable. Lasso Regression: Lasso is similar to Ridge in interpretability, but it tends to drive some coefficients to zero, effectively performing feature selection. This can simplify the model and make it more interpretable by focusing on the most relevant features.

On the other hand, Random Forest, Support Vector Regression (SVR), and Multi-Layer Perceptron (MLP) are generally less interpretable.

- **Which model is best from a medical/practical perspective?**

From a medical or practical perspective, a model that balances interpretability and predictive accuracy is often preferred. This enables healthcare professionals to understand the model's decision-making process and potentially gain insights into brain-signal-to-movement relationships.

Lasso's interpretability makes it advantageous in a medical setting, as it can highlight the most critical ECoG features influencing thumb flexion prediction. Its sparse solution (selecting only the most relevant features) simplifies analysis and could potentially help isolate important brain regions or signal patterns, which is insightful for clinicians.

- **Do we need non-linearity in this application?**

The non-linear models like Random Forest and MLP didn't yield optimal results on the test set, which might suggest that their additional complexity didn't translate into better generalization for this specific dataset.

- **Is validation on the test data a good practice?**

Validation on the test data is indeed a crucial practice, but it should be done with caution. The test dataset serves as an independent evaluation of the model's performance and helps ensure that the model generalizes well to unseen data. However, over-reliance on test data can lead to overfitting, where the model performs well on test data but poorly in real-world scenarios.

- **What would you do if you need to provide a model to a client for prediction in production?**

When providing a model to a client for prediction in a production environment, I would implement a comprehensive strategy to ensure its effectiveness and reliability. First, I would ensure a robust train-validation-test split during the model development phase, allowing for thorough evaluation and tuning of the model's performance without biasing the final test results. Once the model is ready, I would document its architecture, hyperparameters, and expected performance metrics, providing the client with clear guidelines on its usage and limitations.

Importantly, I would establish a monitoring system to track the model's performance in real-time once deployed. This involves regularly evaluating its predictions against actual outcomes to detect any drift or degradation in performance. If the model's accuracy diminishes over time due to changes in the underlying data patterns, I would recommend periodic retraining or fine-tuning to maintain its predictive capabilities.

1.1.9 8-) Conclusion

Reflecting on this project, I found the exploration of various regression methods both enlightening and challenging. One of the key difficulties was grasping the mathematical concepts underlying these models, such as the principles of boosting in Gradient Boosting Regression. I realized that a deeper understanding of these concepts requires dedicated time and focus to fully appreciate their implications in practice. In future projects, I would allocate more time to study these mathematical foundations, as they are crucial for effectively applying these techniques in real-world scenarios.

Overall, the lab provided valuable insights into the practical application of different regression methods, emphasizing the importance of understanding overfitting and the necessity of measuring error on both training and test data. This experience has reinforced my ability to select and implement suitable models while being mindful of their limitations. Furthermore, I can envision utilizing these tools in a professional setting, where rigorous data analysis and model evaluation are essential for making informed decisions. The skills I've developed in this project will undoubtedly serve as a strong foundation for my future endeavors in data science and machine learning.