Machine Learning & Big Data
Assignment 6

Introduction

In this assignment, we will analyze 2 things:

1) How to use sci-kit functions to train a neural network model

2) Using cross-validation and test sets to choose a degree to polynomial

Figure 1.1 & 1.2

- We are overfitting our examples
- We created our model and obtained a prediction
  - The model was trained by sci-kit function
- We just used training set and test set
  - %67 training, % 33 test

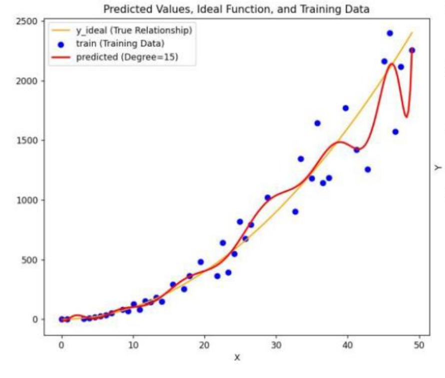
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

def gen_data(m, seed=1, scale=0.7):
    """ Generate a dataset based on x^2 with added noise """
    c = 0
    x_train = np.linspace(0, 49, m)
    np.random.seed(seed)
    y_ideal = x_train**2 + c
    y_train = y_ideal + scale * y_ideal * (np.random.sample((m,)) - 0.5)
    x_ideal = x_train
    return x_train, y_train, x_ideal, y_ideal

def compute_error(predictions, targets):
    """ Compute mean squared error """
    return mean_squared_error(targets, predictions) / 2

# Generate data
m = 64
x_train, y_train, x_ideal, y_ideal = gen_data(m)

# Split data into training and test sets
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.33, random_state=1)

# Transform data into polynomial features
poly_degree = 15
poly_features = PolynomialFeatures(degree=poly_degree, include_bias=False)

# Fit and transform training data
x_poly_train = poly_features.fit_transform(x_train.reshape(-1, 1))

# Apply same transformation to test data
x_poly_test = poly_features.transform(x_test.reshape(-1, 1))

# Normalize features
scaler = StandardScaler()
x_poly_train_scaled = scaler.fit_transform(x_poly_train)
x_poly_test_scaled = scaler.transform(x_poly_test)

# Train Linear Regression model
model = LinearRegression()
model.fit(x_poly_train_scaled, y_train)

# Generate x values for plotting predictions (more finely spaced)
x_values = np.linspace(0, 49, 200)  # Range of x values for prediction
```

```python
x_poly_train_scaled = scaler.fit_transform(x_poly_train)
x_poly_test_scaled = scaler.transform(x_poly_test)

# Train Linear Regression model
model = LinearRegression()
model.fit(x_poly_train_scaled, y_train)

# Generate x values for plotting predictions (more finely spaced)
x_values = np.linspace(0, 49, 200)  # Range of x values for prediction

# Transform x_values into polynomial features and scale them
x_poly_values = poly_features.transform(x_values.reshape(-1, 1))
x_poly_values_scaled = scaler.transform(x_poly_values)

# Make predictions on the new x values
y_pred_values = model.predict(x_poly_values_scaled)

# Compute training and test errors
y_pred_train = model.predict(x_poly_train_scaled)
y_pred_test = model.predict(x_poly_test_scaled)
train_error = compute_error(y_pred_train, y_train)
test_error = compute_error(y_pred_test, y_test)

# Display errors
print(f"Training Error (J_train): {train_error:.2f}")
print(f"Test Error (J_test): {test_error:.2f}")

# Plotting the graphs
plt.figure(figsize=(14, 6))

# Plot 1: Ideal Function and Training Data
plt.subplot(1, 2, 1)
plt.plot(x_ideal, y_ideal, color='orange', label='y_ideal (True Relationship)')
plt.scatter(x_train, y_train, color='blue', label='train (Training Data)')
plt.title('Ideal Function and Training Data')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()

# Plot 2: Predicted Values, Ideal Function, and Training Data
plt.subplot(1, 2, 2)
plt.plot(x_ideal, y_ideal, color='orange', label='y_ideal (True Relationship)')
plt.scatter(x_train, y_train, color='blue', label='train (Training Data)')
plt.plot(x_values, y_pred_values, color='red', label='predicted (Degree=15)', linewidth=2)
plt.title('Predicted Values, Ideal Function, and Training Data')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
```

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

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
PS C:\Users\ardah\UCM> python -u "c:\Users\ardah\UCM\p6\p6_2.py"
Training Error (J_train): 11855.05
Test Error (J_test): 48579.59
PS C:\Users\ardah\UCM>
```



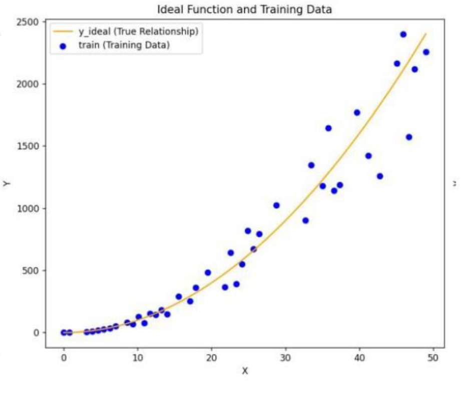
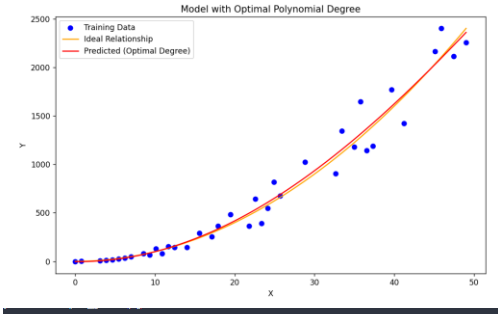


Fig 1.3

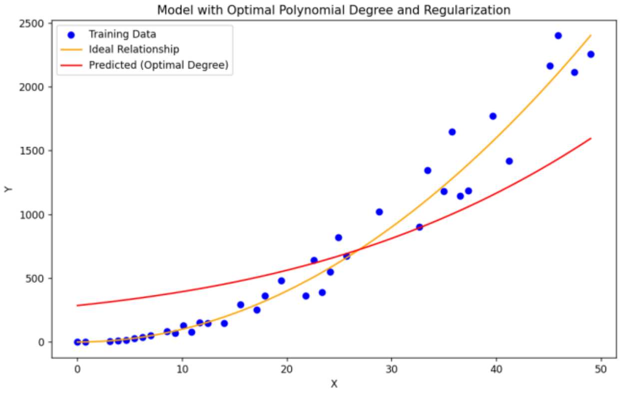
- We are now using cross-validation set also
    - %60 training, %20 cross-validation, %20 test set
- We are not regularizing, yet…




```python
# Generate data
m = 64
x_train, y_train, x_ideal, y_ideal = gen_data(m)
x_train = x_train[:, None]

# Split data into training, validation, and test sets
x_train, x_temp, y_train, y_temp = train_test_split(x_train, y_train, test_size=0.4, random_state=1)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5, random_state=1)

best_degree = None
best_error = float('inf')

# Loop through polynomial degrees
for degree in range(1, 11):
    # Transform data into polynomial features
    poly_features = PolynomialFeatures(degree=degree, include_bias=False)
    x_poly_train = poly_features.fit_transform(x_train)
    x_poly_val = poly_features.transform(x_val)

    # Scale data
    scaler = StandardScaler()
    x_poly_train_scaled = scaler.fit_transform(x_poly_train)
    x_poly_val_scaled = scaler.transform(x_poly_val)

    # Train Ridge model
    model = Ridge()
    model.fit(x_poly_train_scaled, y_train)

    # Predict validation set
    y_pred_val = model.predict(x_poly_val_scaled)

    # Compute validation error
    val_error = mean_squared_error(y_val, y_pred_val)

    # Check if this is the best model so far
    if val_error < best_error:
        best_error = val_error
        best_degree = degree

print("Best Degree:", best_degree)

# Now repeat the process for selecting the best value for λ
param_grid = {'alpha': [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 300, 600, 900]}
poly_features = PolynomialFeatures(degree=best_degree, include_bias=False)
x_poly_train = poly_features.fit_transform(x_train)
x_poly_val = poly_features.transform(x_val)
x_poly_test = poly_features.transform(x_test)

scaler = StandardScaler()
x_poly_train_scaled = scaler.fit_transform(x_poly_train)
x_poly_val_scaled = scaler.transform(x_poly_val)
x_poly_test_scaled = scaler.transform(x_poly_test)

grid_search = GridSearchCV(Ridge(), param_grid, cv=5)
grid_search.fit(x_poly_train_scaled, y_train)
```

Fig 1.4

- Now we are also regularizing



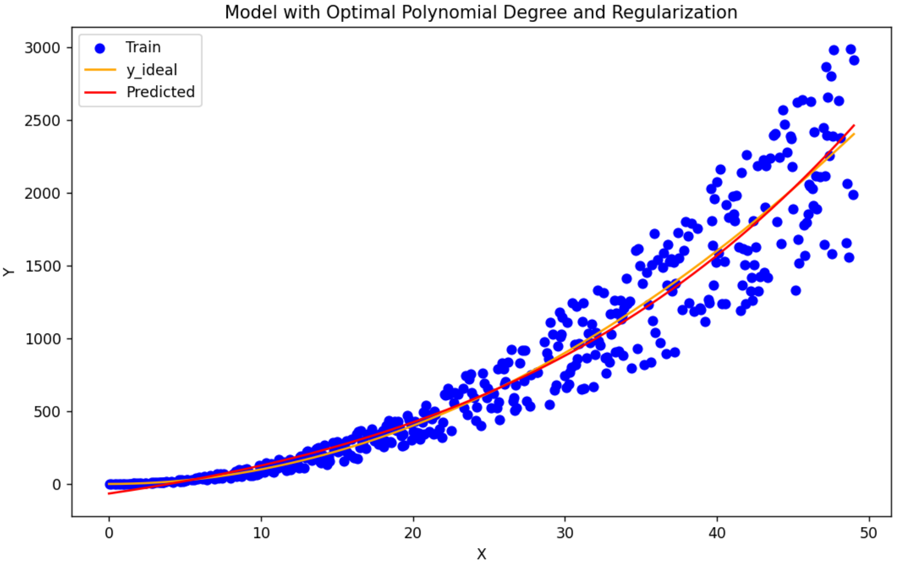

```python
# Generate data
m = 64
x_train, y_train, x_ideal, y_ideal = gen_data(m)
x_train = x_train[:, None]

# Split data into training, validation, and test sets
x_train, x_temp, y_train, y_temp = train_test_split(x_train, y_train, test_size=0.4, random_state=1)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5, random_state=1)

best_degree = None
best_error = float('inf')

# Loop through polynomial degrees
for degree in range(1, 11):
    # Transform data into polynomial features
    poly_features = PolynomialFeatures(degree=degree, include_bias=False)
    x_poly_train = poly_features.fit_transform(x_train)
    x_poly_val = poly_features.transform(x_val)

    # Scale data
    scaler = StandardScaler()
    x_poly_train_scaled = scaler.fit_transform(x_poly_train)
    x_poly_val_scaled = scaler.transform(x_poly_val)

    # Train Ridge model with different alpha values
    alphas = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 300, 600, 900]
    best_alpha = None
    best_alpha_error = float('inf')

    for alpha in alphas:
        model = Ridge(alpha=alpha)
        model.fit(x_poly_train_scaled, y_train)
        y_pred_val = model.predict(x_poly_val_scaled)
        val_error = mean_squared_error(y_val, y_pred_val)

        if val_error < best_alpha_error:
            best_alpha_error = val_error
            best_alpha = alpha

    # Retrain Ridge model with the best alpha
    model = Ridge(alpha=best_alpha)
    model.fit(x_poly_train_scaled, y_train)
    y_pred_val = model.predict(x_poly_val_scaled)
    val_error = mean_squared_error(y_val, y_pred_val)

    # Check if this is the best model so far
    if val_error < best_error:
        best_error = val_error
        best_degree = degree

print("Best Degree:", best_degree)
print("Best Alpha:", best_alpha)

# Now evaluate the selected model on the test set
poly_features = PolynomialFeatures(degree=best_degree, include_bias=False)
x_poly_train = poly_features.fit_transform(x_train)
x_poly_test = poly_features.transform(x_test)

scaler = StandardScaler()
x_poly_train_scaled = scaler.fit_transform(x_poly_train)
x_poly_test_scaled = scaler.transform(x_poly_test)

model = Ridge(alpha=best_alpha)
model.fit(x_poly_train_scaled, y_train)
y_pred_test = model.predict(x_poly_test_scaled)
test_error = mean_squared_error(y_test, y_pred_test)
print("Test Error:", test_error)


# Plot train, predicted, and y_ideal for the optimal polynomial degree with regularization term
x_values = np.linspace(0, 49, 200)
x_poly_values = poly_features.transform(x_values.reshape(-1, 1))
x_poly_values_scaled = scaler.transform(x_poly_values)
y_pred_values = model.predict(x_poly_values_scaled)

plt.figure(figsize=(10, 6))
plt.scatter(x_train, y_train, label='Training Data', color='blue')
plt.plot(x_ideal, y_ideal, label='Ideal Relationship', color='orange')
plt.plot(x_values, y_pred_values, label='Predicted (Optimal Degree)', color='red')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Model with Optimal Polynomial Degree and Regularization')
```

Fig 1.5

- We are increasing our data size to 750
    - We are training with hyper-paramters

```python
m = 750
x_train, y_train, x_ideal, y_ideal = gen_data(m)
x_train = x_train[:, None]

# Split data into training, validation, and test sets
x_train, x_temp, y_train, y_temp = train_test_split(x_train, y_train, test_size=0.4, random_state=1)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5, random_state=1)

best_degree = None
best_alpha = None
best_error = float('inf')

# Define the grid of hyperparameters
param_grid = {
    'degree': range(1, 20),  # Increase the range of degrees
    'alpha': [1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 300, 600, 900]  # Adjust the range of alphas
}

# Loop through all combinations of hyperparameters
for degree in param_grid['degree']:
    for alpha in param_grid['alpha']:
        # Transform data into polynomial features
        poly_features = PolynomialFeatures(degree=degree, include_bias=False)
        x_poly_train = poly_features.fit_transform(x_train)
        x_poly_val = poly_features.transform(x_val)

        # Scale data
        scaler = StandardScaler()
        x_poly_train_scaled = scaler.fit_transform(x_poly_train)
        x_poly_val_scaled = scaler.transform(x_poly_val)

        # Train Ridge model
        model = Ridge(alpha=alpha)
        model.fit(x_poly_train_scaled, y_train)

        # Predict validation set
        y_pred_val = model.predict(x_poly_val_scaled)

        # Compute validation error
        val_error = compute_error(y_val, y_pred_val)

        # Check if this is the best model so far
        if val_error < best_error:
            best_error = val_error
            best_degree = degree
            best_alpha = alpha

print("Best Degree:", best_degree)
print("Best Alpha:", best_alpha)

# Now train the best model on the entire training set with the optimal hyperparameters
poly_features = PolynomialFeatures(degree=best_degree, include_bias=False)
x_poly_train = poly_features.fit_transform(x_train)
x_poly_test = poly_features.transform(x_test)

scaler = StandardScaler()
x_poly_train_scaled = scaler.fit_transform(x_poly_train)
x_poly_test_scaled = scaler.transform(x_poly_test)

best_model = Ridge(alpha=best_alpha)
best_model.fit(x_poly_train_scaled, y_train)

# Evaluate the selected model on the test set
y_pred_test = best_model.predict(x_poly_test_scaled)
y_pred_train = best_model.predict(x_poly_train_scaled)
train_error = compute_error(y_pred_train, y_train)
test_error = compute_error(y_pred_test, y_test)
print("Train Error:", train_error)
print("Test Error:", test_error)

# Plot train, predicted, and y_ideal for the optimal polynomial degree
x_values = np.linspace(0, 49, 200)
x_poly_values = poly_features.transform(x_values.reshape(-1, 1))
x_poly_values_scaled = scaler.transform(x_poly_values)
y_pred_values = best_model.predict(x_poly_values_scaled)

plt.figure(figsize=(10, 6))
plt.scatter(x_train, y_train, label='Train', color='blue')
plt.plot(x_ideal, y_ideal, label='y_ideal', color='orange')
plt.plot(x_values, y_pred_values, label='Predicted', color='red')
plt.xlabel('X')
```

```
PS C:\Users\ardah\UCM> python -u "c:\Users\ardah\UCM\p6\p6_yeni.py"
Best Degree: 5
Best Alpha: 10
Train Error: 22869.839453485423
Test Error: 20325.61641589232
```



Model with Optimal Polynomial Degree and Regularization
Train
y_ideal
Predicted
Y
X
3000
2500
2000
1500
1000
500
0
10
20
30
40
50

Fig 1.6

- We are getting our training-error and cross-validation error

```
p6 > p6_yeni.py > ...
# Generate synthetic data
m_total = 1000
x_data, y_data, _, _ = gen_data(m_total)

# Split data into training and validation sets
X_train_val, X_test, y_train_val, y_test = train_test_split(x_data[:, None], y_data, test_size=0.4, random_state=1)
X_cv, X_test, y_cv, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state=1)

# Define subset sizes
num_steps = 12
m_range = np.array([m_total * i // num_steps for i in range(1, num_steps + 1)])

train_errors = []
val_errors = []

# Iterate over subset sizes
for i in range(num_steps):
    # Select a subset of the training data
    X_train_subset = X_train_val[:m_range[i], :]
    y_train_subset = y_train_val[:m_range[i]]

    # Transform data into polynomial features
    poly_features = PolynomialFeatures(degree=16, include_bias=False)
    X_poly_train = poly_features.fit_transform(X_train_subset)
    X_poly_val = poly_features.transform(X_cv)

    # Scale data
    scaler = StandardScaler()
    X_poly_train_scaled = scaler.fit_transform(X_poly_train)
    X_poly_val_scaled = scaler.transform(X_poly_val)

    # Train Linear Regression model
    model = LinearRegression()
    model.fit(X_poly_train_scaled, y_train_subset)

    # Predict on training and validation sets
    y_pred_train = model.predict(X_poly_train_scaled)
    y_pred_val = model.predict(X_poly_val_scaled)

    # Compute errors
    train_error = compute_error(y_pred_train, y_train_subset)
    val_error = compute_error(y_pred_val, y_cv)

    # Append errors to lists
    train_errors.append(train_error)
    val_errors.append(val_error)
```

```
PS C:\Users\ardah\UCM> python -u "c:\Users\ardah\UCM\p6\p6_yeni.py"
Best Degree: 5
Best Alpha: 10
Train Error: 22869.839453485423
Test Error: 20325.61641589232
PS C:\Users\ardah\UCM> python -u "c:\Users\ardah\UCM\p6\p6_yeni.py"
```