nonlinear-regression

February 21, 2025

0.1 IMPORTS

[93]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from numpy.linalg import pinv
from sklearn.model_selection import train_test_split
```

0.2 sinusoidal regressor

[94]:
```
class SinusoidalRegressor:
    def __init__(self):
        self.k = None
        self.weights = None

    def phi(self, x):
        """Generate basis functions for a given x"""
        return np.column_stack((np.ones_like(x := np.asarray(x)), np.sin(np.
 ↪outer(x, np.arange(1, self.k + 1))), np.cos(np.outer(x, np.arange(1, self.k␣
 ↪+ 1)))))

    def fit(self, X_train, Y_train, k):
        """Train the model using pseudo-inverse with optional regularization"""
        self.k = k
        X_train = X_train
        Y_train = Y_train

        Phi = self.phi(X_train)
        self.weights = pinv(Phi.T @ Phi) @ Phi.T @ Y_train

    def predict(self, X):
        """Make predictions"""
        if self.weights is None:
            raise ValueError("Model is not fitted yet.")
        X = X
        Phi = self.phi(X)
        return Phi @ self.weights
```

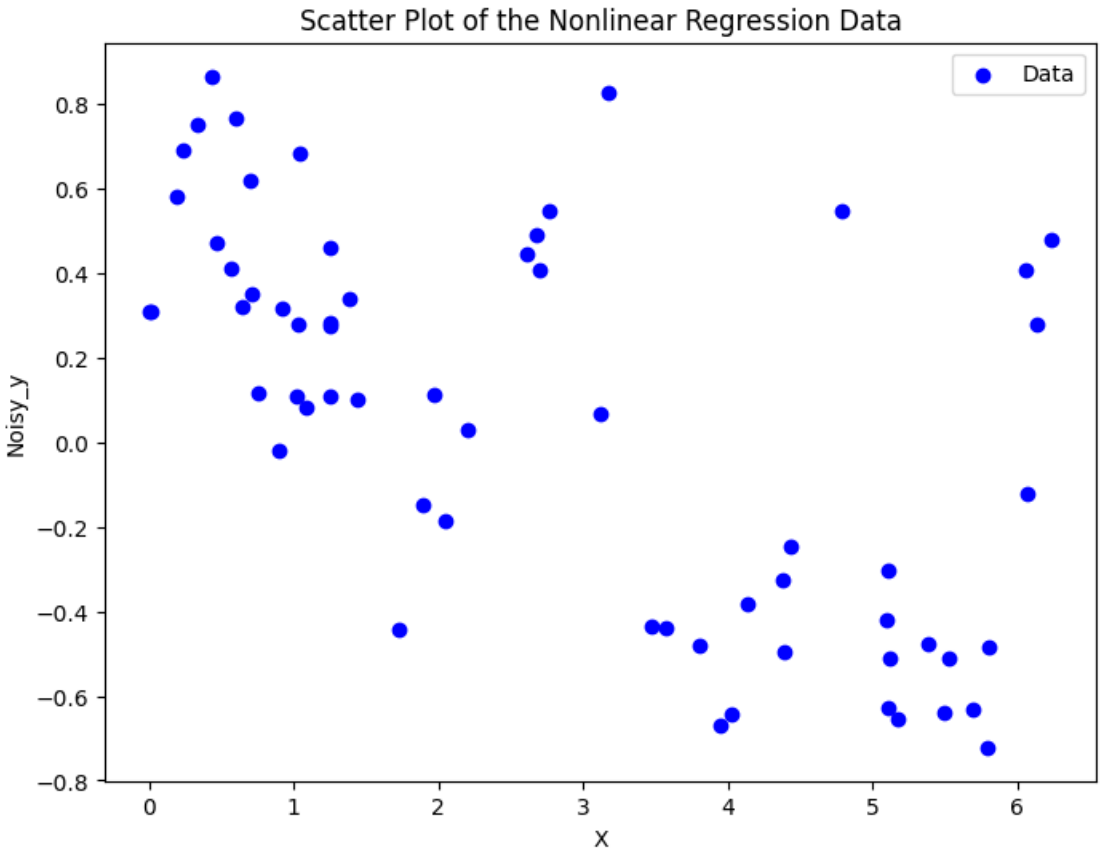
```
    def rmse(self, X_val, Y_val):
        """Compute RMSE"""
        y_pred = self.predict(X_val)
        return np.sqrt(np.mean((Y_val - y_pred) ** 2))
```

0.3 Loading the data and the model

[95]:
```
# Load Data
np.random.seed(61)
csv_file = 'nonlinear-regression-data.csv'
data = pd.read_csv(csv_file)
x = np.array(data['X'])
y = np.array(data['Noisy_y'])

# Split Data
X_train, X_val, Y_train, Y_val = train_test_split(x, y, test_size=16,␣
 ↪random_state=61)

# Initialize Model
sinusoidal_regressor = SinusoidalRegressor()
```

VISUALIZING THE DATASET We have an independent variable X and some dependent Noisy_Y that is associated with each value of X.

[96]:
```
# Plot the data
plt.figure(figsize=(8, 6))
plt.scatter(x, y, color='blue', label='Data')
plt.xlabel('X')
plt.ylabel('Noisy_y')
plt.title('Scatter Plot of the Nonlinear Regression Data')
plt.legend()
plt.show()

# size of the dataset
N = len(X_train)
print(f"Size of the dataset: {N}")
N = len(X_val)
print(f"Size of the dataset: {N}")
```



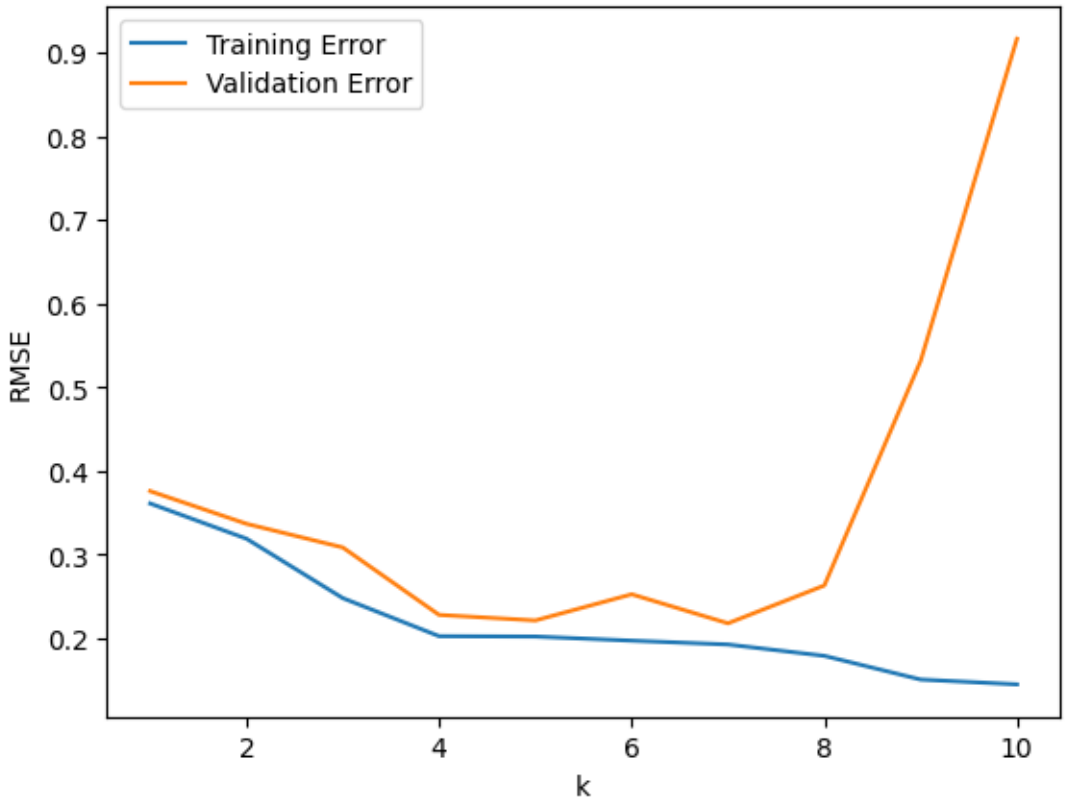

```
Size of the dataset: 45
Size of the dataset: 16
```

0.4 TRAINING THE MODEL

As K increases, we increase more sine and cosine basis functions used in the model, this makes the model more complex.

[97]:
```python
# Fit model and compute errors
train_errors = []
val_errors = []
k_values = range(1, 11)
for k in k_values:
    sinusoidal_regressor.fit(X_train, Y_train, k)
    train_errors.append(sinusoidal_regressor.rmse(X_train, Y_train))
    val_errors.append(sinusoidal_regressor.rmse(X_val, Y_val))

# Plot Training and Validation Errors
plt.plot(k_values, train_errors, label="Training Error")
plt.plot(k_values, val_errors, label="Validation Error")
```

```
plt.xlabel("k")
plt.ylabel("RMSE")
plt.legend()
plt.show()
```



[98]:
```
optimal_k_train = np.argmin(train_errors) + 1
print(f"The optimal k that minimizes the training data is: {optimal_k_train}")
```

```
The optimal k that minimizes the training data is: 10
```

[99]:
```
optimal_k_val = np.argmin(val_errors) + 1
print(f"The optimal k that minimizes the validation error is: {optimal_k_val}")
```

```
The optimal k that minimizes the validation error is: 7
```

0.4.1 Compare the optimal k values that you found for the training error and validation error. Explain why they are different or the same.

The optimal k that minimizes the training error is: 10

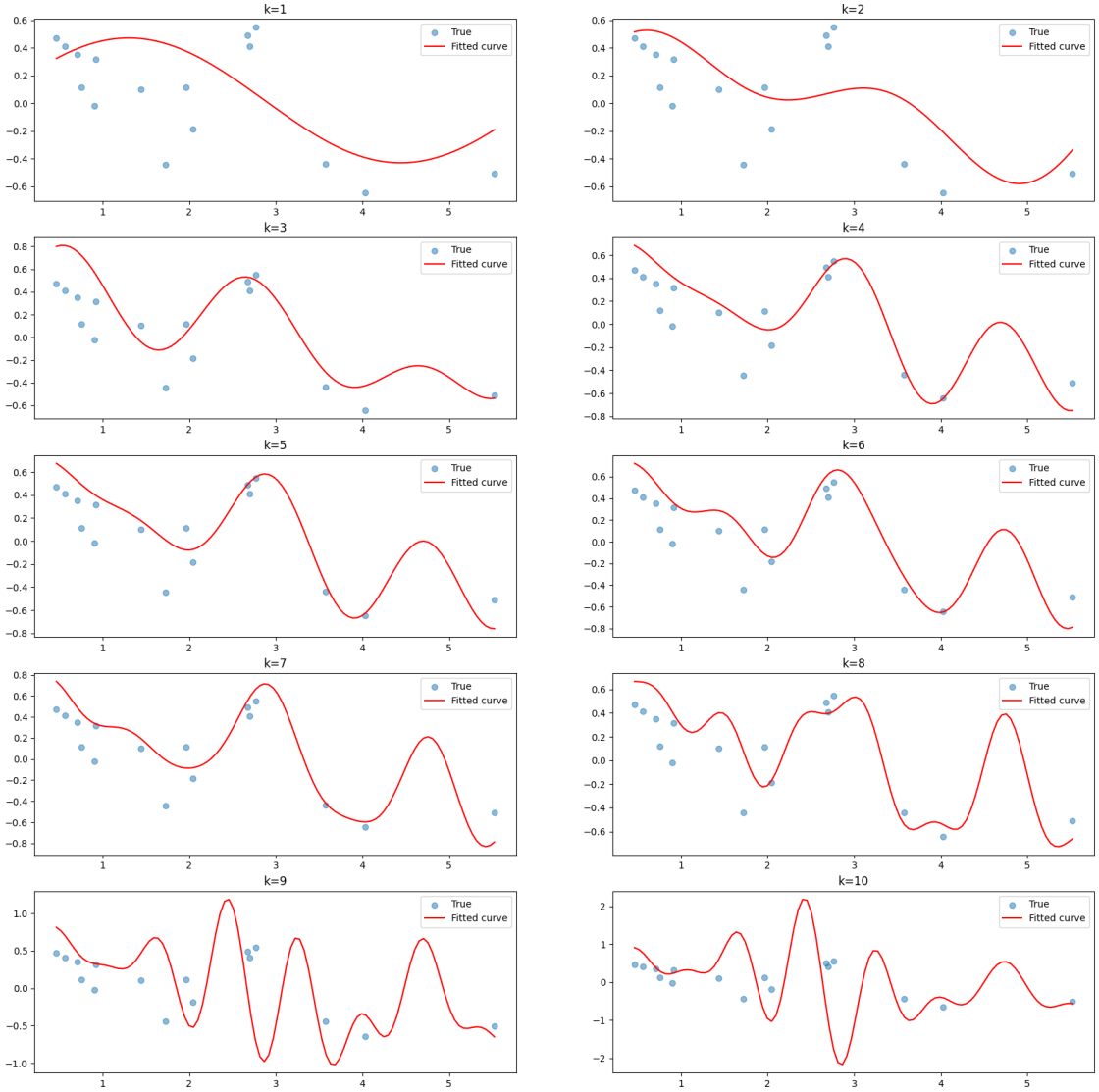
The optimal k that minimizes the validation error is: 7

The optimal k values for training and validation errors are different. This discrepancy occurs

because the training error measures how well the model fits the training data, while the validation error measures how well the model generalizes to unseen data. A lower k value might fit the training data very well (low training error) but may not generalize well to new data (high validation error). Conversely, a higher k value might provide a better balance between fitting the training data and generalizing to new data, resulting in a lower validation error. This phenomenon is known as overfitting and underfitting. The optimal k for validation error is typically preferred as it indicates better generalization performance.

0.5 MODEL PREDICTIONS

[100]:
```
# Plot Predictions for Different k Values
fig, axs = plt.subplots(5, 2, figsize=(20, 20))
x_line = np.linspace(X_val.min(), X_val.max(), 100)
for k in k_values:
    sinusoidal_regressor.fit(X_train, Y_train, k)
    y_line = sinusoidal_regressor.predict(x_line)

    ax = axs[(k - 1) // 2, (k - 1) % 2]
    ax.scatter(X_val, Y_val, label="True", alpha=0.5)
    ax.plot(x_line, y_line, color='r', label="Fitted curve")
    ax.set_title(f"k={k}")
    ax.legend()
plt.show()
```



0.5.1 For each k = 1, 3, 5, 10, create a scatter plot of the validation data points. On the same plot, for each k, draw a line for the fitted polynomial.

[101]:
```
# Plot Predictions for k = 1, 3, 5, 10
k_values_to_plot = [1, 3, 5, 10]
fig, axs = plt.subplots(2, 2, figsize=(15, 10))

for idx, k in enumerate(k_values_to_plot):
    sinusoidal_regressor.fit(X_train, Y_train, k)
    y_line = sinusoidal_regressor.predict(x_line)

    ax = axs[idx // 2, idx % 2]
    ax.scatter(X_val, Y_val, label="Validation Data", alpha=0.5)
```

```
    ax.plot(x_line, y_line, color='r', label=f"Fitted curve (k={k})")
    ax.set_title(f"k={k}")
    ax.legend()

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