5241Assignment1

Francis Zhang xz3279

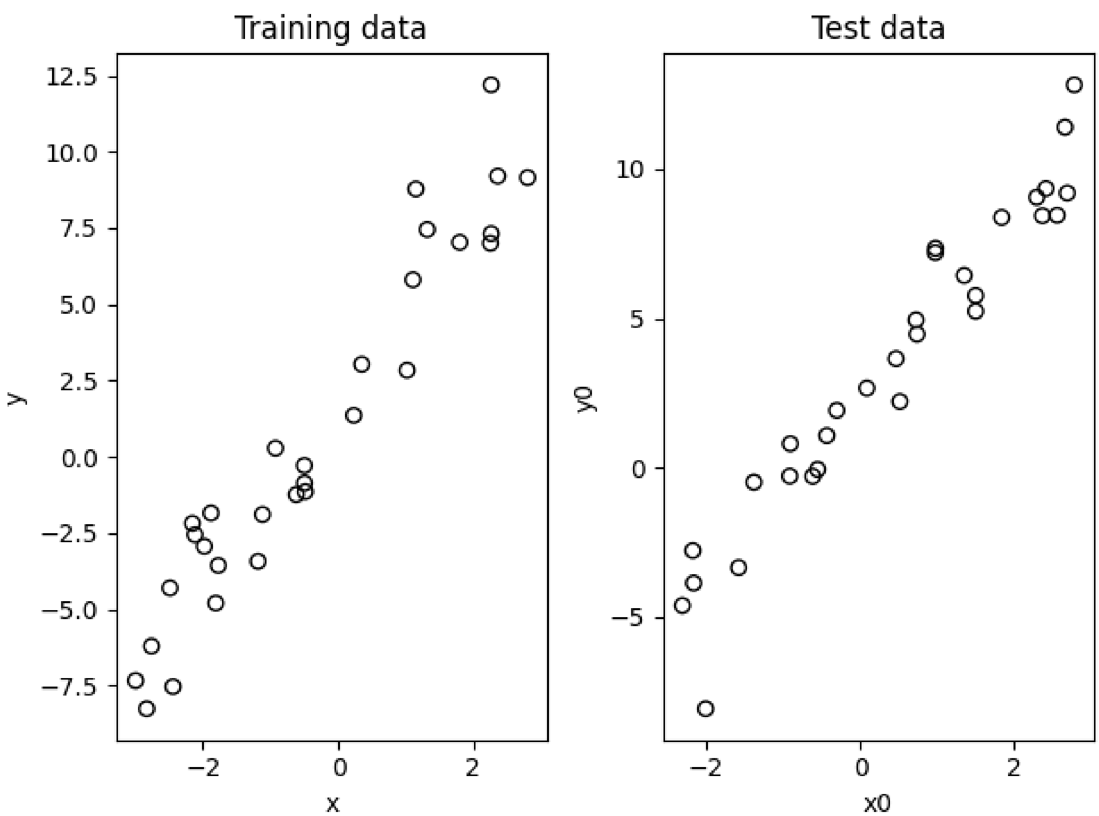
February 7, 2024

Problem 1

We simulate in a simple way. Use $y = 3x + 2 + noise$. And then see the MSEs for both training set and test set.
1. More flexibility $\Longrightarrow$ fits more datas. So the MSEs should be decresing here.
2. In general, more flexibility means less $bias^2$ and more variance. Increasing flexibility at first lowers total error a lot. However, as $bias^2$ is apporaching 0, but variance can be much more and more. So total error of training set is decreasing at first to a optimal value, and then increases. Which is **Bias-Variance Tradeoff**. Simulations are attached below. In my perspective, I'm not sure about the MSE here is in which part of the U-curve(**Bias-Variance Tradeoff**). We begin our simulation below for both **1.** and **2.**. (The rest of my "text" answer is on page 9.)

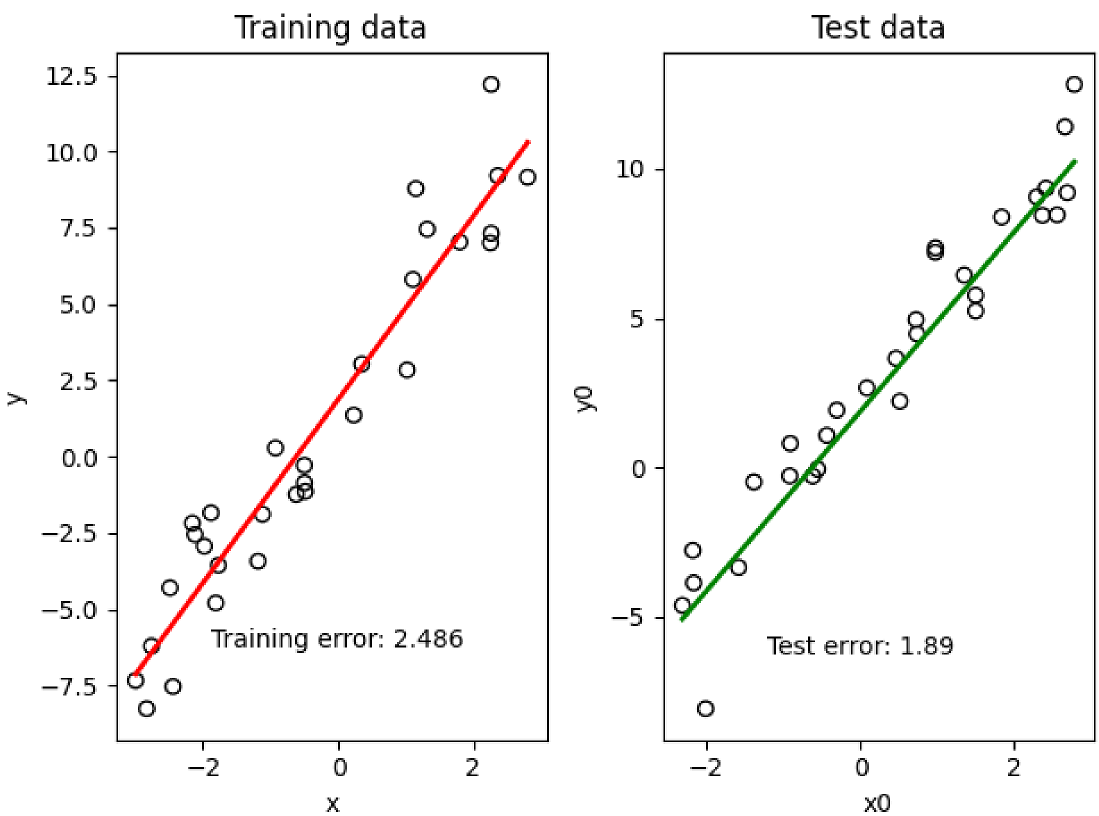
[63]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

np.random.seed(1)  # Set seed for reproducibility

n = 30  # Number of observations

# Generate training data
x = np.sort(np.random.uniform(-3, 3, n))
# Linear Function: y = 2 + 3x + noise
y = 2 + 3 * x + 2 * np.random.randn(n) #adding random noise, times 2 to make it␣
↪more noisier

# Generate test data
x0 = np.sort(np.random.uniform(-3, 3, n))
y0 = 2 + 3 * x0 + 2 * np.random.randn(n)

# Set up a 1x2 layout for plotting
plt.subplot(1, 2, 1)
plt.scatter(x, y, marker='o', facecolors='none', edgecolors='black')  # Hollow␣
↪black circles
plt.title("Training data")
```

```
plt.xlabel("x")
plt.ylabel("y")

plt.subplot(1, 2, 2)
plt.scatter(x0, y0, marker='o', facecolors='none', edgecolors='black')  # Hollow␣
↪black circles
plt.title("Test data")
plt.xlabel("x0")
plt.ylabel("y0")

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



[85]:
```
# Fit linear regression model to training data
lm_1 = LinearRegression()
lm_1.fit(x.reshape(-1, 1), y)

# Predictions on training data
yhat_1 = lm_1.predict(x.reshape(-1, 1))
train_err_1 = np.mean((y - yhat_1)**2)
```

```python
# Predictions on test data
y0hat_1 = lm_1.predict(x0.reshape(-1, 1))
test_err_1 = np.mean((y0 - y0hat_1)**2)

# Set up a 1x2 layout for plotting
plt.subplot(1, 2, 1)
plt.scatter(x, y, marker='o', facecolors='none', edgecolors='black') # Scatter plot of training data
plt.plot(x, yhat_1, color='red', linewidth=2)  # Regression line
plt.title("Training data")
plt.xlabel("x")
plt.ylabel("y")
plt.text(0, -6, f"Training error: {round(train_err_1, 3)}", ha='center', va='center', color='black')

plt.subplot(1, 2, 2)
plt.scatter(x0, y0, marker='o', facecolors='none', edgecolors='black') # Scatter plot of test data
plt.plot(x0, y0hat_1, color='green', linewidth=2)  # Regression line
plt.title("Test data")
plt.xlabel("x0")
plt.ylabel("y0")
plt.text(0, -6, f"Test error: {round(test_err_1, 3)}", ha='center', va='center', color='black')

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



[86]:
```
lm_1.intercept_
```

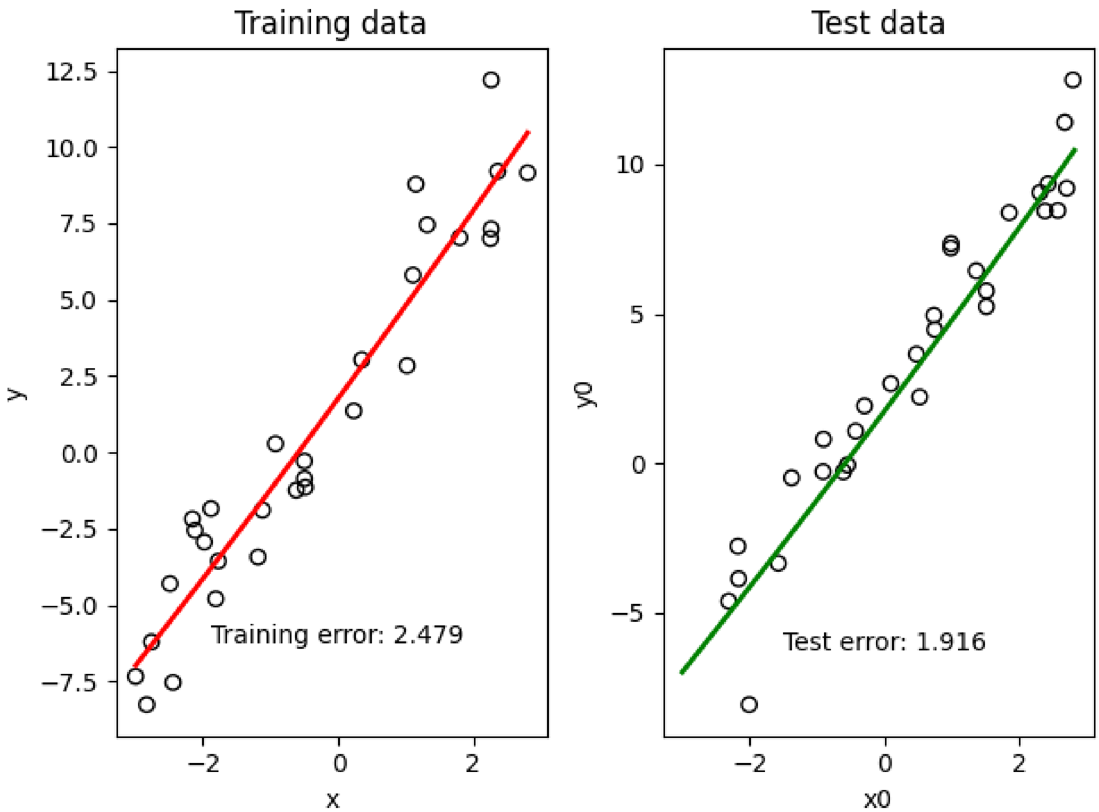
[86]: 1.8624318882799002

[87]:
```
lm_1.coef_
```

[87]: array([2.99977939])

[88]:
```
# Fit 2th order polynomial regression model to training data
lm_2 = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())
lm_2.fit(x.reshape(-1, 1), y)

# Predictions on training data
yhat_2 = lm_2.predict(x.reshape(-1, 1))
train_err_2 = np.mean((y - yhat_2)**2)

# Predictions on test data
y0hat_2 = lm_2.predict(x0.reshape(-1, 1))
test_err_2 = np.mean((y0 - y0hat_2)**2)

# Set up a 1x2 layout for plotting
```

```python
plt.subplot(1, 2, 1)
plt.scatter(x, y, marker='o', facecolors='none', edgecolors='black')  # Scatter plot of training data
xx = np.linspace(min(x), max(x), 100)
plt.plot(xx, lm_2.predict(xx.reshape(-1, 1)), color='red', linewidth=2)  # Regression line
plt.title("Training data")
plt.xlabel("x")
plt.ylabel("y")
plt.text(0, -6, f"Training error: {round(train_err_2, 3)}", ha='center', va='center', color='black')

plt.subplot(1, 2, 2)
plt.scatter(x0, y0, marker='o', facecolors='none', edgecolors='black')  # Scatter plot of test data
plt.plot(xx, lm_2.predict(xx.reshape(-1, 1)), color='green', linewidth=2)  # Regression line
plt.title("Test data")
plt.xlabel("x0")
plt.ylabel("y0")
plt.text(0, -6, f"Test error: {round(test_err_2, 3)}", ha='center', va='center', color='black')

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



[89]:
```
lm_2.named_steps['linearregression'].intercept_
```

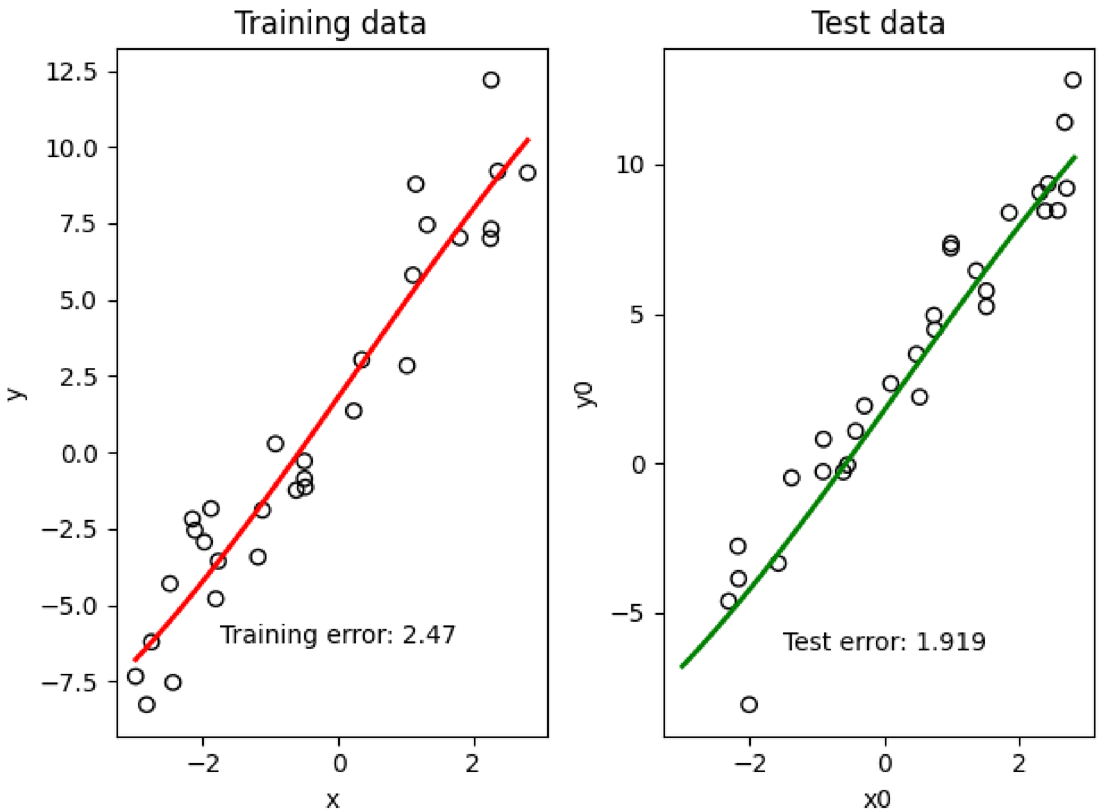
[89]: 1.7572092032211697

[90]:
```
lm_2.named_steps['linearregression'].coef_
```

[90]: array([0.   , 3.00921458, 0.03195098])

[91]:
```
# Fit 3th order polynomial regression model to training data
lm_3 = make_pipeline(PolynomialFeatures(degree=3), LinearRegression())
lm_3.fit(x.reshape(-1, 1), y)

# Predictions on training data
yhat_3 = lm_3.predict(x.reshape(-1, 1))
train_err_3 = np.mean((y - yhat_3)**2)

# Predictions on test data
y0hat_3 = lm_3.predict(x0.reshape(-1, 1))
test_err_3 = np.mean((y0 - y0hat_3)**2)

# Set up a 1x2 layout for plotting
```

```python
plt.subplot(1, 2, 1)
plt.scatter(x, y, marker='o', facecolors='none', edgecolors='black')  # Scatter plot of training data
xx = np.linspace(min(x), max(x), 100)
plt.plot(xx, lm_3.predict(xx.reshape(-1, 1)), color='red', linewidth=2)  # Regression line
plt.title("Training data")
plt.xlabel("x")
plt.ylabel("y")
plt.text(0, -6, f"Training error: {round(train_err_3, 3)}", ha='center', va='center', color='black')

plt.subplot(1, 2, 2)
plt.scatter(x0, y0, marker='o', facecolors='none', edgecolors='black')  # Scatter plot of test data
plt.plot(xx, lm_3.predict(xx.reshape(-1, 1)), color='green', linewidth=2)  # Regression line
plt.title("Test data")
plt.xlabel("x0")
plt.ylabel("y0")
plt.text(0, -6, f"Test error: {round(test_err_3, 3)}", ha='center', va='center', color='black')

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



[92]:
```
lm_3.named_steps['linearregression'].intercept_
```

[92]: 1.7961030966373994

[93]:
```
lm_3.named_steps['linearregression'].coef_
```

[93]:
```
array([ 0.        ,  3.14176142,  0.01900498, -0.02478674])
```

We can see that when the flexibility increases,$MSE_{training}$ decreases and MSE_{test} increases.

3.
(a) The pattern below(on page10) should be the same. As we only have 1 point here. It does not change for increasing the flexibility.
(b) The codes begin on page 11, the graph is on page 12.

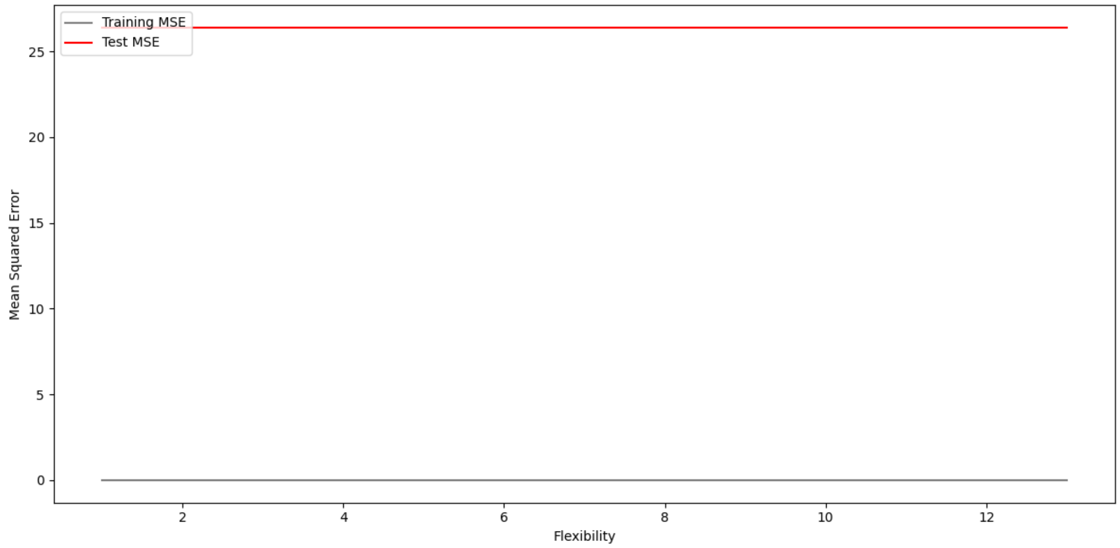
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

np.random.seed(1)  # Set seed for reproducibility

n = 1  # Number of observations

# Generate training data
x = np.sort(np.random.uniform(-3, 3, n))
# Linear Function: y = 2 + 3x + noise
y = 2 + 3 * x + 2 * np.random.randn(n) #adding random noise, times 2 to make it␣
 ↪more noisier

# Generate test data
x0 = np.sort(np.random.uniform(-3, 3, n))
y0 = 2 + 3 * x0 + 2 * np.random.randn(n)

# Define the degrees of polynomial features
degrees = range(1, 14)

# Initialize lists to store training and test errors
train_errors = []
test_errors = []

# Iterate over different degrees
for degree in degrees:
    # Fit model
    lm = make_pipeline(PolynomialFeatures(degree=degree), LinearRegression())
    lm.fit(x.reshape(-1, 1), y)

    # Predictions on training data
    yhat_train = lm.predict(x.reshape(-1, 1))
    train_err = np.mean((y - yhat_train)**2)

    # Predictions on test data
    yhat_test = lm.predict(x0.reshape(-1, 1))
    test_err = np.mean((y0 - yhat_test)**2)
```

```
    # Append errors to lists
    train_errors.append(train_err)
    test_errors.append(test_err)

# Print errors
for degree, train_err, test_err in zip(degrees, train_errors, test_errors):
    print(f"Degree {degree}: Training error: {train_err}, Test error:␣
 ↪{test_err}")
```

```
Degree 1: Training error: 0.0, Test error: 26.396077576339028
Degree 2: Training error: 0.0, Test error: 26.396077576339028
Degree 3: Training error: 0.0, Test error: 26.396077576339028
Degree 4: Training error: 0.0, Test error: 26.396077576339028
Degree 5: Training error: 0.0, Test error: 26.396077576339028
Degree 6: Training error: 0.0, Test error: 26.396077576339028
Degree 7: Training error: 0.0, Test error: 26.396077576339028
Degree 8: Training error: 0.0, Test error: 26.396077576339028
Degree 9: Training error: 0.0, Test error: 26.396077576339028
Degree 10: Training error: 0.0, Test error: 26.396077576339028
Degree 11: Training error: 0.0, Test error: 26.396077576339028
Degree 12: Training error: 0.0, Test error: 26.396077576339028
Degree 13: Training error: 0.0, Test error: 26.396077576339028
```

[2]:

```
# Plot
plt.figure(figsize=(12, 6))
plt.xlabel('Flexibility')
plt.ylabel('Mean Squared Error')
plt.plot(degrees, train_errors, color='grey', label= 'Training MSE')
plt.plot(degrees, test_errors, color='red',  label= 'Test MSE')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```

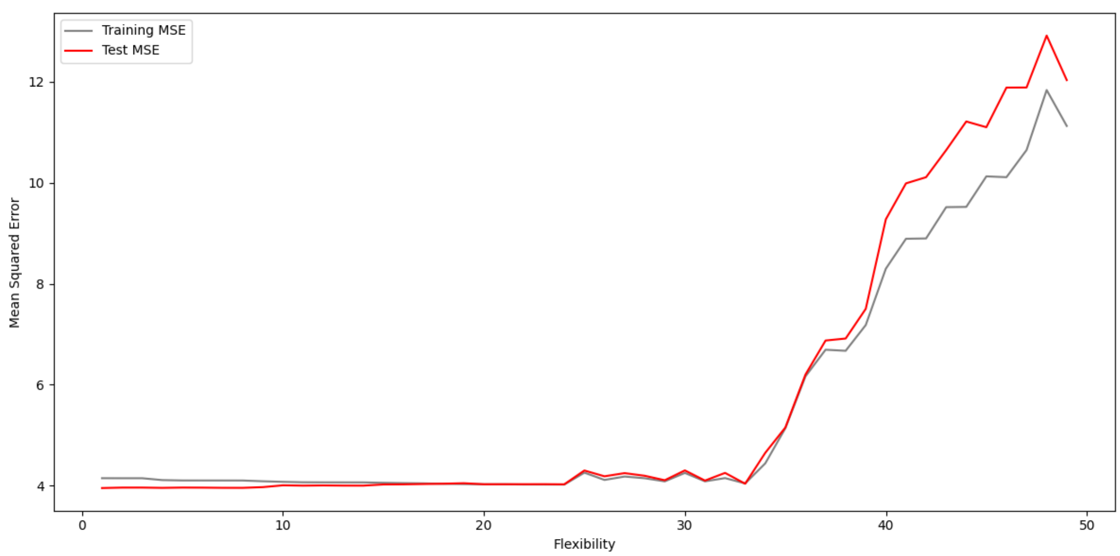


[13]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

np.random.seed(1)  # Set seed for reproducibility

n = 1000  # Number of observations

# Generate training data
x = np.sort(np.random.uniform(-3, 3, n))
# Linear Function: y = 2 + 3x + noise
y = 2 + 3 * x + 2 * np.random.randn(n) #adding random noise, times 2 to make it␣
↪more noisier

# Generate test data
x0 = np.sort(np.random.uniform(-3, 3, n))
y0 = 2 + 3 * x0 + 2 * np.random.randn(n)

# Define the degrees of polynomial features
degrees = range(1, 50)

# Initialize lists to store training and test errors
train_errors = []
test_errors = []

# Iterate over different degrees
```

```
for degree in degrees:
    # Fit model
    lm = make_pipeline(PolynomialFeatures(degree=degree), LinearRegression())
    lm.fit(x.reshape(-1, 1), y)

    # Predictions on training data
    yhat_train = lm.predict(x.reshape(-1, 1))
    train_err = np.mean((y - yhat_train)**2)

    # Predictions on test data
    yhat_test = lm.predict(x0.reshape(-1, 1))
    test_err = np.mean((y0 - yhat_test)**2)

    # Append errors to lists
    train_errors.append(train_err)
    test_errors.append(test_err)
```

[14]:

```
# Plot
plt.figure(figsize=(12, 6))
plt.xlabel('Flexibility')
plt.ylabel('Mean Squared Error')
plt.plot(degrees, train_errors, color='grey', label= 'Training MSE')
plt.plot(degrees, test_errors, color='red',  label= 'Test MSE')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```



Problem 2

1.
As $p(x|\mu,\nu) := (\frac{\nu}{\mu})^{\nu}\frac{x^{\nu-1}}{\Gamma(\nu)}\exp\Big(-\frac{\nu x}{\mu}\Big)$, $p(x|\theta) = (\frac{\nu}{\mu})^{\nu}\frac{x^{\nu-1}}{\Gamma(\nu)}\exp\Big(-\frac{\nu x}{\mu}\Big)$.
$L(\theta) = \prod_{i=1}^{n} p(x_i|\theta) = (\frac{\nu}{\mu})^{n\nu}\frac{\prod_{i=1}^{n} x_i^{\nu-1}}{\Gamma^{(n)}(\nu)}\exp\Big(-\sum_{i=1}^{n}\frac{\nu x_i}{\mu}\Big)$
Suppose $l(\theta)$ to be log-likelihood.
$l(\theta) = \log L(\theta) = n\nu(\log\nu - \log\mu) + (\nu-1)\sum_{i=1}^{n}\log x_i - n\log\Gamma(\nu) - \frac{\nu}{\mu}\sum_{i=1}^{n} x_i$
2.
$\frac{\partial l}{\partial\mu} = -\frac{1}{\mu}n\nu + \frac{\nu}{\mu^2}\sum_{i=1}^{n} x_i$
When $\frac{\partial l}{\partial\mu} = 0 \implies \frac{\sum x_i}{\mu} = n \implies \hat{\mu} = \frac{\sum x_i}{n} = \overline{x}$
Checking the mean and variance and MLE, from what we've learned from Gaussian, $\hat{\mu} = \overline{x}$ is not a surprise.
3.
$\frac{\partial l}{\partial\nu} = n(\log\nu - \log\mu) + n - n\frac{\Gamma'(\nu)}{\Gamma(\nu)} + \sum_{i=1}^{n}\log x_i - \frac{\sum x_i}{\mu}$
When $\frac{\partial l}{\partial\nu} = 0 \implies n(\log\nu - \log\mu) + n - n\frac{\Gamma'(\nu)}{\Gamma(\nu)} + \sum_{i=1}^{n}\log x_i - n = n(\log\nu - \log\mu) + \sum_{i=1}^{n}\log x_i - n\frac{\Gamma'(\nu)}{\Gamma(\nu)} = 0$ where $n(\log\nu - \log\mu) + \sum_{i=1}^{n}\log x_i = \sum\log\Big(\frac{x_i\hat{\nu}}{\mu}\Big)$, $-n\frac{\Gamma'(\nu)}{\Gamma(\nu)} = -\sum\phi(\hat{\nu})$.
So, $\sum_{i=1}^{n}(\log\Big(\frac{x_i\hat{\nu}}{\mu}\Big) - (\frac{x_i}{\mu} - 1) - \phi(\hat{\nu})) = n(\log\nu - \log\mu) + \sum_{i=1}^{n}\log x_i - \frac{\sum x_i}{\mu} + n - n\frac{\Gamma'(\nu)}{\Gamma(\nu)} = 0$

Problem 3

Our target is to get $\min R(f)$ from the Bayes-Optimal Classifier f_0, due to the monotonicity of the integral $R(f) = \int_{\mathbb{R}^d} R(f|\mathbf{x})p(\mathbf{x})d\mathbf{x}$, we can say f_0 minimizes $R(f)$ by minimizing $R(f|\mathbf{x}) := \sum_{y\in[K]} L^{0-1}(y, f(\mathbf{x}))P(y|\mathbf{x})$. To find the f minimizes $R(f|\mathbf{x})$:

$$\begin{aligned}
f(x) &= \arg\min_{y\in[K]}\sum_{y\in[K]} L^{0-1}(y|f(\mathbf{x}))P(y|\mathbf{x}) \\
&= \arg\min_{y\in[K]}(1 - P(y|\mathbf{x})) \\
&\quad\text{(mismatch loss is 1 and math loss 0} \\
&\quad\implies 0\cdot p(match) + 1\cdot p(mismatch) = 1\cdot(1 - p(match)) = 1 - p(y|x))
\end{aligned}$$

So in this case, $f(x) = \arg\max_{y\in[K]} P(y|\mathbf{x})$, this is exactly $f_0(\mathbf{x}) = \arg\max_{y\in[K]} P(y|\mathbf{x})$. Hence f_0 is what we want here.

Problem 4

1. $posterior \propto likelihood * prior$
As $L(\theta_1,...,\theta_K|x_1,...,x_n) = \prod\theta^{n_k}$ and $q(\theta_1,...,\theta_K) = \frac{1}{B(\alpha_1,...,\alpha_K)}\prod\theta^{\alpha_k-1}$ where $B(\alpha_1,...,\alpha_K) = \frac{\prod\Gamma(\alpha_k)}{\Gamma(\sum\alpha_k)}$, so $\Pi(\theta_1,...,\theta_K|x_1,...,x_n) = \prod\theta^{n_k}\cdot\frac{\Gamma(\sum\alpha_k)}{\prod\Gamma(\alpha_k)}\prod\theta^{\alpha_k-1} = \frac{\Gamma(\sum_{k=1}^{K}\alpha_k)}{\prod_{k=1}^{K}\Gamma(\alpha_k)}\prod_{k=1}^{K}\theta^{\alpha_k+n_k-1}$

2.

$$\begin{aligned}
\hat{\theta_{MAP}} &= \arg\max \frac{\Gamma(\sum_{k=1}^{K}\alpha_k)}{\prod_{k=1}^{K}\Gamma(\alpha_k)} \prod_{k=1}^{K} \theta^{\alpha_k+n_k-1} \\
&= \arg\max \prod_{k=1}^{K} \theta^{\alpha_k+n_k-1} \\
&= \arg\max \log \prod_{k=1}^{K} \theta^{\alpha_k+n_k-1} = \arg\max \sum_{k=1}^{K} \log \theta^{\alpha_k+n_k-1} \\
&= \arg\max \sum_{k=1}^{K} (n_i+\alpha_i-1)\log\theta_i
\end{aligned}$$

$$\begin{aligned}
0 &= \frac{\partial}{\partial\theta_i}(\sum_{k=1}^{K} \log \theta^{\alpha_k+n_k-1} + (n_k+\alpha_k-1)\log\left(1-\sum_{i=1}^{K-1}\theta_i\right)) \\
&= \frac{(n_i+\alpha_i-1)}{\theta_i} - \frac{(n_k+\alpha_k-1)}{1-\sum_{i=1}^{K-1}\theta_i} \\
&= \frac{(n_i+\alpha_i-1)}{\theta_i} - \frac{(n_k+\alpha_k-1)}{\theta_k}
\end{aligned}$$

That is $\hat{\theta}_i = \frac{n_i+\alpha_i-1}{\sum(n_i+\alpha_i-1)}$