M10-L1-P1

November 18, 2023

1 M10-L1 Problem 1

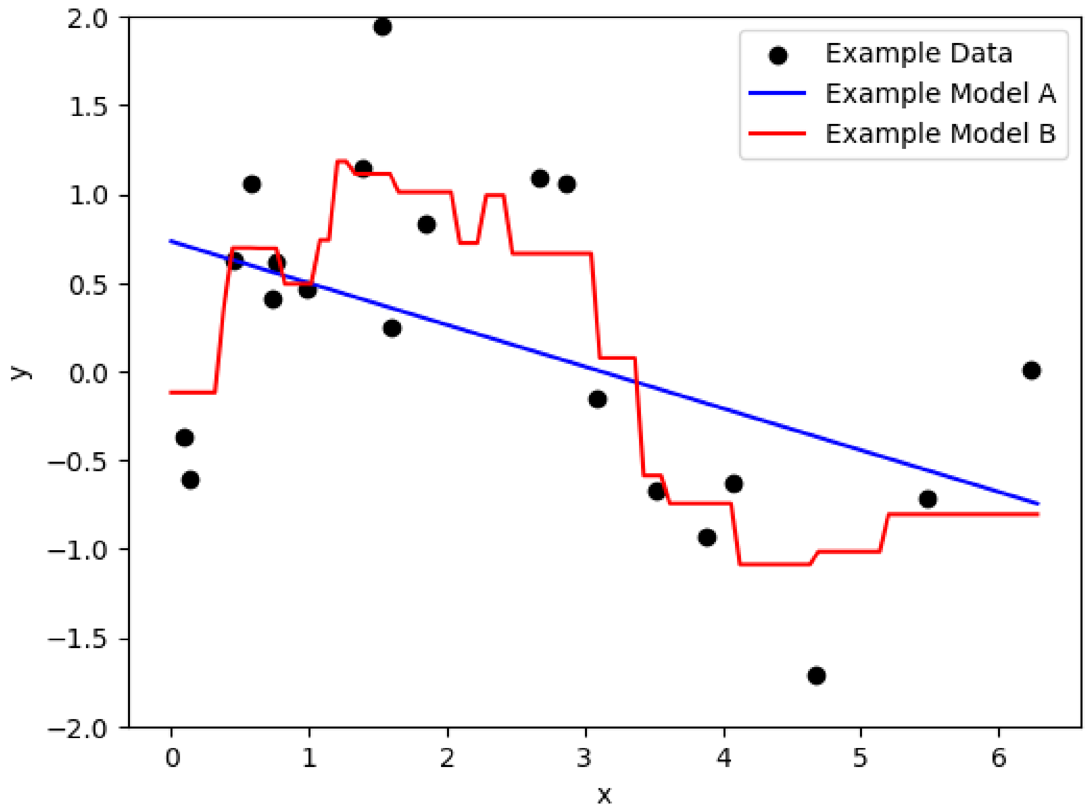
In this problem you will look compare models with lower/higher variance/bias by computing bias and variance at a single point.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor

def plot_model(model,color="blue"):
    x = np.linspace(0, np.pi*2, 100)
    y = model.predict(x.reshape(-1,1))
    plt.plot(x, y, color=color)
    plt.xlabel("x")
    plt.ylabel("y")

def plot_data(x, y):
    plt.scatter(x,y,color="black")

def eval_model_at_point(model, x):
    return model.predict(np.array([[x]])).item()

def train_models():
    x = np.random.uniform(0,np.pi*2,20).reshape(-1,1)
    y = np.random.normal(np.sin(x),0.5).flatten()

    modelA = LinearRegression()
    modelB = KNeighborsRegressor(3)
    modelA.fit(x,y)
    modelB.fit(x,y)
    return modelA, modelB, x, y
```

The function `train_models` gets 20 new data points and trains two models on these data points. Model A is a linear regression model, while model B is a 3-nearest neighbor regressor.

[ ]:
```
modelA, modelB, x, y = train_models()
plt.figure()
```

```
plot_data(x,y)
plot_model(modelA,"blue")
plot_model(modelB,"red")
plt.legend(["Example Data", "Example Model A", "Example Model B"])
plt.ylim([-2,2])
plt.show()
```



1.1 Training models

First, train 50 instances of model A and 50 instances of model B. Store all 100 total models for use in the next few cells. Generate these models with the function: `modelA, modelB, x, y = train_models()`.

[ ]:
```
modelA, modelB, x, y = zip(*[train_models() for i in range(50)])
```

1.2 Bias and Variance

Now we will use the definitions of bias and variance to compute the bias and variance of each type of model. You will focus on the point x = 1.57 only. First, compute the prediction for each model at x. (You can use the function `eval_model_at_point(model, x)`).

[ ]:
```
x = 1.57

x_A = [eval_model_at_point(model, x) for model in modelA]
x_B = [eval_model_at_point(model, x) for model in modelB]
```

In this cell, use the values you computed above to compute and print the bias and variance of model A at the point x = 1.57. The true function value `y_GT` is given as 1 for x=1.57.

[ ]:
```
yGT = 1

bias_A = np.mean(x_A) - yGT
var_A = np.mean(np.power(np.subtract(x_A, np.mean(x_A)), 2))

bias_B = np.mean(x_B) - yGT
var_B = np.mean(np.power(np.subtract(x_B, np.mean(x_B)), 2))

print(f"Model A:   Bias = {bias_A:.3f},  Variance = {var_A:.3f}")
print(f"Model B:   Bias = {bias_B:.3f},  Variance = {var_B:.3f}")
```

```
Model A:   Bias = -0.471,  Variance = 0.039
Model B:   Bias = 0.012,  Variance = 0.082
```

Questions

1. Which model has smaller bias at $x = 1.57$?

    `Model B has a smaller bias because it has a lower magnitude bias.`

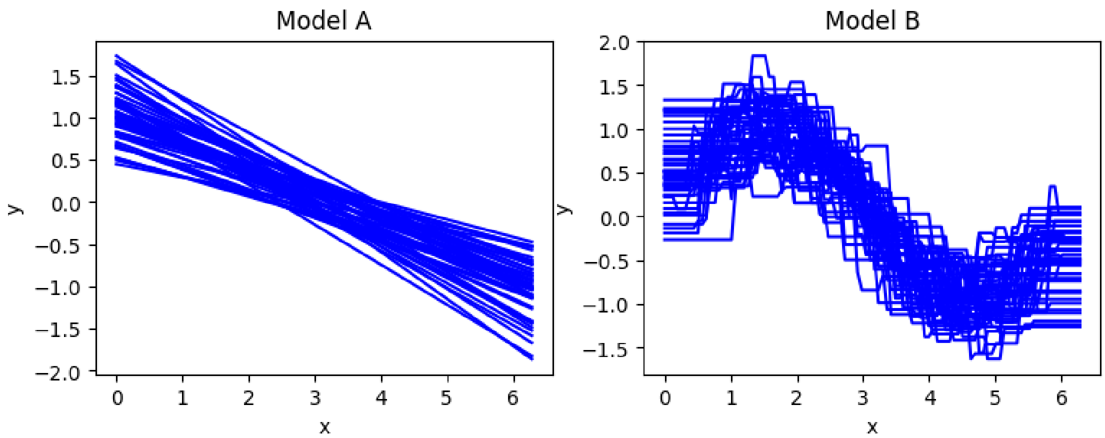
2. Which model has lower variance at $x = 1.57$?

    `Model A has the lower variance at this point.`

1.3 Plotting models

Now use the `plot_model` function to overlay all Model A predictions on one plot and all Model B predictions on another. Notice the spread of each model.

[ ]:
```
plt.figure(figsize=(9,3))

plt.subplot(1,2,1)
plt.title("Model A")
for model in modelA: plot_model(model, color="blue")

plt.subplot(1,2,2)
plt.title("Model B")
for model in modelB: plot_model(model, color="blue")

plt.show()
```


Model A
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
0 1 2 3 4 5 6
x
Model B
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
y
0 1 2 3 4 5 6
x