Practice Exercises for Module 7.

Problem 1 (Validation Set): Consider the following dataset with $N = 6$ and $p = 1$:

$$\mathcal{D} = \{(1, 1.2), (2, 1.8), (3, 3.5), (4, 4.7), (5, 4.7), (6, 5.4)\}$$

We would like to compare the performance of the following three models (feel free to use software to learn the parameters of each model; there is no need to do the computations by hand) which ones of the models would you use in practice?

$$\mathcal{M}_0 : \widehat{Y}_0 = \widehat{\alpha}_0$$
$$\mathcal{M}_1 : \widehat{Y}_1 = \widehat{\beta}_0 + \widehat{\beta}_1 X$$
$$\mathcal{M}_2 : \widehat{Y}_2 = \gamma_0 + \gamma_1 X + \gamma_2 X^2$$

a) Use the validation set approach to estimate the testing error of the three models above assuming that the data set was randomly split into the following training and validation subsets :

$$\mathcal{D}_{Train} = \{(1, 1.2), (3, 3.5), (4, 4.7)\}$$
$$\mathcal{D}_{Test} = \{(2, 1.8), (5, 4.7), (6, 5.4)\}$$

Solution code for learning parameters : For this problem, we will calculate the parameters of the models using the code given below :

```
Data = pd.DataFrame(data = [[1,1.2,2,1.8],[3,3.5,5,4.7],[4,4.7,6,5.4]] , columns =
    ["X_tr","Y_tr","X_te","Y_te"])

mse_train = []
mse_test = []

for d in range(0,3):
    poly_features = PolynomialFeatures(degree = d) #set the desired degree
    poly_model = LinearRegression(fit_intercept=True) #use linear regression

    x_train_poly = poly_features.fit_transform(Data[["X_tr"]]) #transform training
     data
    poly_model.fit(x_train_poly, Data[["Y_tr"]]) #fit model with training data
    only
    y_train_pred = poly_model.predict(x_train_poly) #predict using training data
    mse_train.append(mean_squared_error(Data[["Y_tr"]], y_train_pred)) #training
    MSE

    x_test_poly = poly_features.fit_transform(Data[["X_te"]]) #transform test data
    y_test_pred = poly_model.predict(x_test_poly) #predict using test data
    mse_test.append(mean_squared_error(Data[["Y_te"]], y_test_pred)) #test MSE

    print("The coefficients for the model are :", poly_model.coef_[:,1:])
    print("The intercept for the model is :",poly_model.intercept_[0])

plt.subplot(2,1,1)
plt.plot(range(0,3), mse_train, '-r')
```

```
plt.ylabel("Training MSE")
plt.subplot(2,1,2)
plt.plot(range(0,3), mse_test, '-b')
plt.xlabel("Flexibility (degree of polynomial)")
plt.ylabel("Test MSE")
plt.show()

print(mse_test)
print(mse_train)
```

Hand Calculations for MSE We will now calculate the MSE losses using the models that we trained uding the above code.
From the above code, the models with learned parameters are:

$$\mathcal{M}_0 \colon \widehat{Y}_0 = 3.1333$$
$$\mathcal{M}_1 \colon \widehat{Y}_1 = 0.02857 + 1.16428571 \cdot X$$
$$\mathcal{M}_2 \colon \widehat{Y}_2 = 0.1 + 1.08333333 \cdot X + 0.01666667 \cdot X^2$$

Therefore the predictions for the M_0 are,

$$\mathcal{D}_{train}(x_i, y_i) = \{(1, 3.1333), (3, 3.1333), (4, 3.1333)\}$$
$$\mathcal{D}_{test}(x_i, y_i) = \{(2, 3.1333), (5, 3.1333), (6, 3.1333)\}$$

The predictions for the M_1 are,

$$\mathcal{D}_{train}(x_i, y_i = \{(1, 1.1929), (3, 3.5214), (4, 4.6857)\}$$
$$\mathcal{D}_{test}(x_i, y_i) = \{(2, 2.3571), (5, 5.85), (6, 7.0143)\}$$

The predictions for the M_2 are,

$$\mathcal{D}_{train}(x_i, y_i = \{(1, 1.2), (3, 3.5), (4, 4.7)\}$$
$$\mathcal{D}_{test}(x_i, y_i) = \{(2, 2.3333), (5, 5.9333), (6, 7.2)\}$$

Now,

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Therefore, test MSE for the above models is

$$\text{Test MSE for } M_0 = \frac{1}{3}((3.1333 - 1.8)^2 + (3.1333 - 4.7)^2 + (3.1333 - 5.4)^2) = 3.1233$$
$$\text{Test MSE for } M_1 = \frac{1}{3}((2.3571 - 1.8)^2 + (5.85 - 4.7)^2 + (7.0143 - 5.4)^2) = 1.4129$$
$$\text{Test MSE for } M_2 = \frac{1}{3}((2.3333 - 1.8)^2 + (5.9333 - 4.7)^2 + (7.2 - 5.4)^2) = 1.6819$$

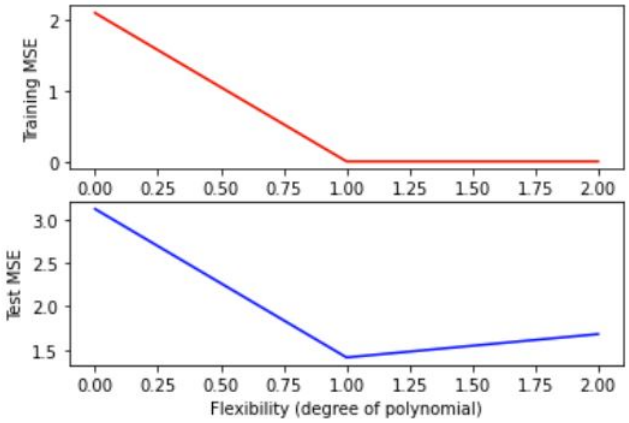
b) Compare the training and the testing errors for the three models above. Is the Training Error decreasing with the flexibility of your model?

Similarly, the train MSE for the above models is

$$\text{Train MSE for } M_0 = \frac{1}{3}((3.1333-1.2)^2 + (3.1333-3.5)^2 + (3.1333-4.7)^2) = 2.1088$$

$$\text{Train MSE for } M_1 = \frac{1}{3}((3.1333-1.2)^2 + (3.1333-3.5)^2 + (3.1333-4.7)^2) = 0.00024$$

$$\text{Train MSE for } M_2 = \frac{1}{3}((3.1333-1.2)^2 + (3.1333-3.5)^2 + (3.1333-4.7)^2) = 2.6952 * 10^{-30}$$



As we can see, the training error is decreasing with flexibility of the model

c) Based on this result, which ones of the models would you use in practice?

Based on the result above, we can see that the test error is lowest for M_1 i.e the optimal degree of the polynomial is 1. Hence, we will use M_1

Problem 2 (k-Fold Cross-Validation): Consider the following dataset with $N = 9$ and $p = 1$:

$$\mathcal{D} = \{(1, 1.2), (2, 1.8), (3, 3.5), (4, 4.7), (5, 4.7), (6, 5.4), (7, 7.1), (8, 7.9), (9, 8.1)\}$$

We would like to compare the performance of the following three models (feel free to use software to learn the parameters of each model; there is no need to do the computations by hand):

$$\mathcal{M}_0 : \widehat{Y}_0 = \widehat{\alpha}_0$$
$$\mathcal{M}_1 : \widehat{Y}_1 = \widehat{\beta}_0 + \widehat{\beta}_1 X$$
$$\mathcal{M}_1 : \widehat{Y}_1 = \gamma_0 + \gamma_1 X + \gamma_2 X^2$$

a) Use the 3-fold cross-validation to estimate the testing error of the three models above assuming that the data set was randomly split into the following three subsets (feel free to use software to train your models):

$$\mathcal{D}_1 = \{(3, 3.5), (5, 4.7), (8, 7.9)\}$$
$$\mathcal{D}_2 = \{(1, 1.2), (4, 4.7), (7, 7.1)\}$$
$$\mathcal{D}_3 = \{(2, 1.8), (6, 5.4), (9, 8.1)\}$$

Solution code for learning parameters :

```
D1 = pd.DataFrame(data = [[3,3.5],[5,4.7],[8,7.9]] , columns = ["X","Y"])
D2 = pd.DataFrame(data = [[1,1.2],[4,4.7],[7,7.1]] , columns = ["X","Y"])
D3 = pd.DataFrame(data = [[2,1.8],[6,5.4],[9,8.1]] , columns = ["X","Y"])

D12 = D1.append(D2, ignore_index=True)
D23 = D2.append(D3, ignore_index=True)
D31 = D3.append(D1, ignore_index=True)

for d in range(0,3):
    poly_features = PolynomialFeatures(degree = d) #set the desired degree
    poly_model = LinearRegression(fit_intercept=True) #use linear regression

    x_train_poly = poly_features.fit_transform(D12[["X"]]) #transform training
    data
    poly_model.fit(x_train_poly, D12[["Y"]]) #fit model with training data only
    x_test_poly = poly_features.fit_transform(D3[["X"]]) #transform test data
    y_test_pred = poly_model.predict(x_test_poly) #predict using test data
    mse_3 = mean_squared_error(D3[["Y"]], y_test_pred)

    print("The coefficients for the model are :", poly_model.coef_[:,1:])
    print("The intercept for the model is :",poly_model.intercept_[0])
    print(mse_3)

    x_train_poly = poly_features.fit_transform(D23[["X"]]) #transform training
    data
    poly_model.fit(x_train_poly, D23[["Y"]]) #fit model with training data only
    x_test_poly = poly_features.fit_transform(D1[["X"]]) #transform test data
    y_test_pred = poly_model.predict(x_test_poly) #predict using test data
    mse_1 = mean_squared_error(D1[["Y"]], y_test_pred)

    print("The coefficients for the model are :", poly_model.coef_[:,1:])
    print("The intercept for the model is :",poly_model.intercept_[0])
    print(mse_1)

    x_train_poly = poly_features.fit_transform(D31[["X"]]) #transform training
    data
    poly_model.fit(x_train_poly, D31[["Y"]]) #fit model with training data only
    x_test_poly = poly_features.fit_transform(D2[["X"]]) #transform test data
    y_test_pred = poly_model.predict(x_test_poly) #predict using test data
    mse_2 = mean_squared_error(D2[["Y"]], y_test_pred)
    print("The coefficients for the model are :", poly_model.coef_[:,1:])
    print("The intercept for the model is :",poly_model.intercept_[0])
    print(mse_2)
```

Therefore, from the above code the test error for M_0 is as follows :

When we use D_1, D_2 for training and D_3 for testing,

$$mse_3 = 6.7225$$

When we use D_2, D_3 for training and D_1 for testing,

$$mse_1 = 3.8714$$

When we use D_3, D_1 for training and D_2 for testing,

$$mse_2 = 6.6789$$

Therefore,

$$\text{Test MSE for } M_0 = 3/9 * (mse_1 + mse_2 + mse_3) = 5.7576$$

The test error for M_1 is as follows :

When we use D_1, D_2 for training and D_3 for testing,

$$mse_3 = 0.4754$$

When we use D_2, D_3 for training and D_1 for testing,

$$mse_1 = 0.1102$$

When we use D_3, D_1 for training and D_2 for testing,

$$mse_2 = 0.3088$$

Therefore,

$$\text{Test MSE for } M_1 = 3/9 * (mse_1 + mse_2 + mse_3) = 0.2981$$

The test error for M_2 is as follows :

When we use D_1, D_2 for training and D_3 for testing,

$$mse_3 = 0.3926$$

When we use D_2, D_3 for training and D_1 for testing,

$$mse_1 = 0.1465$$

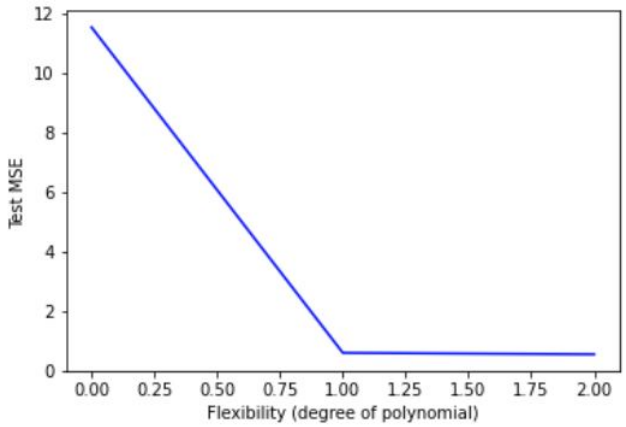
When we use D_3, D_1 for training and D_2 for testing,

$$mse_2 = 0.2825$$

Therefore,

$$\text{Test MSE for } M_2 = 3/9 * (mse_1 + mse_2 + mse_3) = 0.2739$$

b) Is the testing Error decreasing with the flexibility of your model?



From the above result, the test error decreases with flexibility

c) Based on this result, which ones of the models would you use in practice?

It is true that the test error for degree 2 polynomial is lower than that of a degree 1 polynomial. However, since the reduction is not too large we will select a simpler model of degree 1 where the test mse shows an elbow.

Problem 3 (Bootstrap): Consider the following dataset with $N = 3$ and $p = 1$:

$$\mathcal{D} = \{(1, 1.2), (2, 1.8), (3, 3.5), (4, 4.7)\}$$

We would like to estimate the uncertainty in the parameters of the following linear model: $\widehat{Y}_1 = \widehat{\beta}_0 + \widehat{\beta}_1 X$. To achieve this, we will use the bootstrap method considering that the following datasets are three bootstrap samples (each dataset is obtained by sampling four random points from $\mathcal{D}$, with replacement):

$$\mathcal{D}_1 = \{(1, 1.2), (3, 3.5), (1, 1.2), (4, 4.7)\}$$
$$\mathcal{D}_2 = \{(4, 4.7), (3, 3.5), (2, 1.8), (4, 4.7)\}$$
$$\mathcal{D}_3 = \{(1, 1.2), (4, 4.7), (1, 1.2), (4, 4.7)\}$$

(As in previous problems, feel free to use software to learn the parameters of each model; there is no need to do the computations by hand)

a) Use the bootstrap technique to estimate the standard deviation of the intercept $\widehat{\beta}_0$.

Solution code for learning parameters :

We will calculate the coefficients by training the model on the 3 bootstrap samples using the code below. Once we have the coefficient values for the models, we will calculate their standard deviation.

```
beta_0 = []
beta_1 = []

for d in D_list:
    poly_features = PolynomialFeatures(degree = 1) #set the desired degree
    poly_model = LinearRegression(fit_intercept=True) #use linear regression
    x_train_poly = poly_features.fit_transform(d[["X"]]) #transform training data
    poly_model.fit(x_train_poly, d[["Y"]]) #fit model with training data only
    beta_0.append(poly_model.intercept_[0])
    beta_1.append(poly_model.coef_[0,1])

beta_0_mean = np.mean(beta_0)
beta_1_mean = np.mean(beta_1)
print(beta_0_mean)
print(beta_1_mean)

sd_beta_0 =  np.sqrt(np.sum((beta_0-beta_0_mean)**2)/2)
sd_beta_1 = np.sqrt(np.sum((beta_1-beta_1_mean)**2)/2)
print(sd_beta_0)
print(sd_beta_1)
```

From the above code we find that the intercepts for the three bootstrap samples are as follows,

$$\hat{\beta}_0 \text{ for bootstrap } D_1 = 0.0333$$
$$\hat{\beta}_0 \text{ for bootstrap } D_2 = -0.9636$$
$$\hat{\beta}_0 \text{ for bootstrap } D_3 = 0.0333$$

The mean for the intercepts is ,

$$\begin{aligned}\bar{\beta_0} &= \frac{1}{M}\sum_{m=1}^{M}\hat{\beta_m} \\ &= \frac{1}{3}(0.0333 - 0.9636 + 0.0333) \\ &= -0.2990\end{aligned}$$

The standard deviation of the intercept is,

$$\begin{aligned}SD(\hat{\beta}_0) &= \sqrt{\frac{1}{M-1}\sum_{m=1}^{M}(\hat{\beta}_0 - \bar{\hat{\beta}_0})^2} \\ &= \sqrt{\frac{(0.0333 + 0.2990)^2 + (-0.9636 + 0.2990)^2 + (0.0333 + 0.2990)^2}{2}} \\ &= 0.5756\end{aligned}$$

b) Use the bootstrap technique to estimate the standard deviation of the slope $\widehat{\beta}_1$.

From the above code we find that the coefficient values for the three bootstrap samples are as follows,

$$\beta_1 \text{ for bootstrap } D_1 = 1.1630$$
$$\beta_1 \text{ for bootstrap } D_2 = 1.4273$$
$$\beta_1 \text{ for bootstrap } D_3 = 1.1667$$

The mean for the intercepts is ,

$$\begin{aligned}\bar{\beta_1} &= \frac{1}{M}\sum_{m=1}^{M}\hat{\beta_m} \\ &= \frac{1}{3}(1.1630 + 1.4273 + 1.1667) \\ &= 1.2523\end{aligned}$$

The standard deviation of the intercept is,

$$\begin{aligned}SD(\hat{\beta}_1) &= \sqrt{\frac{1}{M-1}\sum_{m=1}^{M}(\hat{\beta}_1 - \bar{\hat{\beta}_1})^2} \\ &= \sqrt{\frac{(1.1630 - 1.2523)^2 + (1.4273 - 1.2523)^2 + (1.1667 - 1.2523)^2}{2}} \\ &= 0.1515\end{aligned}$$