Model Assessment and Selection

Shizhi Chen-10307389

Python code Part 1:

```python
# Load the libraries needed
%matplotlib inline
import csv
import numpy as np
import matplotlib.pyplot as plt
import time

# Seed random number generator
np.random.seed(123)

# Helper functions for polynomials
def powers( X, n ):
    """ Returns an array of the powers of the elements of X up to the nth power """
    return np.power(np.expand_dims(X, axis=-1), [np.arange(n)])

def polynomial( X, poly_coeff ):
    """ Returns the value of a polynomial at x with given coefficients """
    deg = np.shape(poly_coeff)[-1]
    return np.dot(powers(X, deg), np.transpose(poly_coeff), )

def fit_polynomial( X, y, n ):
    """ Returns the coefficients of the n-degree polynomial fit to (X, y) """
    X_pwrs = powers(X, n+1)
    # Do linear least squares fit
    coeff, _, _, _ = np.linalg.lstsq(X_pwrs, y, rcond=None)
    return coeff
```

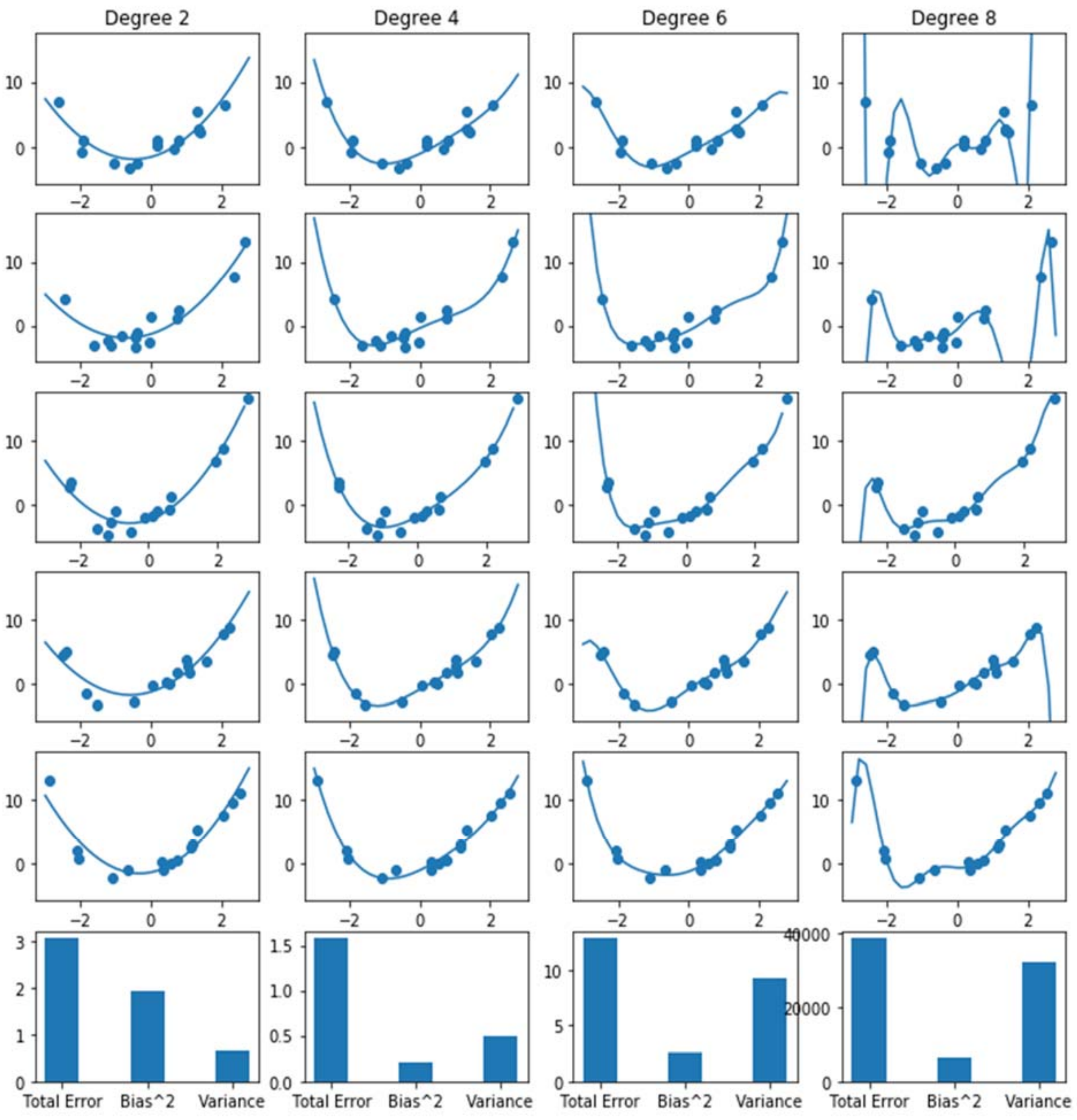
```python
#Visualise different degree polynomials on different data
num_resamplings = 5
polynomial_degrees = [ 2, 4, 6, 8 ]

# Load saved data
X, y, test_X, test_y, test_y_gt = np.load("./class8_generated_data.npy")

# Plots for plotting
fig, subplots = plt.subplots(nrows=num_resamplings+1,
                             ncols=len(polynomial_degrees),
                             figsize=(12, 12), )

folds_X = np.split(X, num_resamplings)
folds_y = np.split(y, num_resamplings)
#folds_eps = np.split(eps, num_resamplings)

# Make sequence of x-values for plotting
axis_X = np.arange(-3.0, 3.0, 0.2)
```

```python
# Functions for estimating mse, bias, and variance from a set of learned params
def bias_sq(x, learned_coeffs, true_y):
    mean_y = np.mean(polynomial(x, learned_coeffs), axis=-1)
    return np.square(mean_y - true_y)

def variance(x, learned_coeffs):
    return np.var(polynomial(x, learned_coeffs), axis=-1)

def square_error(x, learned_coeffs, y):
    pred_y = polynomial(x, learned_coeffs)
    return np.square(pred_y - np.expand_dims(y, -1))
```

```python
for j, n in enumerate(polynomial_degrees):

    # Placeholder for learned coefficients
    learned_coefficients = np.zeros((num_resamplings, n+1),)

    for i in range(num_resamplings):
        # Get ith subsample of data
        sub_X = folds_X[i]
        sub_y = folds_y[i]

        # Plot the data
        subplots[i,j].set_ylim([y.min()-1., y.max()+1.])
        subplots[i,j].scatter(sub_X, sub_y)

        # Fit polynomial to data
        learned_coefficients[i] = fit_polynomial( sub_X, sub_y, n )

        # Plot the learned polynomial
        poly_y = polynomial(axis_X, learned_coefficients[i])
        subplots[i,j].plot(axis_X, poly_y, )

    # Add titles
    subplots[0,j].set_title("Degree {:d}".format(n))

    # Plot MSE, bias squared, and variance
    mse = np.mean(square_error(test_X, learned_coefficients, test_y))
    bias = np.mean(bias_sq(test_X, learned_coefficients, test_y_gt))
    var = np.mean(variance(test_X, learned_coefficients))
    subplots[-1,j].bar([-2, .0, 2], [mse, bias, var], )
    subplots[-1,j].set_xticklabels(('','Total Error', 'Bias^2', 'Variance'))
plt.show()
```

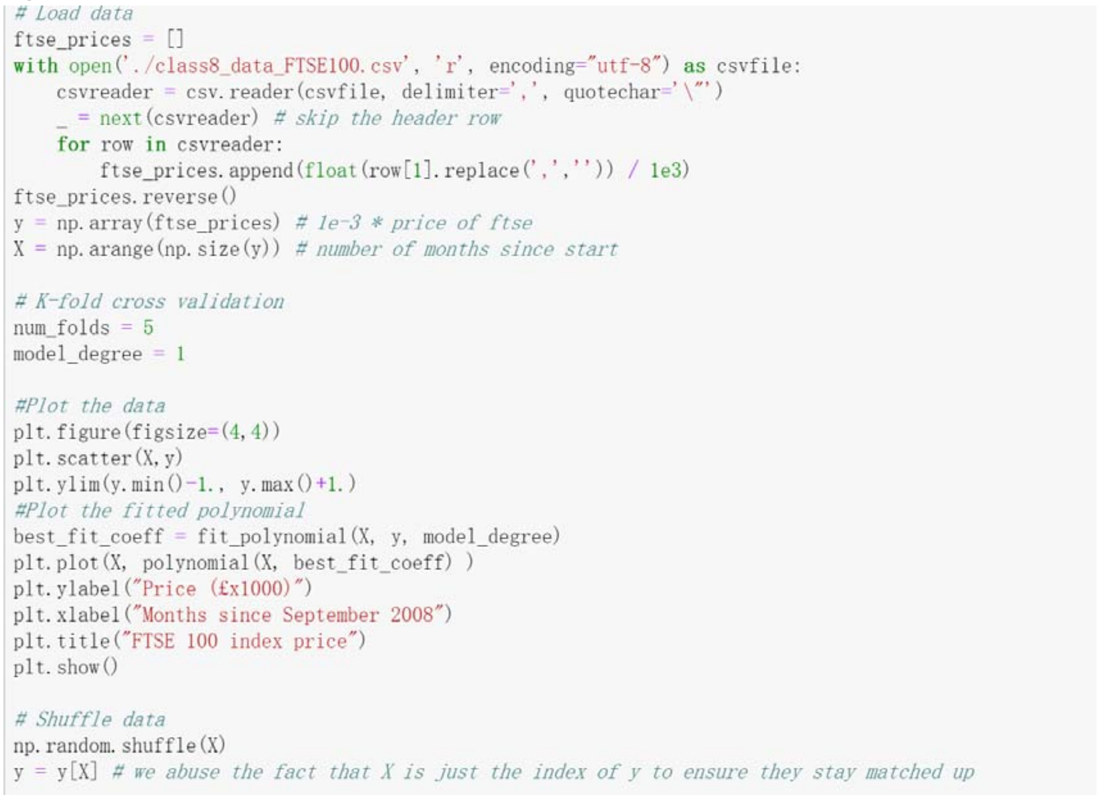
Assignment 1: Run the code provided and look at the plots of polynomials of different degrees fit to multiple samples randomly drawn from the data set. With the experimental evidence and your justification, comment on (i) how the bias and variance relate to a degree of the polynomial and (ii) which polynomial degree has been used to generate the data set.

Part 1 python code output:

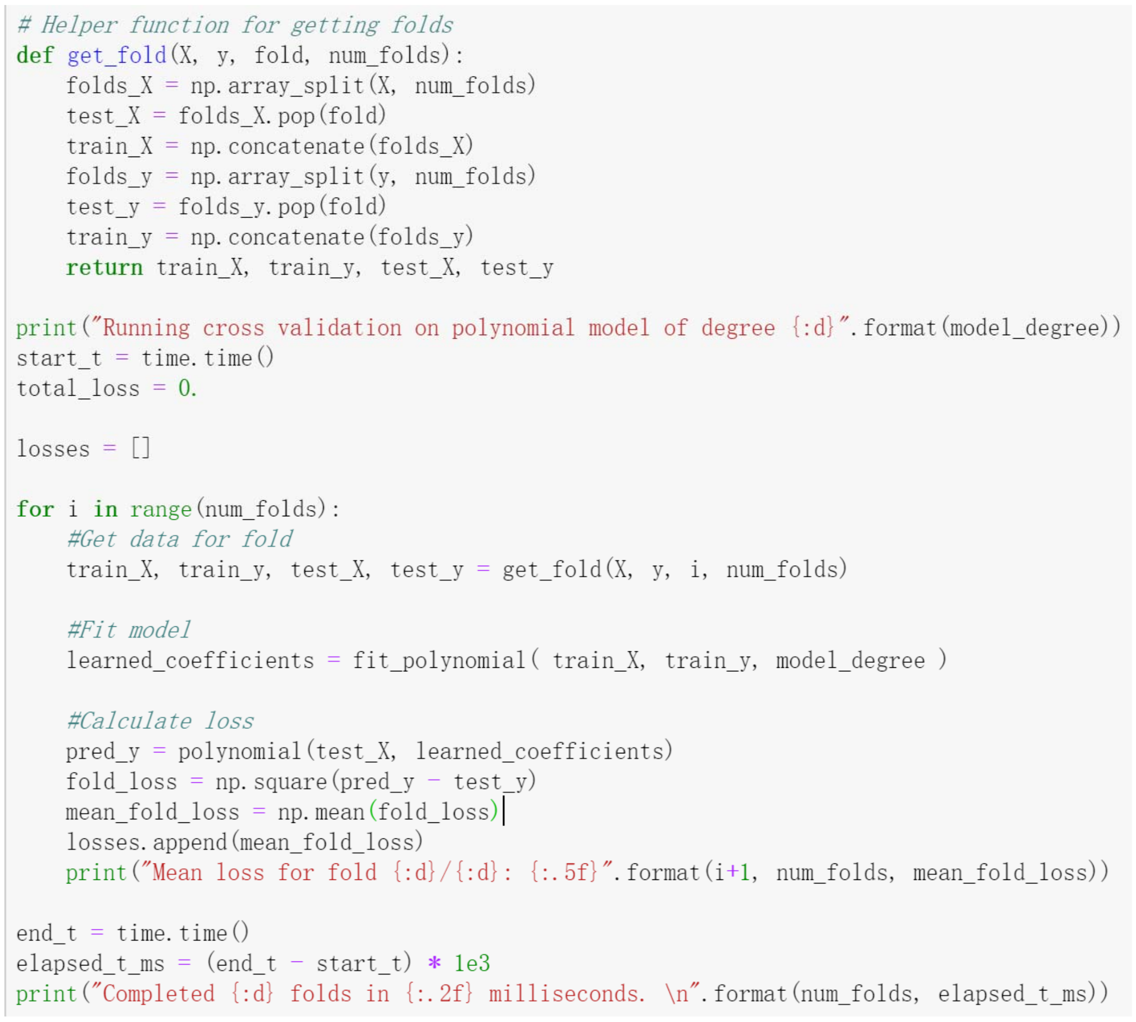


From the output we can see that when the degree of the polynomial is low then the bias and variance will be high. The bias and variance will decrease as the degree of the polynomial increases as we will tend to get a better fit. However, the bias and variance will again increase as higher degree polynomials that overfit the training set. In this dataset, the polynomial degree of 2,4,6 and 8 have been used and at the polynomial degree of 4 is considered as the good fit.

Python code Part 2:



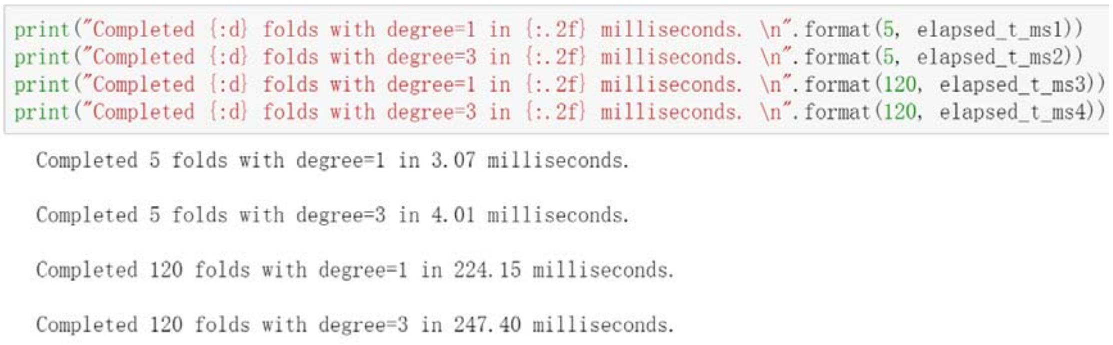
```python
# Load data
ftse_prices = []
with open('./class8_data_FTSE100.csv', 'r', encoding="utf-8") as csvfile:
    csvreader = csv.reader(csvfile, delimiter=',', quotechar='\"')
    _ = next(csvreader) # skip the header row
    for row in csvreader:
        ftse_prices.append(float(row[1].replace(',','')) / 1e3)
ftse_prices.reverse()
y = np.array(ftse_prices) # 1e-3 * price of ftse
X = np.arange(np.size(y)) # number of months since start

# K-fold cross validation
num_folds = 5
model_degree = 1

#Plot the data
plt.figure(figsize=(4,4))
plt.scatter(X,y)
plt.ylim(y.min()-1., y.max()+1.)
#Plot the fitted polynomial
best_fit_coeff = fit_polynomial(X, y, model_degree)
plt.plot(X, polynomial(X, best_fit_coeff) )
plt.ylabel("Price (£x1000)")
plt.xlabel("Months since September 2008")
plt.title("FTSE 100 index price")
plt.show()

# Shuffle data
np.random.shuffle(X)
y = y[X] # we abuse the fact that X is just the index of y to ensure they stay matched up
```



```python
# Helper function for getting folds
def get_fold(X, y, fold, num_folds):
    folds_X = np.array_split(X, num_folds)
    test_X = folds_X.pop(fold)
    train_X = np.concatenate(folds_X)
    folds_y = np.array_split(y, num_folds)
    test_y = folds_y.pop(fold)
    train_y = np.concatenate(folds_y)
    return train_X, train_y, test_X, test_y

print("Running cross validation on polynomial model of degree {:d}".format(model_degree))
start_t = time.time()
total_loss = 0.

losses = []

for i in range(num_folds):
    #Get data for fold
    train_X, train_y, test_X, test_y = get_fold(X, y, i, num_folds)

    #Fit model
    learned_coefficients = fit_polynomial( train_X, train_y, model_degree )

    #Calculate loss
    pred_y = polynomial(test_X, learned_coefficients)
    fold_loss = np.square(pred_y - test_y)
    mean_fold_loss = np.mean(fold_loss)
    losses.append(mean_fold_loss)
    print("Mean loss for fold {:d}/{:d}: {:.5f}".format(i+1, num_folds, mean_fold_loss))

end_t = time.time()
elapsed_t_ms = (end_t - start_t) * 1e3
print("Completed {:d} folds in {:.2f} milliseconds. \n".format(num_folds, elapsed_t_ms))
```

Assignment 2: For each of the two polynomial models (d = 1 or d = 3), record the approximate time spent by using 5-fold cross validation and LOOCV, respectively. Based on the experimental results, comment on the difference between the two cross-validation methods in terms of computational efficiency.

Change the parameters num_folds and model_degree to see the results of 5-fold cross validation (d=1 or d=3) and LOOCV (d=1 or d=3), and print the 4 models time spent:



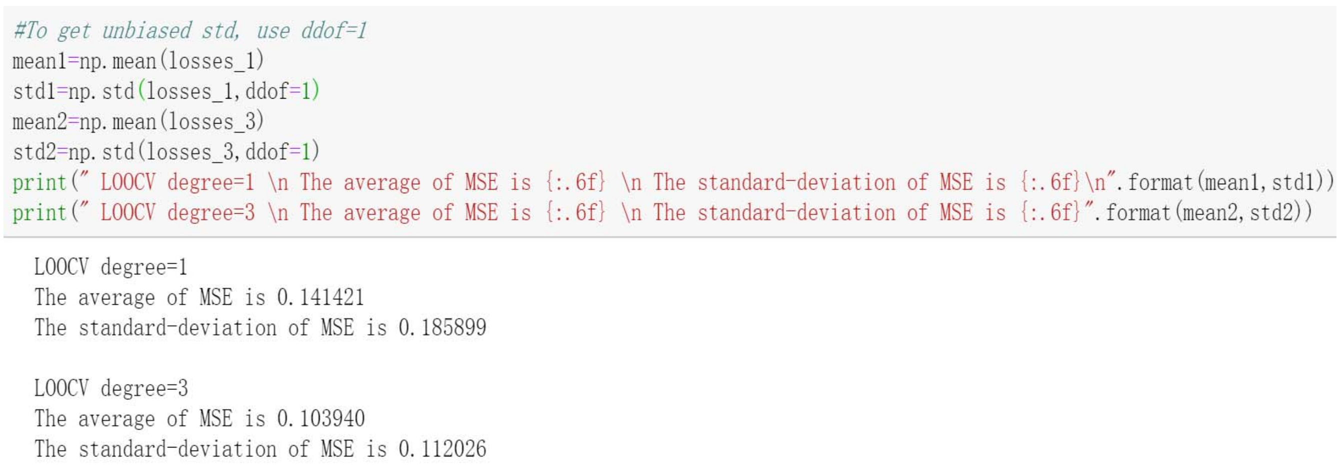
```
print("Completed {:d} folds with degree=1 in {:.2f} milliseconds. \n".format(5, elapsed_t_ms1))
print("Completed {:d} folds with degree=3 in {:.2f} milliseconds. \n".format(5, elapsed_t_ms2))
print("Completed {:d} folds with degree=1 in {:.2f} milliseconds. \n".format(120, elapsed_t_ms3))
print("Completed {:d} folds with degree=3 in {:.2f} milliseconds. \n".format(120, elapsed_t_ms4))
```

```
Completed 5 folds with degree=1 in 3.07 milliseconds.

Completed 5 folds with degree=3 in 4.01 milliseconds.

Completed 120 folds with degree=1 in 224.15 milliseconds.

Completed 120 folds with degree=3 in 247.40 milliseconds.
```

From the perspective of model runtime, it is clear that 5-fold cross validation is much more efficient than LOOCV. In the same model, with the higher degree, the time it takes will be slightly longer.

Assignment 3: For each of the two polynomial models (d = 1 or d = 3), calculate the average, and standard-deviation, of the mean square error loss on the validation set using LOOCV and record your results.

Python code and output:



```
#To get unbiased std, use ddof=1
mean1=np.mean(losses_1)
std1=np.std(losses_1,ddof=1)
mean2=np.mean(losses_3)
std2=np.std(losses_3,ddof=1)
print(" LOOCV degree=1 \n The average of MSE is {:.6f} \n The standard-deviation of MSE is {:.6f}\n".format(mean1,std1))
print(" LOOCV degree=3 \n The average of MSE is {:.6f} \n The standard-deviation of MSE is {:.6f}".format(mean2,std2))
```

```
 LOOCV degree=1
 The average of MSE is 0.141421
 The standard-deviation of MSE is 0.185899

 LOOCV degree=3
 The average of MSE is 0.103940
 The standard-deviation of MSE is 0.112026
```

Assignment 4: With the evidence and your justification, comment on whether the difference between the mean square errors of two models achieved in Assignment 3 is statistically significant. Be sure to include your code used for your justification in your report.

Use t-test to justify if the mean square errors of two models is statistically significant. However，in order to perform a two-sample t test we need to verify that the two samples are homoscedastic or heteroscedastic first.

Step1: Use scipy.stats module f to do F-test:

$$H_0: var_1=var_2,\ H_1: var_1 \neq var_2$$

```
from scipy.stats import f
```

```
nobs1 = 120
nobs2 = 120
var1=np.var(losses_1)
var2=np.var(losses_3)
F = var2 / var1
df1 = nobs1-1
df2 = nobs2-1
p_value = 1 - 2 * abs(0.5 - f.cdf(F, df1, df2))
print ("F statistic is {:.6f} ".format(F))
print ("p_value is {:.6f} ".format(p_value))
```

```
F statistic is 0.363146
p_value is 0.000000
```

From the result we can see that p_value is small than a=0.05, Rejecting the null hypothesis and means that the two samples are heteroscedastic.

Step 2: Conducting a t test of heteroscedasticity (setting eaqual_var=false)

$$H_0: \mu_1=\mu_2,\ H_1: \mu_1 \neq \mu_2$$

```
from scipy import stats
(statistic, pvalue) = stats.ttest_ind(losses_1,losses_3,equal_var=False)
print ("t statistic is {:.6f} ".format(statistic))
print ("pvalue is {:.6f} ".format(pvalue))
```

```
t statistic is 1.891721
pvalue is 0.060008
```

At the significance level of a=0.05, the p value=0.060008 is greater than 0.05, so the null hypothesis can not be rejected. Therefore, we consider that there is no significant difference between the mean square errors of the two models.