Data 558: Statistical Machine Learning

Spring 2023

Homework – 3

Arjun Sharma

Conceptual Questions:

1. Problem 1:

Review k-fold cross validation. Specifically:

(a) explain how k-fold cross-validation is implemented,

(b) advantages and disadvantages of k-fold cross-validation relative to validation set approach and LOOCV.

Answer:

(a) In K-fold cross validation, the data available to the user is randomly split into 'K' smaller, equally sized subsets, also referred to as *folds*. The first fold is treated as the validation set, and the model is fit on the remaining K-1 folds of data. The process is repeated K times where K ϵ *N*. Each time, the data is randomly sampled. The metric used to evaluate model performance is usually an error metric, typically the Mean-Squared Error, which is calculated for the model performance on the test sets. At the end of the process, we obtain K values of the test-error. The K-fold cross validation estimate is computed by averaging the said values. Mathematically, this is expressed as:

$$CV_{(K)} = MSE_1 + MSE_2 + MSE_3 + \dots + MSE_K = = \frac{1}{K}\sum_{i=1}^{K} MSE(i)$$

(b)

K-Fold Cross Validation vs. LOOCV

Advantages of K-Fold CV vs LOOCV:

1. Much lower number of repetitions of fitting and evaluating the model. Computationally very inexpensive compared to LOOCV, especially for more complex models.

2. The MSE for K-fold cross validation is much less prone to variance, a model fit using K-fold cross validation can be expected to be more generalizable than one trained using LOOCV

3. K-Fold CV gives better estimates of test error rates than LOOCV. This can be attributed to the bias-variance trade-off, as models that are fitted using the K-Fold CV techniques are less prone to variance compared to models fitted using LOOCV, which uses one datapoint as a validation set for each iteration.

Disadvantages of K-Fold CV vs LOOCV:

1. There is no randomness involved in fitting a model using the LOOCV technique, in contrast, the random selection of training examples in each fold in k-fold cross validation leaves room for some variability in the testing estimate, which is not the case in LOOCV.

2. In situations where we are interested in locating the minimum point in the estimated test MSE curve, K-Fold cross validation cannot be precisely calculated, however, this is possible when LOOCV is employed.

3. LOOCV is much less prone to bias because the validation set in each iteration is a singular datapoint.

K-Fold Cross Validation vs Validation Set Approach

Advantages of K-Fold CV vs Validation Set Approach:

1. In the case of the validation set approach the validation estimate of the test error rate can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set. The K-fold cross validation techniques – by iterating through the dataset K times – helps reduce this variance in test error rate.

2. In the validation approach, only a subset of the observations—those that are included in the training set rather than in the validation set—are used to fit the model. The k-fold cross validation approach allows for the model to work with a larger number of datapoints because of the randomness of the sampling and repetition of the same.

Disadvantages of K-Fold CV vs Validation Set Approach:

1. K-Fold Cross Validation is potentially computationally more expensive than validation set approach because of repetition of the training process, this is exacerbated in case of more complex models.

2. Since we average our resulting test-error estimates from the k testing sets in the K-Fold CV approach, we may obtain a relatively ambiguous test-error estimate using this approach compared to the validation set approach, which gives us the test-error estimate resulting from the singular test set.

2. Problem 2:

When do you expect the bootstrap procedure to give accurate confidence intervals and when would it give very inaccurate confidence intervals?

Answer:

The error in the bootstrap estimates can be computed mathematically based on the equation below:

$$\mathrm{SE}_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}\left(\hat{\alpha}^{*r} - \frac{1}{B}\sum_{r'=1}^{B}\hat{\alpha}^{*r'}\right)^2}.$$

Where B is the number of bootstraps, and $\hat{\alpha}^{*r}$ is the estimate for α.

The judgement for when bootstrapping procedure gives accurate confidence intervals comes down to minimizing the standard error.

Bootstrapping gives us accurate confidence intervals when:

1. B is large. From the Standard Error equation above, Larger B values would be less sensitive to influence of bootstrapped datasets which have extreme datapoints, and

2. Size of each bootstrapped dataset (i.e. n for each Z) is also large. Smaller bootstrapped datasets would be prone to influence from extreme values.

Bootstrapping gives us inaccurate confidence intervals when:

1. B is small. From the Standard Error equation above, smaller values of B would lead to risk of higher impact due to bootstrapped datasets having extreme datapoints, and

2. Size of each bootstrapped dataset is small. In this scenario, extreme values in a smaller dataset can lead to more noise.

3. Problem 3:

What is cross-validation trying to mimic? Describe two scenarios where cross-validation may not be appropriate. Please give a complete description

Answer:

K-fold cross validation attempts to mimic the process of dividing the available data into training and testing data in order to evaluate how the model has generalized on the data, and how the model is expected to perform on real-world data upon deployment.

In other words, the K-fold cross validation process is an imitation of the validation set approach. However, the approach is not always appropriate. Here are some scenarios where K-fold cross validation fails, or is at best a sub-optimal approach:

1. Time-Series Problems:

In time series problems, when the data is distributed over k-folds at random, the chronology of the datapoints would get disturbed. The model might end up being tested on some datapoints that occurred earlier than some of the datapoints it was trained on. For example, consider a model being developed to predict heart attacks based on the user's blood pressure and pulse data over time, it is possible that the use of k-fold cross validation would lead to the model being trained on data from January 1, 2023, while being tested on data from say, November 25, 2022.

This is a problem because:

a. The order of time being jumbled leads to erratic trends over time and,

b. Data leakage occurs when the latest data and older data are used together.

2. Problems with imbalanced data:

When k-fold cross validation is employed on imbalanced data either by the outcome or by the data distribution, it may introduce bias, since the data may not be properly distributed across the different folds of data. Going back to the example above, we would expect the instances of a datapoint corresponding to a heart attack to be very rare compared to normal readings. If K-fold cross validation were to be applied to such data, it is possible to have imbalanced folds, some of the situations below could occur:

a. Some folds having no datapoints corresponding to a heart attack,

b. Some folds having much more datapoints corresponding to a heart attack compared to other folds, etc.,

In scenarios like these, K-fold cross validation is not a suitable approach.

Applied Questions:

Problem 1:

In this problem, you will code your own cross-validation procedure and test it out on simulated data. Generate the simulated data as follows. Let x be a standard normal random variable and y = 0.5 + 0.5x − x2 + x3 + ε where ε is also a standard normal. Draw n = 100 samples that will be used for training and validation, and another n = 100 samples that will be used for evaluating test error.

(a) (10 points) Perform holdout validation on linear, quadratic, cubic, quartic, and quintic functions. Produce a plot with x-axis being complexity, y-axis MSE, and curve of validated MSE and the true test MSE. [Here, the true test MSE is the MSE obtained by fitting the a given complexity model to the test data]. Repeat the same steps for 10-fold validation and leave-one-out validation.

(b) (6 points) What is the optimal complexity chosen by each validation approach? What about the corresponding validation MSE? Which validation approach performs most favorably?

(c) (6 points) Repeat step (a) but with seed numbers 5 and 10. What do you notice?

(a)

```
import numpy as np

np.random.seed(1)
```

```
n_train = 100
n_test = 100

x_train = np.random.randn(n_train)
y_train = 0.5 + 0.5*x_train - x_train**2 + x_train**3 + np.random.randn(n_train)

x_test = np.random.randn(n_test)
y_test = 0.5 + 0.5*x_test - x_test**2 + x_test**3 + np.random.randn(n_test)
```

```
X_train = np.column_stack((np.ones(n_train), x_train, x_train**2, x_train**3, x_train**4, x_train**5))
X_test = np.column_stack((np.ones(n_test), x_test, x_test**2, x_test**3, x_test**4, x_test**5))
```

Holdout Validation

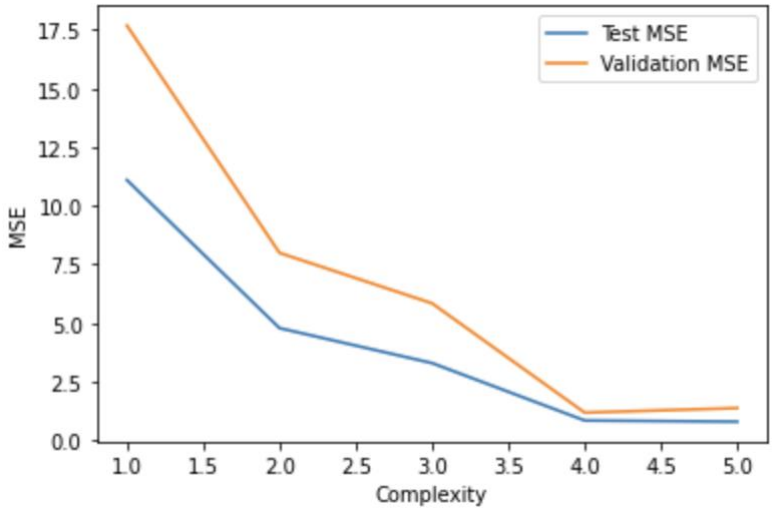
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

complexities = [1, 2, 3, 4, 5]
train_mses = []
val_mses = []
test_mses = []

for c in complexities:
    model = LinearRegression().fit(X_train[:, :c], y_train)
    val_mse = mean_squared_error(y_train, model.predict(X_train[:, :c]))
    test_mse = mean_squared_error(y_test, model.predict(X_test[:, :c]))
    val_mses.append(val_mse)
    test_mses.append(test_mse)

import matplotlib.pyplot as plt

plt.plot(complexities, val_mses, label='Validation MSE')
plt.plot(complexities, test_mses, label='Test MSE')
plt.xlabel('Complexity')
plt.ylabel('MSE')
plt.legend()
plt.show()
```



In [5]:
```
for i in range(5):
    print(f'Test MSE for complexity {i+1}: {val_mses[i]}')
```

```
Validation MSE for complexity 1: 11.096953693711491
Validation MSE for complexity 2: 4.785665849823237
Validation MSE for complexity 3: 3.2955832297166063
Validation MSE for complexity 4: 0.8481156274804116
Validation MSE for complexity 5: 0.7982194716115643
```

In [6]:
```
for i in range(5):
    print(f'Validation MSE for complexity {i+1}: {test_mses[i]}')
```

```
Test MSE for complexity 1: 17.683922391751313
Test MSE for complexity 2: 7.993021259910479
Test MSE for complexity 3: 5.843513142414914
Test MSE for complexity 4: 1.1839800601827455
Test MSE for complexity 5: 1.3758375509289429
```

K-Fold Cross Validation

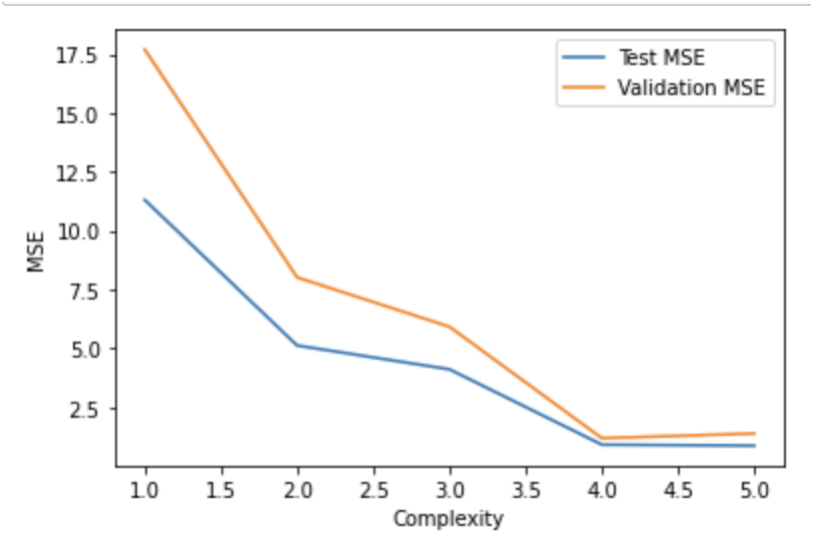
```
k = 10 # number of folds
n = 100
complexities = [1, 2, 3, 4, 5]
train_mses = []
val_mses = []
test_mses = []

for c in complexities:
    val_mse = 0
    train_mse = 0
    test_mse = 0
    for i in range(k):
        start = int(i*n/k)
        end = int((i+1)*n/k)
        val_idx = list(range(start, end))
        train_idx = list(set(range(n)) - set(val_idx))

        X_train_fold = X_train[train_idx, :c]
        y_train_fold = y_train[train_idx]
        X_val_fold = X_train[val_idx, :c]
        y_val_fold = y_train[val_idx]

        model = LinearRegression().fit(X_train_fold, y_train_fold)
        train_mse += mean_squared_error(y_train_fold, model.predict(X_train_fold))
        val_mse += mean_squared_error(y_val_fold, model.predict(X_val_fold))
        test_mse += mean_squared_error(y_test, model.predict(X_test[:, :c]))

    train_mses.append(train_mse/k)
    val_mses.append(val_mse/k)
    test_mses.append(test_mse/k)
```




```
In [8]: for i in range(5):
            print(f'Test MSE for complexity {i+1}: {test_mses[i]}')
```

```
Validation MSE for complexity 1: 11.307585647019673
Validation MSE for complexity 2: 5.124674469448967
Validation MSE for complexity 3: 4.113429160161744
Validation MSE for complexity 4: 0.9118181094477201
Validation MSE for complexity 5: 0.870604144882412
```

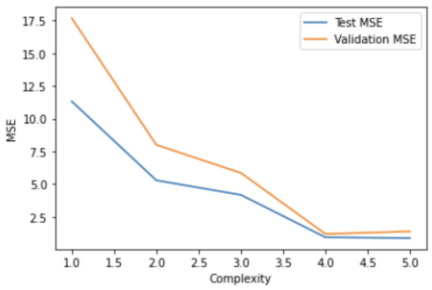
```
In [9]: for i in range(5):
            print(f'Validation MSE for complexity {i+1}: {val_mses[i]}')
```

```
Test MSE for complexity 1: 17.695008284030685
Test MSE for complexity 2: 8.019361838660997
Test MSE for complexity 3: 5.921922686050077
Test MSE for complexity 4: 1.1887977562965908
Test MSE for complexity 5: 1.3848188117875608
```

LOOCV

The code for K-fold cross validation is used, except that K = number of data points.

Results:




```
In [10]: for i in range(5):
             print(f'Validatiom MSE for complexity {i+1}: {val_mses[i]}')
```

```
Test MSE for complexity 1: 17.68505461843162
Test MSE for complexity 2: 7.996637215595804
Test MSE for complexity 3: 5.851489220757631
Test MSE for complexity 4: 1.1846506015860208
Test MSE for complexity 5: 1.3765415567416612
```

```
In [11]: for i in range(5):
             print(f'Test MSE for complexity {i+1}: {test_mses[i]}')
```

```
Validation MSE for complexity 1: 11.322266803093049
Validation MSE for complexity 2: 5.285695440895268
Validation MSE for complexity 3: 4.16996357963943
Validation MSE for complexity 4: 0.9268768781648815
Validation MSE for complexity 5: 0.8669116865881079
```

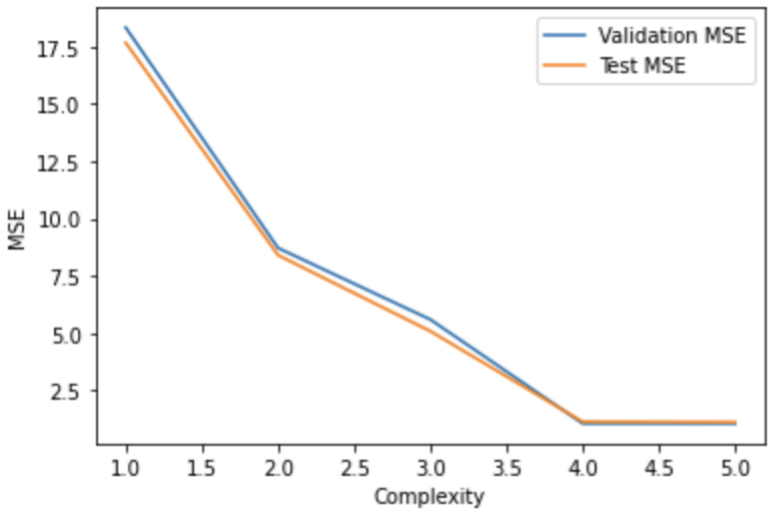
(b)

The optimal complexity chosen by all approaches is 5. LOOCV approach performs the best out of all three.

(c)

For seed 5:

Holdout



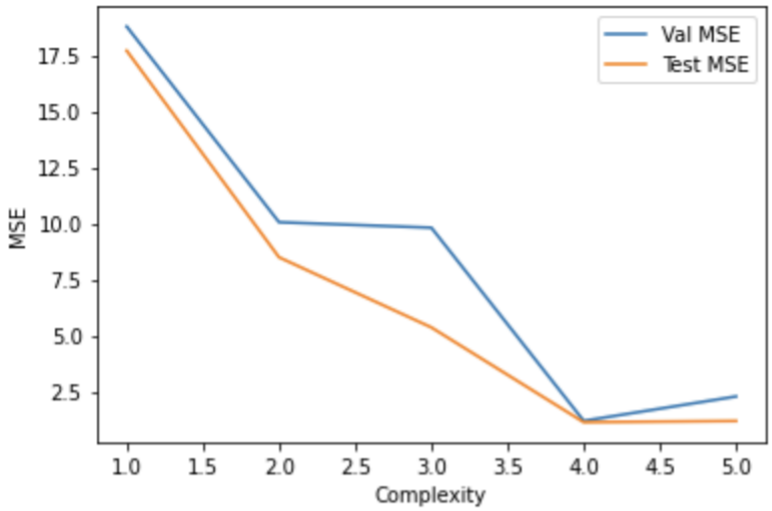

```
for i in range(5):
    print(f'Test MSE for complexity {i+1}: {val_mses[i]}')
```

```
Test MSE for complexity 1: 18.34231556038236
Test MSE for complexity 2: 8.71322436960584
Test MSE for complexity 3: 5.589055801239294
Test MSE for complexity 4: 1.0449889583464549
Test MSE for complexity 5: 1.0426628085797551
```

```
for i in range(5):
    print(f'Validation MSE for complexity {i+1}: {test_mses[i]}')
```

```
Validation MSE for complexity 1: 17.677790803859228
Validation MSE for complexity 2: 8.402985965635715
Validation MSE for complexity 3: 5.079025354766381
Validation MSE for complexity 4: 1.122850465130493
Validation MSE for complexity 5: 1.1030518302702534
```

Kfold



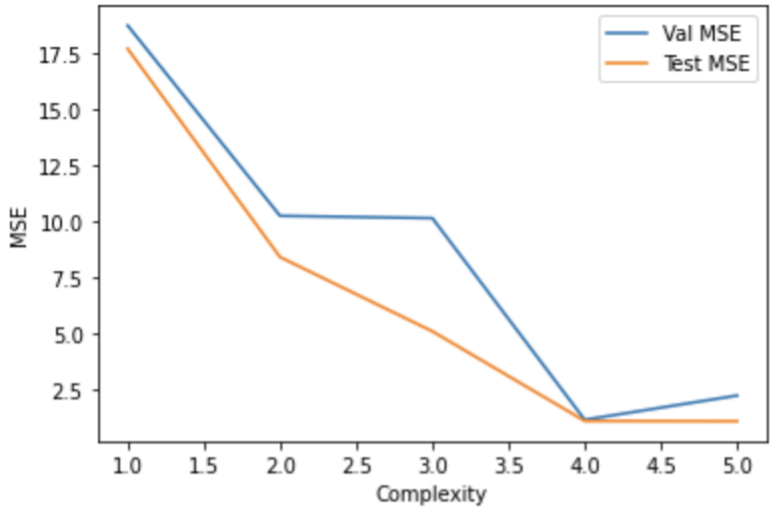

```
for i in range(5):
    print(f'Validatiom MSE for complexity {i+1}: {val_mses[i]}')
```

```
Validatiom MSE for complexity 1: 18.780486275534535
Validatiom MSE for complexity 2: 10.048659911896907
Validatiom MSE for complexity 3: 9.804247400270151
Validatiom MSE for complexity 4: 1.1870059052300364
Validatiom MSE for complexity 5: 2.27446990672018
```

```
for i in range(5):
    print(f'Test MSE for complexity {i+1}: {test_mses[i]}')
```

```
Test MSE for complexity 1: 17.70085242044619
Test MSE for complexity 2: 8.483772533581538
Test MSE for complexity 3: 5.354905628948137
Test MSE for complexity 4: 1.127570939730098
Test MSE for complexity 5: 1.180766939857093
```

LOOCV




```
for i in range(5):
    print(f'Validatiom MSE for complexity {i+1}: {val_mses[i]}')
```

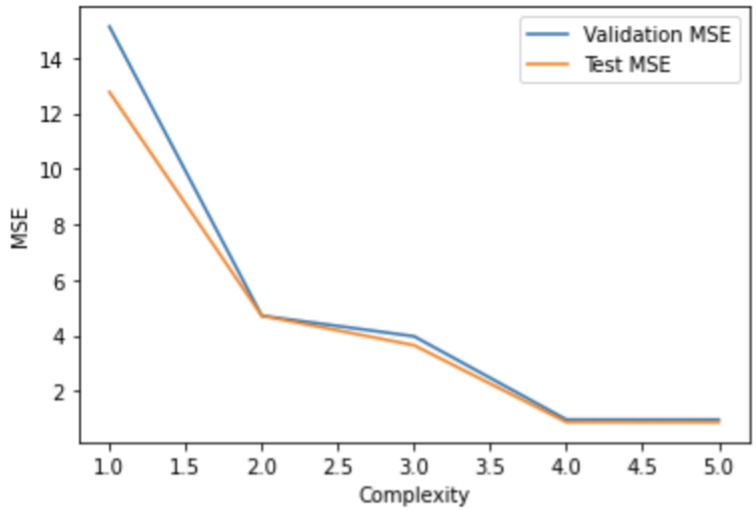
```
Validatiom MSE for complexity 1: 18.714738863771412
Validatiom MSE for complexity 2: 10.246462947676951
Validatiom MSE for complexity 3: 10.143138011291299
Validatiom MSE for complexity 4: 1.1737737984110723
Validatiom MSE for complexity 5: 2.25129364165133
```

```
for i in range(5):
    print(f'Test MSE for complexity {i+1}: {test_mses[i]}')
```

```
Test MSE for complexity 1: 17.679662277745614
Test MSE for complexity 2: 8.411857944590372
Test MSE for complexity 3: 5.108877457530325
Test MSE for complexity 4: 1.1232129546270788
Test MSE for complexity 5: 1.110555241112761
```

For seed 10:

Holdout



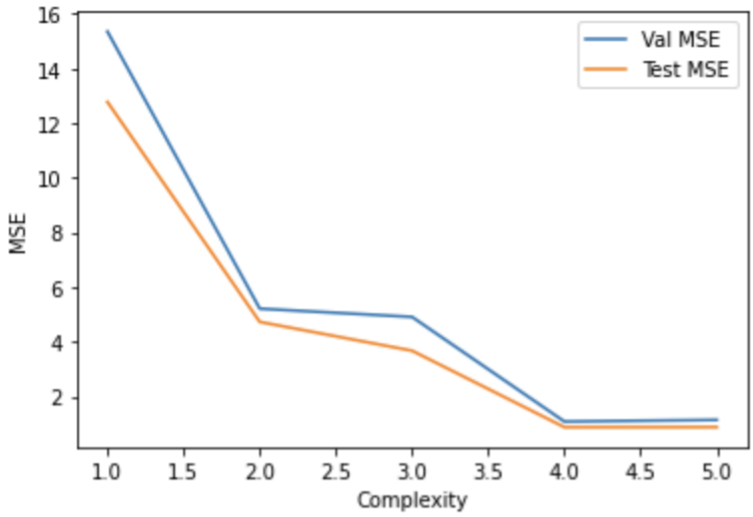

```
for i in range(5):
    print(f'Test MSE for complexity {i+1}: {val_mses[i]}')
```

```
Test MSE for complexity 1: 15.128094884660277
Test MSE for complexity 2: 4.706379347049027
Test MSE for complexity 3: 3.965638462126783
Test MSE for complexity 4: 0.9653483853390273
Test MSE for complexity 5: 0.9635492918461344
```

```
for i in range(5):
    print(f'Validation MSE for complexity {i+1}: {test_mses[i]}')
```

```
Validation MSE for complexity 1: 12.768415358089467
Validation MSE for complexity 2: 4.707925563157676
Validation MSE for complexity 3: 3.64288374221428
Validation MSE for complexity 4: 0.8744769003114946
Validation MSE for complexity 5: 0.8710507088579736
```

Kfold



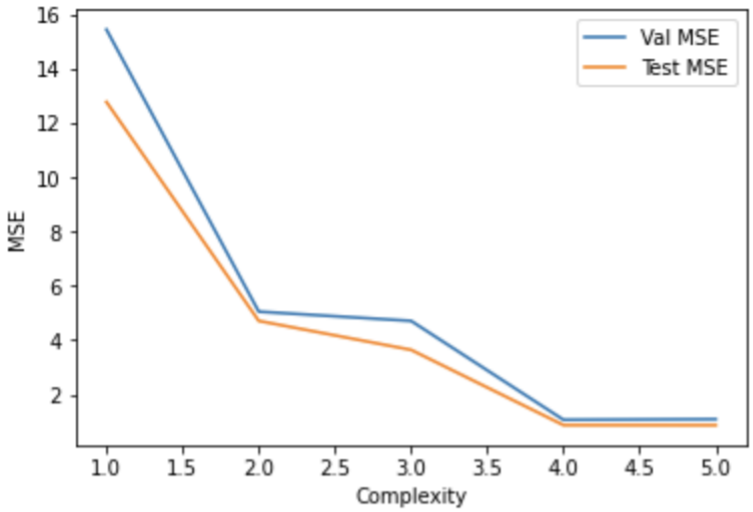

```python
for i in range(5):
    print(f'Validatiom MSE for complexity {i+1}: {val_mses[i]}')
```

```
Validatiom MSE for complexity 1: 15.355415768150698
Validatiom MSE for complexity 2: 5.215980481791458
Validatiom MSE for complexity 3: 4.9137947998435365
Validatiom MSE for complexity 4: 1.0868809373309232
Validatiom MSE for complexity 5: 1.1470036082084167
```

```python
for i in range(5):
    print(f'Test MSE for complexity {i+1}: {test_mses[i]}')
```

```
Test MSE for complexity 1: 12.78037961511528
Test MSE for complexity 2: 4.7315541628005295
Test MSE for complexity 3: 3.6781329682334
Test MSE for complexity 4: 0.8801234676448377
Test MSE for complexity 5: 0.8831289004408667
```

LOOCV




```
for i in range(5):
    print(f'Validatiom MSE for complexity {i+1}: {val_mses[i]}')
```

```
Validatiom MSE for complexity 1: 15.435256488787145
Validatiom MSE for complexity 2: 5.050356155468451
Validatiom MSE for complexity 3: 4.710392728228236
Validatiom MSE for complexity 4: 1.0710664600733877
Validatiom MSE for complexity 5: 1.0925510799150577
```

```
for i in range(5):
    print(f'Test MSE for complexity {i+1}: {test_mses[i]}')
```

```
Test MSE for complexity 1: 12.769958883738347
Test MSE for complexity 2: 4.709497335542761
Test MSE for complexity 3: 3.644350427230683
Test MSE for complexity 4: 0.8748852777723536
Test MSE for complexity 5: 0.8715915779661135
```

It is observed that the same trends are preserved across seeds. LOOCV performs best

Problem 2:

Problem 2 (24 points): Consider the setup in the previous problem. Our goal is to obtain confidence intervals for parameters of the cubic model by coding up your own bootstrap procedure.

(a) (8 points) Using B = 1000 bootstrap datasets, plot empirical distribution of your coefficients for the linear, quadratic, and cubic coefficients.
(b) (8 points) Based on the previous results, report the 95% interval for each coefficient and whether the true coefficient value is in the interval.
(c) (8 points) Compare your results to the R or python function.

Solution:

(a)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import resample
from scipy import stats
from sklearn.preprocessing import PolynomialFeatures
import statsmodels.api as sm
np.random.seed(1)
```

```
# Training Data
n = 100
X_train = np.random.normal(loc=0.0, scale=1.0, size = 100)
epsilon = np.random.normal(loc=0.0, scale=1.0, size = 100)
y_train = 0.5 + 0.5*X_train - X_train**2 + X_train**3 + epsilon

# Test data
X_test = np.random.normal(loc=0.0, scale=1.0, size = 100)
y_test = 0.5 + 0.5*X_test - X_test**2 + X_test**3 + epsilon
```

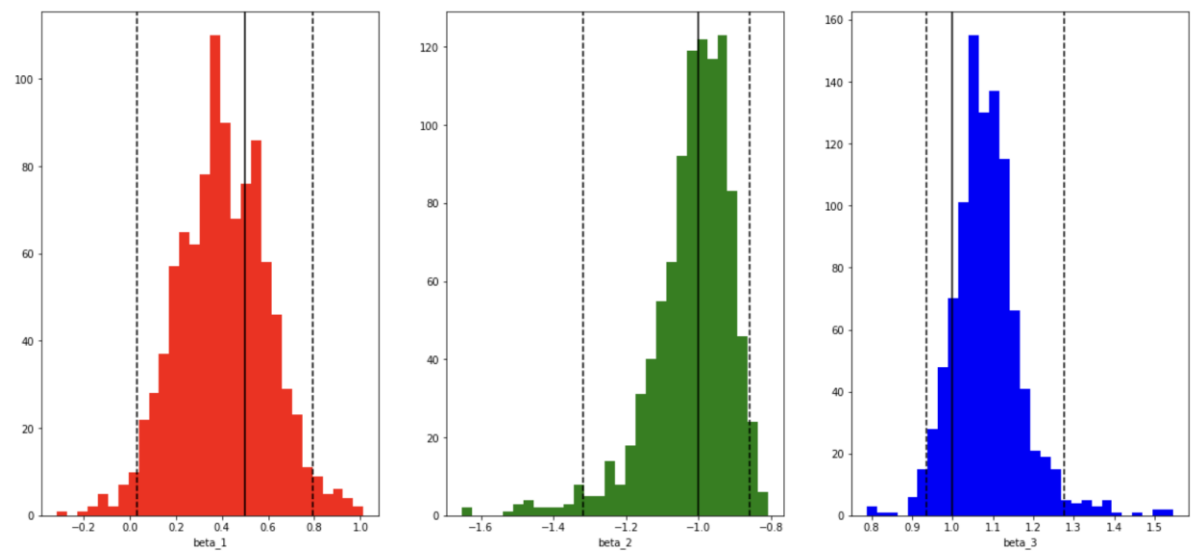
```
def bootstrap_polyfit(n, x, y):
    coefs = np.zeros((n, 3))
    for i in range(n):
        # Data Resampling
        x_resampled, y_resampled = resample(x, y, random_state = None)
        coefs_sample = np.polyfit(x_resampled, y_resampled, deg = 3)
        coefs[i,:] = [coefs_sample[2], coefs_sample[1], coefs_sample[0]]
    estimated_coefs = np.mean(coefs, axis = 0)

    return estimated_coefs, coefs
```

```
estimated_coefs, coefs = bootstrap_polyfit(1000, X_train, y_train)

# Plot the empirical distributions of the coefficients
fig, axes = plt.subplots(1, 3, figsize=(20,9))
color = ["red", "green", "blue"]
CI = np.percentile(coefs, [2.5, 97.5], axis = 0)
true_estimates = [0.5, -1, 1]

for i in range(3):
    axes[i].hist(coefs[:,i], bins=30, color = color[i])
    axes[i].set_xlabel(f'beta_{i+1}')
    lower_bound = CI[0,i]
    upper_bound = CI[1,i]
    axes[i].axvline(lower_bound, linestyle='--', color='black')
    axes[i].axvline(upper_bound, linestyle='--', color='black')
    axes[i].axvline(true_estimates[i], linestyle='solid', color='black')
plt.show()
```



(b)

```
print(f'CI for β1: [{CI[0,0]}, {CI[1,0]}]')
print(f'CI for β2: [{CI[0,1]}, {CI[1,1]}]')
print(f'CI for β3: [{CI[0,2]}, {CI[1,2]}]')
```

```
CI for β1: [0.03093065125663895, 0.7935765153321505]
CI for β2: [-1.3188815034350931, -0.8595465838622145]
CI for β3: [0.9363334421363982, 1.2775174501040012]
```

```
print(f'β1 (Estimated): {estimated_coefs[0]}')
print(f'β2 (Estimated): {estimated_coefs[1]}')
print(f'β3 (Estimated): {estimated_coefs[2]}')
```

```
β1 (Estimated): 0.4036348373539214
β2 (Estimated): -1.0178988955009853
β3 (Estimated): 1.0856754714168113
```

(c)

```
X_train = np.column_stack([np.ones(n), X_train, X_train**2, X_train**3])
model = sm.OLS(y_train, X_train).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.924
Model:	OLS	**Adj. R-squared:**	0.921
Method:	Least Squares	**F-statistic:**	386.7
Date:	Fri, 05 May 2023	**Prob (F-statistic):**	1.89e-53
Time:	15:44:33	**Log-Likelihood:**	-133.66
No. Observations:	100	**AIC:**	275.3
Df Residuals:	96	**BIC:**	285.7
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.6280	0.115	5.448	0.000	0.399	0.857
x1	0.4065	0.187	2.171	0.032	0.035	0.778
x2	-0.9753	0.085	-11.449	0.000	-1.144	-0.806
x3	1.0788	0.065	16.644	0.000	0.950	1.208

Omnibus:	1.539	**Durbin-Watson:**	2.129
Prob(Omnibus):	0.463	**Jarque-Bera (JB):**	1.081
Skew:	-0.236	**Prob(JB):**	0.583
Kurtosis:	3.193	**Cond. No.**	5.53

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
SE = model.bse
t_stat = stats.t.ppf(0.975, n - 4)
linear = model.params[1] + np.array([-1, 1]) * t_stat * SE[1]
quadratic = model.params[2] + np.array([-1, 1]) * t_stat * SE[2]
cubic = model.params[3] + np.array([-1, 1]) * t_stat * SE[3]

print('CI for Linear Coefficient:', linear)
print('CI for Quadratic Coefficient:', quadratic)
print('CI for Cubic Coefficient:', cubic)
```

```
CI for Linear Coefficient: [0.03486142 0.77807054]
CI for Quadratic Coefficient: [-1.14442531 -0.80621781]
CI for Cubic Coefficient: [0.95018282 1.20750628]
```

Based on the results, although there are minor differences in the absolute values, it appears that the bootstrapping method developed by me is consistent with the results obtained through the statsmodels library.

Problem 4:

The “heart” dataset from the “Heart.csv” file contains information on the presence or absence of heart disease in patients. We will build a logistic regression model to predict the probability of heart disease based on the patient’s age, sex, resting blood pressure, serum cholesterol, etc.

(a) (4 points) Using the glm() function in R (or its correspondence in the statsmodels package in Python), estimate the coefficients for age, sex, resting blood pressure, and serum cholesterol in the logistic regression model. Compute the estimated standard errors for these coefficients using the summary() function.

(b) (6 points) Write a function, cv.fn(), that takes as input the dataset and a value for k (the number of folds for cross-validation) and outputs the cross-validation estimate of the misclassification rate for the logistic regression model. You may use the cv.glm() function in R or the cross val score() function in scikit-learn (Python) to perform the cross-validation
(c) (4 points) Use the cv.fn() function to compute the cross-validation estimate of the misclassification rate for the logistic regression model.

(d) (6 points) Use the bootstrapping technique to perform a bootstrap analysis of the logistic regression model. Write a function, boot.fn(), that takes as input the dataset and an index of the observations and outputs the coefficient estimates for the logistic regression model. Use your boot.fn() function together with the bootstrapping technique to estimate the standard errors of the logistic regression coefficients for age, sex, resting blood pressure, and serum cholesterol.

(e) Comment on the differences between the estimated standard errors obtained using the glm() function and the bootstrap method. Which method do you think is more reliable and why?

Solution:

(a)

```{r}
set.seed(1)
heart <- read.csv("C://Users//arjun//Desktop//Quarter_3//ML//Homeworks//HW3//heart.csv")
heart$AHD<-ifelse(heart$AHD == "Yes", 1, 0)
heart

df <- data.frame(heart$Age, heart$Sex, heart$RestBP, heart$Chol, heart$AHD)
```

```{r}
fit_heart <- glm(AHD ~ Age + Sex + RestBP + Chol, data = heart, family = binomial)
summary(fit_heart)
```

```
Call:
glm(formula = AHD ~ Age + Sex + RestBP + Chol, family = binomial,
    data = heart)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.8518  -1.0297  -0.4384   1.0201   1.8785

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -7.389471   1.380537  -5.353 8.67e-08 ***
Age          0.055793   0.015539   3.591  0.00033 ***
Sex          1.677529   0.308207   5.443 5.24e-08 ***
RestBP       0.014116   0.007593   1.859  0.06302 .
Chol         0.004573   0.002535   1.804  0.07126 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 417.98  on 302  degrees of freedom
Residual deviance: 365.07  on 298  degrees of freedom
AIC: 375.07

Number of Fisher Scoring iterations: 3
```

(b)

```{r}
# HW3 Applied 4-b
cv.fn <- function(data, k) {
  library("boot")
  glm.fit <- glm(AHD ~ Age + Sex + RestBP + Chol, data = data, family = binomial)
  cv.err <- cv.glm(data, glm.fit, K = k)$delta[1]
  return(cv.err)
}
```

(c)

```r
#data <- read.csv("mydata.csv")
cv.err <- cv.fn(heart, k = 10)

# Print the cross-validation estimate of the misclassification rate
print(cv.err)
```

```
[1] 0.2193802
```

(d)

```r
#logistic regression
glm.fit <- glm(AHD ~ Age + Sex + RestBP + Chol, data = heart, family = binomial)

# bootstrapping function
boot.fn <- function(data, index) {
  return(coef(glm(AHD ~ Age + Sex + RestBP + Chol, data = heart, subset = index, family = binomial)))
}

library("boot")
set.seed(123)
boot.out <- boot(heart, boot.fn, R = 1000)

#standard errors of the logistic regression coefficients
boot.out
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = heart, statistic = boot.fn, R = 1000)


Bootstrap Statistics :
        original        bias    std. error
t1* -7.389471280 -0.2120294617 1.455545346
t2*  0.055792766  0.0021215492 0.015299853
t3*  1.677529301  0.0390198038 0.307256076
t4*  0.014115768  0.0001480609 0.008049876
t5*  0.004572997  0.0002145920 0.002799201
```

(e)

The difference between traditional Logistic Regression and Bootstrapping is not significant at B = 1000. However, had the bootstrap value been lower, the standard errors would have been significantly higher. Due to the smaller sample size, Logistic Regression appears to be more reliable. Had the sample size been larger, Bootstrapping would have been considered more reliable.

Problem 5:

repeat the previous question but with the opposite programming language. If you used R, now use Python. If you used Python, now use R.