Sarah Riaz

USC ID: 1389209718

sriaz@usc.edu

Homework 2

Question 1:

c)

i) Min, max, Mean, standard deviation, median, skewness and kurtosis of time series are effective for time series classification. The most commonly used time domain features used for time series classification are min, max, mean, median and standard deviation.

ii)

The most commonly used time domain features used for time series classification are min, max, mean, median and standard deviation. Therefore, I calculated these features for each time series using min(), max(), mean(), std() and median() functions. I have used two libraries/packages for this task: numpy and pandas.

I calculated these features of each time domain series in all the datasets of each folder representing different activities e.g. walking, bending.

iii)

Bootstrapped 95% confidence intervals of Min:

Low: 0.414562115907

High: 4.68457829017

Bootstrapped 95% confidence intervals of Max:

Low: 2.21848246184

High: 3.41148439184

Bootstrapped 95% confidence intervals of Mean:

Low: 1.25202641058

High: 2.90346680359

Bootstrapped 95% confidence intervals of Median:

Low: 1.18816762121

High: 2.97413437201

Bootstrapped 95% confidence intervals of Standard Deviation:

Low: 0.50313601049

High: 0.900094241374

Code:

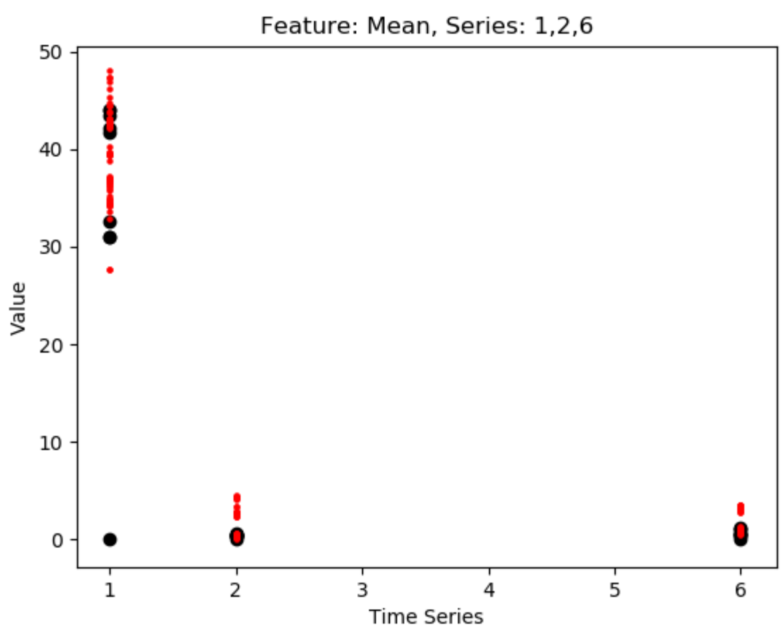
```
import scikits.bootstrap as bootstrap

# standard deviation of each time-domain feature
min_data = [6.51562407169, 0.0, 2.40872055483, 0.0, 4.28029876297,
0.0262173802034]

max_data = [2.56192815232, 3.88072298016, 3.38089262879, 1.82927287927,
3.3823198048, 1.92387534411]

mean_data = [3.23227197338, 1.11362166938, 2.55635152353, 0.967510948022,
3.4742345875, 0.939652426485]

median_data = [3.35245027268, 0.95992089351, 2.51819498442, 0.95138220168,
3.58532073343, 0.874062722993]

std_data = [1.12536705831, 0.681276225331, 0.571370859196, 0.401493440186,
0.671594023992, 0.445455996043]

CIs = bootstrap.ci(data=min_data)
print("Bootstrapped 95% confidence intervals of Min \nLow:", CIs[0], "\nHigh:",
CIs[1])

CIs = bootstrap.ci(data=max_data)
print("Bootstrapped 95% confidence intervals of Max \nLow:", CIs[0], "\nHigh:",
CIs[1])

CIs = bootstrap.ci(data=mean_data)
print("Bootstrapped 95% confidence intervals of Mean \nLow:", CIs[0],
"\nHigh:", CIs[1])

CIs = bootstrap.ci(data=median_data)
print("Bootstrapped 95% confidence intervals of Median \nLow:", CIs[0],
"\nHigh:", CIs[1])

CIs = bootstrap.ci(data=std_data)
print("Bootstrapped 95% confidence intervals of Standard Deviation \nLow:",
CIs[0], "\nHigh:", CIs[1])
```

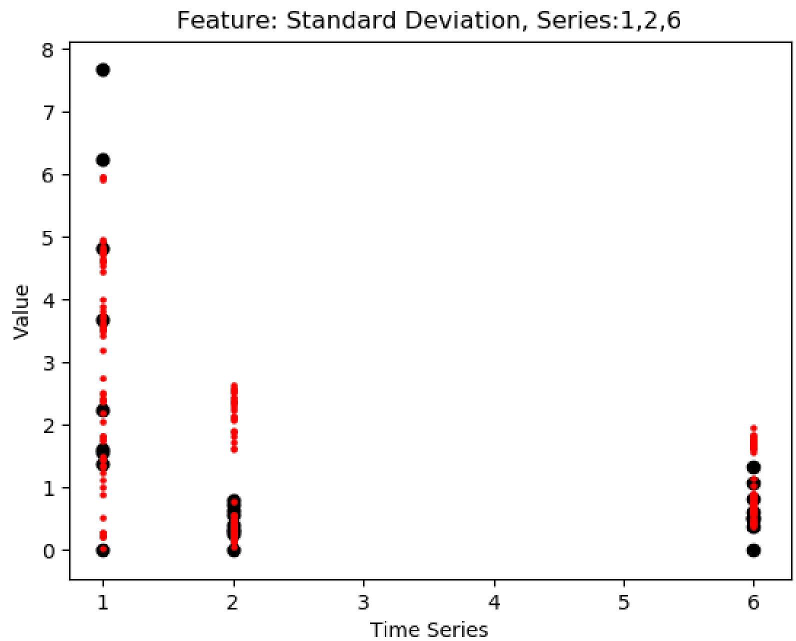
iv)

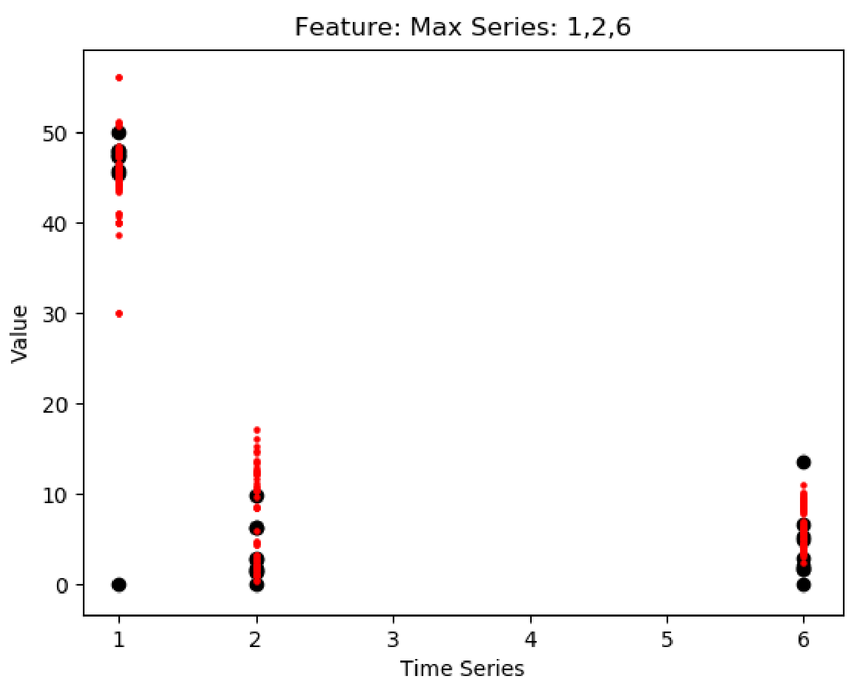
I have selected max, mean and standard deviation as the most important features because after analyzing the results min and median don't seem as significant as others. Min value of most of the time domain series was zero. Others have a good range of distinct values.

d)

i) Scatter plots (black shows bending values):







Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

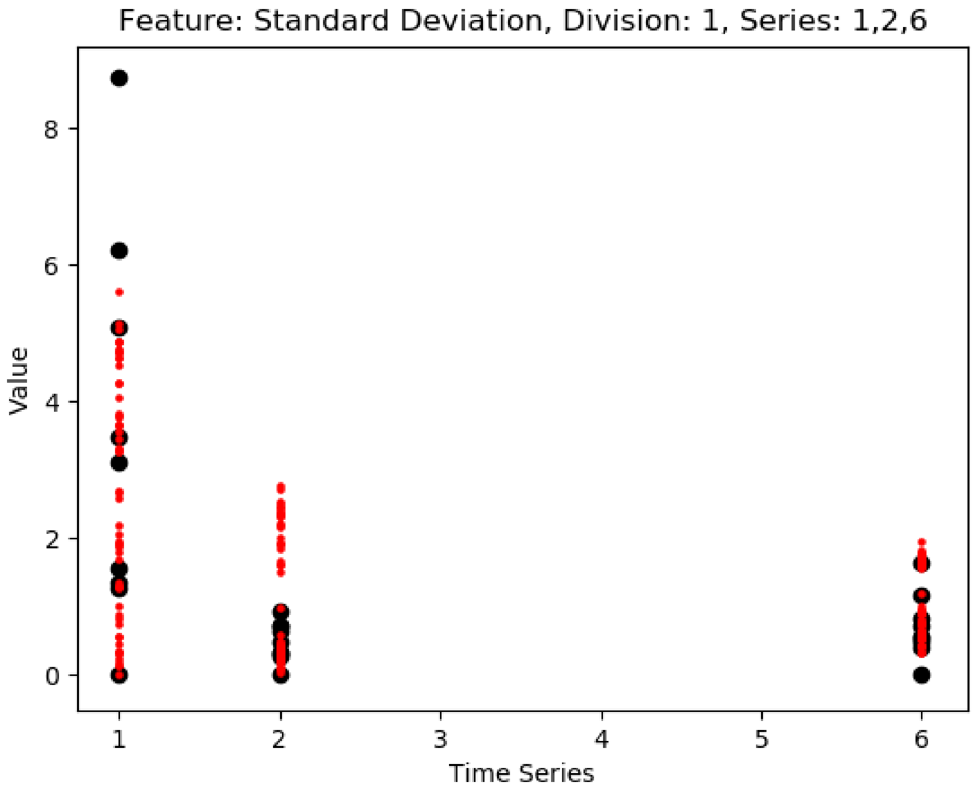
```python
bending_std_series_1=[]

bending_std_series_1=feature_std_1
bending_std_single_array=np.array(bending_std_series_1)
bending_std=bending_std_single_array.ravel()

bending_std_1_split=[]
for i in range(0,len(bending_std)):
    tmp_val=bending_std[i]
    for j in range(0,len(tmp_val)):
        bending_std_1_split.append(tmp_val[j])
bending_std_1_split_array=np.array(bending_std_1_split)

#other activities std
others_std_series_1=[]
others_std_series_1=feature_std_1
others_std_single_array=np.array(others_std_series_1)
others_std=others_std_single_array.ravel()

others_std_1_split=others_std

bending_std_series_6=[]

bending_std_series_6 = feature_std_6
bending_std_single_array = np.array(bending_std_series_6)
bending_std = bending_std_single_array.ravel()

bending_std_6_split=[]
for i in range(0,len(bending_std)):
    tmp_val=bending_std[i]
    for j in range(0,len(tmp_val)):
        bending_std_6_split.append(tmp_val[j])
bending_std_6_split_array=np.array(bending_std_6_split)


bending_std_1_split_array= np.nan_to_num(bending_std_1_split_array)
others_std_1_split=np.nan_to_num(others_std_1_split)

bending_std_2_split_array=np.nan_to_num(bending_std_2_split_array)
others_std_2_split=np.nan_to_num(others_std_2_split)

bending_std_6_split_array=np.nan_to_num(bending_std_6_split_array)
others_std_6_split=np.nan_to_num(others_std_6_split)

#other activities max
others_max_series_1=[]
others_max_series_1=feature_max_1
others_max_single_array=np.array(others_max_series_1)
others_max=others_max_single_array.ravel()

others_max_1_split=others_max

bending_max_series_1=[]
```

```python
for i in range(0,len(bending_max)):
    tmp_val=bending_max[i]
    for j in range(0,len(tmp_val)):
        bending_max_1_split.append(tmp_val[j])
bending_max_1_split_array=np.array(bending_max_1_split)

bending_max_2_split=[]
for i in range(0,len(bending_max)):
    tmp_val=bending_max[i]
    for j in range(0,len(tmp_val)):
        bending_max_2_split.append(tmp_val[j])
bending_max_2_split_array=np.array(bending_max_2_split)

bending_max_series_6=[]

bending_max_series_6=feature_max_6
bending_max_single_array=np.array(bending_max_series_6)
bending_max=bending_max_single_array.ravel()

bending_max_6_split=[]
for i in range(0,len(bending_max)):
    tmp_val=bending_max[i]
    for j in range(0,len(tmp_val)):
        bending_max_6_split.append(tmp_val[j])
bending_max_6_split_array=np.array(bending_max_6_split)

bending_max_1_split_array=np.nan_to_num(bending_max_1_split_array)
others_max_1_split=np.nan_to_num(others_max_1_split)

bending_max_2_split_array=np.nan_to_num(bending_max_2_split_array)
others_max_2_split=np.nan_to_num(others_max_2_split)

bending_max_6_split_array=np.nan_to_num(bending_max_6_split_array)
others_max_6_split=np.nan_to_num(others_max_6_split)


bending_mean_series_1=[]

bending_mean_series_1=feature_mean_1
bending_mean_single_array=np.array(bending_mean_series_1)
bending_mean=bending_mean_single_array.ravel()

bending_mean_1_split=[]
for i in range(0,len(bending_mean)):
    tmp_val=bending_mean[i]
    for j in range(0,len(tmp_val)):
        bending_mean_1_split.append(tmp_val[j])
bending_mean_1_split_array=np.array(bending_mean_1_split)

#other activities mean
others_mean_series_1=[]
others_mean_series_1=feature_mean_1
others_mean_single_array=np.array(others_mean_series_1)
others_mean=others_mean_single_array.ravel()
```

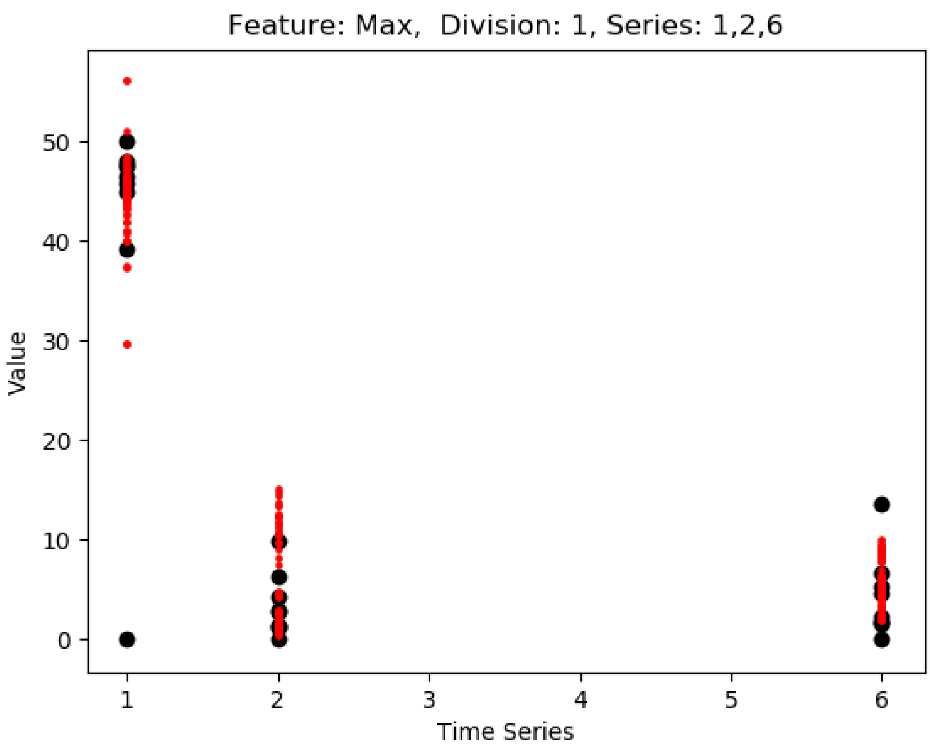
```
others_mean_1_split=others_mean

bending_mean_series_2=[]

bending_mean_series_2=feature_mean_2
bending_mean_single_array=np.array(bending_mean_series_2)
bending_mean=bending_mean_single_array.ravel()


bending_mean_2_split=[]
for i in range(0,len(bending_mean)):
    tmp_val=bending_mean[i]
    for j in range(0,len(tmp_val)):
        bending_mean_2_split.append(tmp_val[j])
bending_mean_2_split_array=np.array(bending_mean_2_split)

bending_mean_series_6=[]
bending_mean_series_6=feature_mean_6
bending_mean_single_array=np.array(bending_mean_series_6)
bending_mean=bending_mean_single_array.ravel()

bending_mean_6_split=[]
for i in range(0,len(bending_mean)):
    tmp_val=bending_mean[i]
    for j in range(0,len(tmp_val)):
        bending_mean_6_split.append(tmp_val[j])
bending_mean_6_split_array=np.array(bending_mean_6_split)

bending_mean_1_split_array=np.nan_to_num(bending_mean_1_split_array)
others_mean_1_split=np.nan_to_num(others_mean_1_split)


bending_mean_2_split_array=np.nan_to_num(bending_mean_2_split_array)
others_mean_2_split=np.nan_to_num(others_mean_2_split)

bending_mean_6_split_array=np.nan_to_num(bending_mean_6_split_array)
others_mean_6_split=np.nan_to_num(others_mean_6_split)


fig, ax = plt.subplots()

plt.xlabel('Time Series')
plt.ylabel('Value')
plt.title('Feature: Mean, Series: 1,2,6')
for i in range(0,len(bending_mean_1_split_array)):
    ax.scatter(1,bending_mean_1_split_array[i],  color='black', label='$x$')
for i in range(0,len(others_mean_1_split)):
    ax.scatter(1,others_mean_1_split[i], 5,color='red', label='$x$')

for i in range(0, len(bending_mean_2_split_array)):
    ax.scatter(2,bending_mean_2_split_array[i], color='black', label='$x$')
for i in range(0, len(others_mean_2_split)):
    ax.scatter(2,others_mean_2_split[i],  5,color='red', label='$x$')

for i in range(0, len(bending_mean_6_split)):
    ax.scatter(6,bending_mean_6_split_array[i],  color='black', label='$x$')
```

```
for i in range(0, len(others_mean_6_split)):
    ax.scatter(6,others_mean_6_split[i], 5,color='red', label='$x$')

plt.show()

fig, ax = plt.subplots()

plt.xlabel('Time Series')
plt.ylabel('Value')
plt.title('Feature: Standard Deviation, Series:1,2,6')
for i in range(0,len(bending_std_1_split_array)):
    ax.scatter(1,bending_std_1_split_array[i],  color='black', label='$x$')
for i in range(0,len(others_std_1_split)):
    ax.scatter(1,others_std_1_split[i], 5,color='red', label='$x$')

for i in range(0, len(bending_std_2_split_array)):
    ax.scatter(2,bending_std_2_split_array[i], color='black', label='$x$')
for i in range(0, len(others_std_2_split)):
    ax.scatter(2,others_std_2_split[i],  5,color='red', label='$x$')

for i in range(0, len(bending_std_6_split)):
    ax.scatter(6,bending_std_6_split_array[i],  color='black', label='$x$')
for i in range(0, len(others_std_6_split)):
    ax.scatter(6,others_std_6_split[i], 5,color='red', label='$x$')

plt.show()


fig, ax = plt.subplots()

plt.xlabel('Time Series')
plt.ylabel('Value')
plt.title('Feature: Max Series: 1,2,6')
for i in range(0,len(bending_max_1_split_array)):
    ax.scatter(1,bending_max_1_split_array[i],  color='black', label='$x$')
for i in range(0,len(others_max_1_split)):
    ax.scatter(1,others_max_1_split[i], 5,color='red', label='$x$')

for i in range(0, len(bending_max_2_split_array)):
    ax.scatter(2,bending_max_2_split_array[i], color='black', label='$x$')
for i in range(0, len(others_max_2_split)):
    ax.scatter(2,others_max_2_split[i],  5,color='red', label='$x$')

for i in range(0, len(bending_max_6_split)):
    ax.scatter(6,bending_max_6_split_array[i],  color='black', label='$x$')
for i in range(0, len(others_max_6_split)):
    ax.scatter(6,others_max_6_split[i], 5,color='red', label='$x$')

plt.show()
```

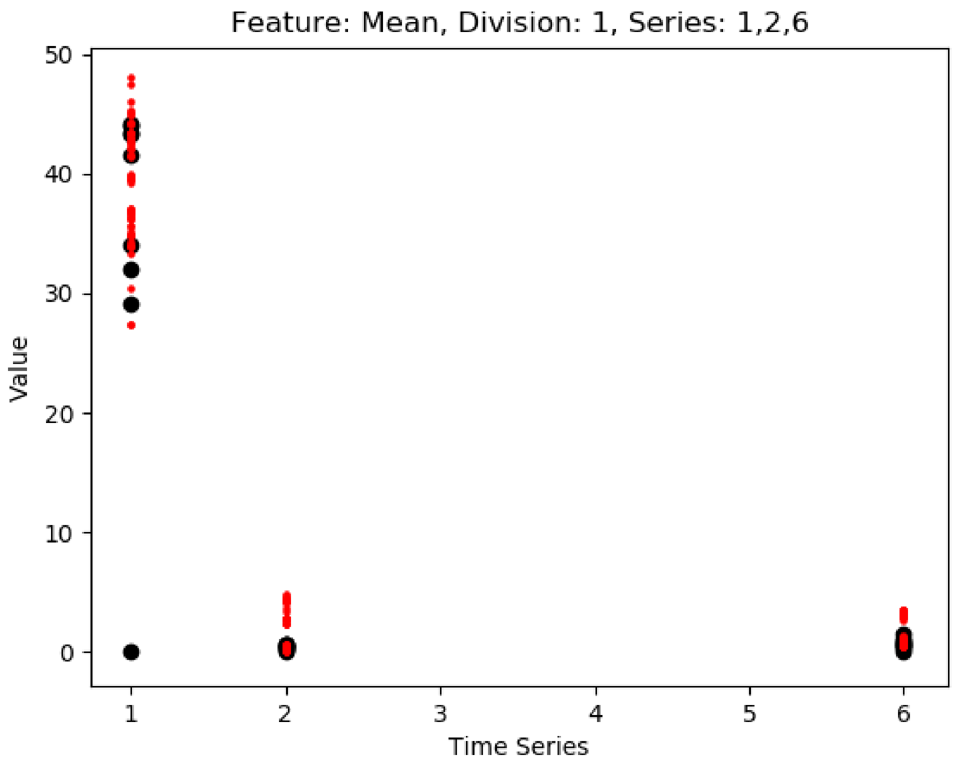
ii)

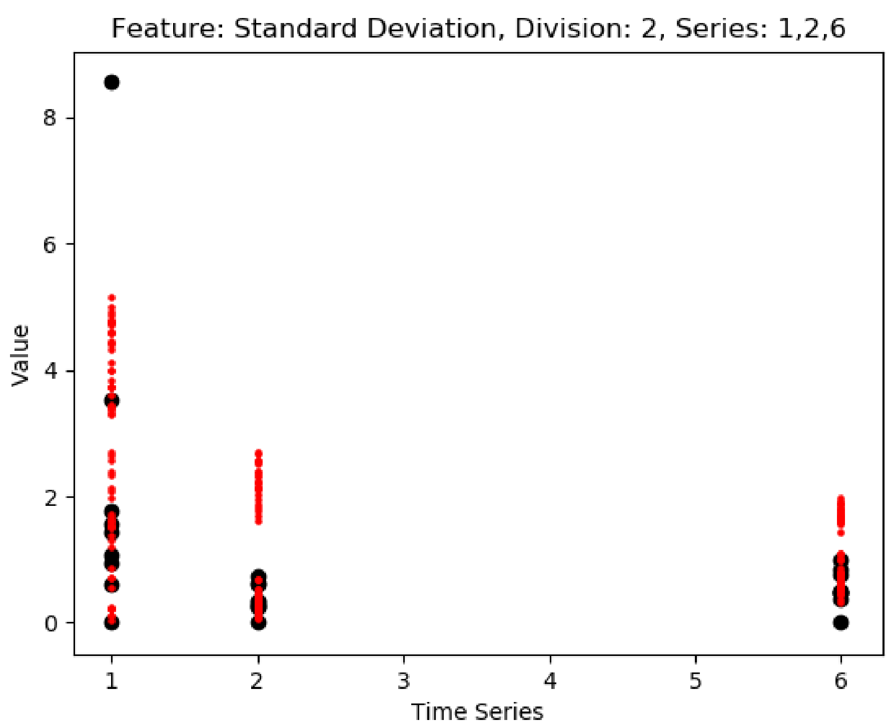


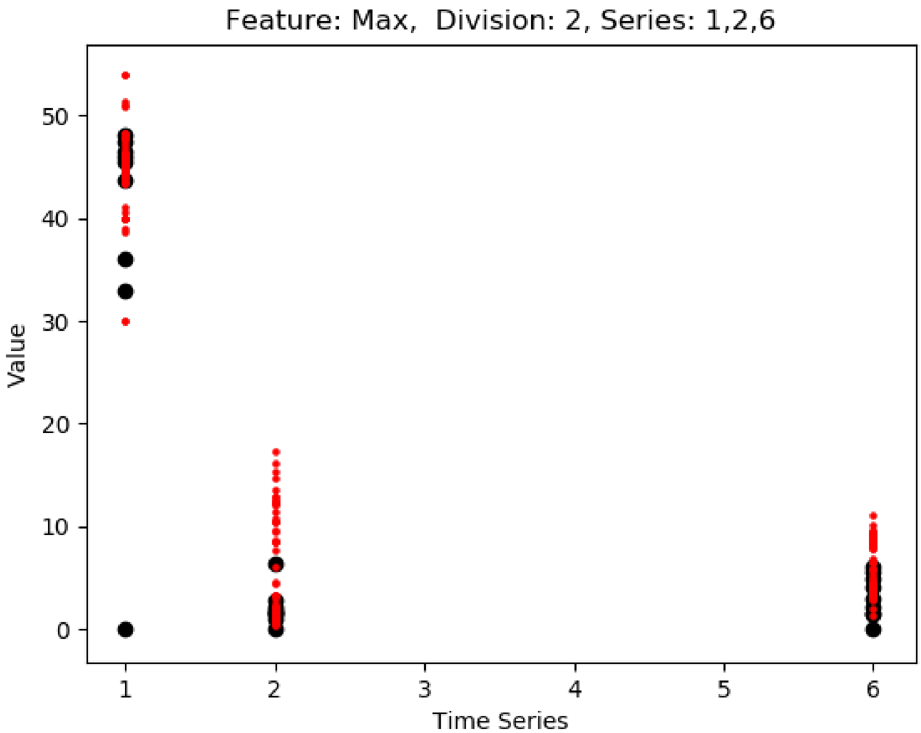
Feature: Standard Deviation, Division: 1, Series: 1,2,6
Value
Time Series
0
2
4
6
8
1
2
3
4
5
6

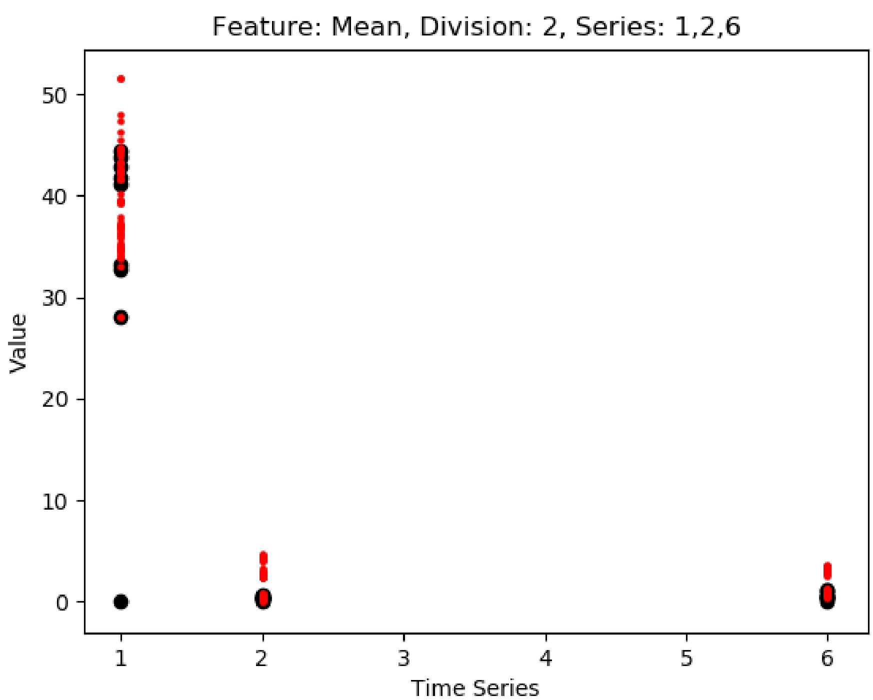


Feature: Max, Division: 1, Series: 1,2,6
Value
Time Series
0
10
20
30
40
50
1
2
3
4
5
6


Feature: Mean, Division: 1, Series: 1,2,6
Value
Time Series
0
10
20
30
40
50
1
2
3
4
5
6


Feature: Standard Deviation, Division: 2, Series: 1,2,6
Value
Time Series
0
2
4
6
8
1
2
3
4
5
6





I can observe considerable difference in scatter plots of the whole dataset and when it’s divided into two. Former gives a wider range of values while the latter gives a narrower range. Value concentration in the plots of mean, max and standard deviation change when the dataset it split.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


bending_std_series_1=[]

bending_std_series_1=feature_std_1[0:2]
bending_std_single_array=np.array(bending_std_series_1)
bending_std=bending_std_single_array.ravel()

bending_std_1_split=[]
for i in range(0,len(bending_std)):
    tmp_val=bending_std[i]
    for j in range(0,len(tmp_val)):
        bending_std_1_split.append(tmp_val[j])
bending_std_1_split_array=np.array(bending_std_1_split)

#other activities std
others_std_series_1=[]
others_std_series_1=feature_std_1[2:]
others_std_single_array=np.array(others_std_series_1)
others_std=others_std_single_array.ravel()

others_std_1_split=others_std

others_std_series_2=[]
others_std_series_2=feature_std_2[2:]
others_std_single_array=np.array(others_std_series_2)
others_std=others_std_single_array.ravel()

others_std_2_split=others_std

bending_std_series_2=[]

bending_std_series_2=feature_std_2[0:2]
bending_std_single_array=np.array(bending_std_series_2)
bending_std=bending_std_single_array.ravel()

bending_std_2_split=[]
for i in range(0,len(bending_std)):
    tmp_val=bending_std[i]
    for j in range(0,len(tmp_val)):
        bending_std_2_split.append(tmp_val[j])
bending_std_2_split_array=np.array(bending_std_2_split)

bending_std_series_6=[]

bending_std_series_6 = feature_std_6[0:2]
bending_std_single_array = np.array(bending_std_series_6)
bending_std = bending_std_single_array.ravel()

bending_std_6_split=[]
for i in range(0,len(bending_std)):
```

```
    tmp_val=bending_std[i]
    for j in range(0,len(tmp_val)):
        bending_std_6_split.append(tmp_val[j])
bending_std_6_split_array=np.array(bending_std_6_split)

others_std_series_6=[]
others_std_series_6=feature_std_6[2:]
others_std_single_array=np.array(others_std_series_6)
others_std=others_std_single_array.ravel()

others_std_6_split=others_std

bending_std_1_split_array= np.nan_to_num(bending_std_1_split_array)
others_std_1_split=np.nan_to_num(others_std_1_split)

bending_std_2_split_array=np.nan_to_num(bending_std_2_split_array)
others_std_2_split=np.nan_to_num(others_std_2_split)

bending_std_6_split_array=np.nan_to_num(bending_std_6_split_array)
others_std_6_split=np.nan_to_num(others_std_6_split)

#other activities max
others_max_series_1=[]
others_max_series_1=feature_max_1[2:]
others_max_single_array=np.array(others_max_series_1)
others_max=others_max_single_array.ravel()

others_max_1_split=others_max

bending_max_series_1=[]

bending_max_series_1=feature_max_1[0:2]
bending_max_single_array=np.array(bending_max_series_1)
bending_max=bending_max_single_array.ravel()
bending_max_1_split=[]
for i in range(0,len(bending_max)):
    tmp_val=bending_max[i]
    for j in range(0,len(tmp_val)):
        bending_max_1_split.append(tmp_val[j])
bending_max_1_split_array=np.array(bending_max_1_split)

bending_max_series_2=[]

bending_max_series_2=feature_max_2[0:2]
bending_max_single_array=np.array(bending_max_series_2)
bending_max=bending_max_single_array.ravel()

bending_max_2_split=[]
for i in range(0,len(bending_max)):
    tmp_val=bending_max[i]
    for j in range(0,len(tmp_val)):
        bending_max_2_split.append(tmp_val[j])
bending_max_2_split_array=np.array(bending_max_2_split)

bending_max_series_6=[]

bending_max_series_6=feature_max_6[0:2]
bending_max_single_array=np.array(bending_max_series_6)
bending_max=bending_max_single_array.ravel()

bending_max_6_split=[]
for i in range(0,len(bending_max)):
    tmp_val=bending_max[i]
```

```python
    for j in range(0,len(tmp_val)):
        bending_max_6_split.append(tmp_val[j])
bending_max_6_split_array=np.array(bending_max_6_split)

others_max_series_6=[]
others_max_series_6=feature_max_6[2:]
others_max_single_array=np.array(others_max_series_6)
others_max=others_max_single_array.ravel()

others_max_6_split=others_max

bending_max_1_split_array=np.nan_to_num(bending_max_1_split_array)
others_max_1_split=np.nan_to_num(others_max_1_split)

bending_max_2_split_array=np.nan_to_num(bending_max_2_split_array)
others_max_2_split=np.nan_to_num(others_max_2_split)

bending_max_6_split_array=np.nan_to_num(bending_max_6_split_array)
others_max_6_split=np.nan_to_num(others_max_6_split)


bending_mean_series_1=[]

bending_mean_series_1=feature_mean_1[0:2]
bending_mean_single_array=np.array(bending_mean_series_1)
bending_mean=bending_mean_single_array.ravel()

bending_mean_1_split=[]
for i in range(0,len(bending_mean)):
    tmp_val=bending_mean[i]
    for j in range(0,len(tmp_val)):
        bending_mean_1_split.append(tmp_val[j])
bending_mean_1_split_array=np.array(bending_mean_1_split)

others_max_series_2=[]
others_max_series_2=feature_max_2[2:]
others_max_single_array=np.array(others_max_series_2)
others_max=others_max_single_array.ravel()

others_max_2_split=others_max

#other activities mean
others_mean_series_1=[]
others_mean_series_1=feature_mean_1[2:]
others_mean_single_array=np.array(others_mean_series_1)
others_mean=others_mean_single_array.ravel()

others_mean_1_split=others_mean

bending_mean_series_2=[]

bending_mean_series_2=feature_mean_2[0:2]
bending_mean_single_array=np.array(bending_mean_series_2)
bending_mean=bending_mean_single_array.ravel()


bending_mean_2_split=[]
for i in range(0,len(bending_mean)):
    tmp_val=bending_mean[i]
    for j in range(0,len(tmp_val)):
        bending_mean_2_split.append(tmp_val[j])
bending_mean_2_split_array=np.array(bending_mean_2_split)
```

```
others_mean_series_2=[]
others_mean_series_2=feature_mean_2[2:]
others_mean_single_array=np.array(others_mean_series_2)
others_mean=others_mean_single_array.ravel()

others_mean_2_split=others_mean

bending_mean_series_6=[]

bending_mean_series_6=feature_mean_6[0:2]
bending_mean_single_array=np.array(bending_mean_series_6)
bending_mean=bending_mean_single_array.ravel()

bending_mean_6_split=[]
for i in range(0,len(bending_mean)):
    tmp_val=bending_mean[i]
    for j in range(0,len(tmp_val)):
        bending_mean_6_split.append(tmp_val[j])
bending_mean_6_split_array=np.array(bending_mean_6_split)

others_mean_series_6=[]
others_mean_series_6=feature_mean_6[2:]
others_mean_single_array=np.array(others_mean_series_6)
others_mean=others_mean_single_array.ravel()

others_mean_6_split=others_mean

bending_mean_1_split_array=np.nan_to_num(bending_mean_1_split_array)
others_mean_1_split=np.nan_to_num(others_mean_1_split)


bending_mean_2_split_array=np.nan_to_num(bending_mean_2_split_array)
others_mean_2_split=np.nan_to_num(others_mean_2_split)

bending_mean_6_split_array=np.nan_to_num(bending_mean_6_split_array)
others_mean_6_split=np.nan_to_num(others_mean_6_split)

fig, ax = plt.subplots()

plt.xlabel('Time Series')
plt.ylabel('Value')
plt.title('Feature: Standard Deviation, Division: '+str(counter)+', Series: 
1,2,6')
for i in range(0,len(bending_std_1_split_array)):
    ax.scatter(1,bending_std_1_split_array[i],  color='black', label='$x$')
for i in range(0,len(others_std_1_split)):
    ax.scatter(1,others_std_1_split[i], 5,color='red', label='$x$')

for i in range(0, len(bending_std_2_split_array)):
    ax.scatter(2,bending_std_2_split_array[i], color='black', label='$x$')
for i in range(0, len(others_std_2_split)):
    ax.scatter(2,others_std_2_split[i],  5,color='red', label='$x$')

for i in range(0, len(bending_std_6_split)):
    ax.scatter(6,bending_std_6_split_array[i],  color='black', label='$x$')
for i in range(0, len(others_std_6_split)):
    ax.scatter(6,others_std_6_split[i], 5,color='red', label='$x$')

plt.show()



fig, ax = plt.subplots()
```

```
plt.xlabel('Time Series')
plt.ylabel('Value')
plt.title('Feature: Max,  Division: '+str(counter)+', Series: 1,2,6')
for i in range(0,len(bending_max_1_split_array)):
    ax.scatter(1,bending_max_1_split_array[i],  color='black', label='$x$')
for i in range(0,len(others_max_1_split)):
    ax.scatter(1,others_max_1_split[i], 5,color='red', label='$x$')

for i in range(0, len(bending_max_2_split_array)):
    ax.scatter(2,bending_max_2_split_array[i], color='black', label='$x$')
for i in range(0, len(others_max_2_split)):
    ax.scatter(2,others_max_2_split[i],  5,color='red', label='$x$')

for i in range(0, len(bending_max_6_split)):
    ax.scatter(6,bending_max_6_split_array[i],  color='black', label='$x$')
for i in range(0, len(others_max_6_split)):
    ax.scatter(6,others_max_6_split[i], 5,color='red', label='$x$')

plt.show()



fig, ax = plt.subplots()

plt.xlabel('Time Series')
plt.ylabel('Value')
plt.title('Feature: Mean, Division: '+str(counter)+', Series: 1,2,6')
for i in range(0,len(bending_mean_1_split_array)):
    ax.scatter(1,bending_mean_1_split_array[i],  color='black', label='$x$')
for i in range(0,len(others_mean_1_split)):
    ax.scatter(1,others_mean_1_split[i], 5,color='red', label='$x$')

for i in range(0, len(bending_mean_2_split_array)):
    ax.scatter(2,bending_mean_2_split_array[i], color='black', label='$x$')
for i in range(0, len(others_mean_2_split)):
    ax.scatter(2,others_mean_2_split[i],  5,color='red', label='$x$')

for i in range(0, len(bending_mean_6_split)):
    ax.scatter(6,bending_mean_6_split_array[i],  color='black', label='$x$')
for i in range(0, len(others_mean_6_split)):
    ax.scatter(6,others_mean_6_split[i], 5,color='red', label='$x$')

plt.show()
```

iii) Different time-domain features were pruned for different values of l because dataset changed every time. Using SelectKBest of sklearn.feature_selection module, I pruned the features using their p-values.

The score for different lengths of datasets in as follows:

l = 1 :

  score = 0.9875

l = 2 :

  score = 0.9375

l = 3 :

  score = 0.904444444444

l = 4 :

score = 0.9875

l = 5 :

score = 0.966666666667

l = 6 :

score = 0.894444444444

l = 7 :

score = 0.938095238095

l = 8 :

score = 0.975

l = 9:

score = 0.888888888889

l = 10 :

score = 0.89546

As the score of l = 1 and 4 is highest, therefore I have chosen l=1 as the best value of l.

Following are the p-values of l=1 and l=4.

L= 1:

Division = 1:

Fold-1 score = 1.00000

p-values:

```
[ 3.92349208e-02  6.18793704e-04  7.41144085e-04  3.07791459e-03
  1.13343942e-03             nan  2.81880591e-04  2.79224872e-01
  4.16025220e-01  7.78354896e-02  5.82606166e-02             nan
  1.98207096e-10  3.63528323e-09  2.76529009e-28  7.70402802e-02
  1.34686726e-02  6.93303942e-01]
```

Fold-2 score = 1.00000

p-values:

```
[ 1.17993621e-02  2.14223590e-04  2.83710707e-03  2.97720532e-06
  2.19498330e-05             nan  1.39482986e-01  6.87769284e-01
  5.85553054e-04  2.23247360e-01  1.02569697e-03             nan
  4.06012864e-08  1.12079342e-06  4.58321576e-25  3.54359362e-04
  4.45003302e-04  7.07875580e-01]
```

Fold-3 score = 0.93750

p-values:

```
[ 6.24521028e-02  1.31965524e-03  3.01541196e-02  1.09623081e-06
```

1.10994891e-05 nan 8.23118121e-04 1.67578549e-01
2.25979327e-01 8.48642579e-01 1.27224713e-03 nan
3.76430645e-13 4.42240752e-11 1.96341189e-24 6.43635564e-03
5.29433280e-04 6.96354909e-01]

Fold-4 score = 1.00000
p-values:
[7.42926219e-01 6.32242697e-01 6.87074716e-01 6.06519155e-10
1.33673803e-05 nan 3.83573659e-05 3.01624484e-03
6.86134011e-01 8.03987221e-01 1.28857636e-03 nan
2.13093924e-15 1.10854530e-15 4.54785847e-41 1.28874086e-06
2.23218174e-04 6.84883030e-01]

Fold-5 score = 1.00000
p-values:
[8.42518241e-01 2.65772070e-01 4.33510339e-01 2.51597858e-03
2.69212313e-02 nan 6.66727786e-04 2.88556899e-03
2.52562142e-03 9.73825019e-01 9.33670187e-02 nan
2.19476571e-15 1.43100395e-19 7.07163278e-73 4.32744647e-03
1.18326240e-02 nan]

L-1 score =0.9875

L= 4:

Division = 1:

Fold-1 score = 0.75000
p-values:
[4.05459081e-01 1.29647097e-01 1.32288046e-03 6.40706448e-05
3.33077316e-02 nan 5.41115621e-02 1.11220191e-03
1.19167517e-02 4.43366743e-02 4.63619232e-02 nan
1.28161713e-07 1.02993831e-14 2.75848554e-33 5.89805775e-05
2.06133889e-02 nan]

Fold-2 score = 1.00000
p-values:
[3.97665038e-01 5.15792178e-01 6.62876310e-03 7.97907571e-04
2.77177740e-02 nan 8.34490625e-02 1.57423726e-02
1.38506718e-01 4.28928118e-02 5.44077638e-02 nan
1.30101339e-07 3.83990251e-09 1.19532975e-17 4.68197023e-04
2.90688331e-02 nan]

Fold-3 score = 1.00000
p-values:
```
[ 3.98774106e-01   5.44767360e-01   1.40981290e-02   1.11811428e-03
  3.41111644e-02              nan   7.59813879e-02   1.45458882e-02
  9.23265460e-02   6.27590719e-02   5.63945311e-02              nan
  4.83519766e-08   2.92934400e-09   1.41170032e-21   5.98292779e-04
  3.24984566e-02              nan]
```

Fold-4 score = 1.00000
p-values:
```
[ 4.28274197e-01   4.63348723e-01   1.49322597e-02   2.74171062e-04
  2.54661797e-02              nan   6.91276696e-02   9.13981963e-03
  1.15041007e-01   3.43149145e-02   3.99938144e-02              nan
  1.76347557e-08   2.29332377e-10   8.01122445e-24   5.65407327e-04
  3.12261664e-02              nan]
```

Fold-5 score = 1.00000
p-values:
```
[ 4.22901648e-01   2.99578124e-01   8.44447969e-03   9.36151116e-06
  7.45426884e-03              nan   3.61749077e-01   9.17575240e-02
  8.64129702e-01   6.43812558e-03   1.07558598e-02              nan
  1.93534500e-06   8.10336888e-07   1.72680068e-18   4.10430336e-05
  8.23683598e-03              nan]
```

Final Score = 0.9499999999999996

Division: 2
Fold-1 score = 1.00000
p values:
```
[ 2.92197511e-01   2.79791774e-01   9.06130788e-01   2.54364193e-01
  6.96086167e-01              nan   3.23709701e-04   1.27859716e-03
  5.42682697e-06   1.25372571e-02   6.26386391e-01              nan
  1.32488477e-07   2.75810087e-10   1.93211803e-33   1.63610863e-01
  5.80607482e-01              nan]
```

Fold-2 score = 1.00000
p-values:
```
[ 5.39040446e-01   4.30661283e-01   9.45290616e-01   9.31432013e-02
  7.20376537e-01              nan   3.52154645e-04   8.63133116e-04
  5.42682697e-06   2.05482044e-02   6.37970012e-01              nan
```

1.06426819e-06   9.76192005e-08   4.44820923e-29   1.12055218e-01
5.89958499e-01   nan]

Fold-3 score =: 1.00000
p-values:
[4.27329363e-01   4.45255441e-01   7.46432209e-01   3.03322001e-01
6.96695098e-01   nan   2.37395554e-04   9.25828693e-04
5.42682697e-06   8.71569616e-03   5.68130163e-01   nan
7.98041003e-08   3.41655273e-09   4.44820923e-29   1.86514643e-01
5.98851310e-01   nan]

Fold-4 score = 1.00000
p-values:
[2.48732517e-01   2.99624491e-01   8.75043484e-01   3.06146319e-01
8.57402661e-01   nan   1.52915866e-05   1.44047430e-04
2.86227487e-11   6.50372858e-04   5.99460101e-01   nan
9.52927285e-08   3.74589259e-08   4.87412302e-24   8.92493925e-01
8.26766624e-01   nan]

Fold-5 score = 1.00000
p-values for:
[2.58781090e-01   3.84622093e-01   8.30649804e-01   1.47369657e-01
7.38170934e-01   nan   2.28395955e-06   4.93194024e-06
3.01868888e-09   7.93062810e-04   5.98386999e-01   nan
3.55398344e-08   3.81585476e-10   8.25273018e-24   6.22625549e-01
7.57687482e-01   nan]

Final Score = 1.0

Division: 3
Fold-1 score = 1.00000
p-values:
[5.31300657e-01   8.29666352e-02   1.24506149e-02   8.83341652e-05
8.73201722e-01   nan   7.07717623e-02   9.00986540e-01
3.43070854e-02   5.97086044e-01   7.72347585e-01   nan
4.36914379e-04   2.30735683e-03   8.88383083e-09   7.09948457e-03
9.47288970e-01   nan]

Fold-2 score = 1.00000
p-values:
[6.39240319e-01   8.13306310e-02   2.47878831e-02   5.82813705e-04

```
  8.85197134e-01             nan   7.68234377e-02   6.49605497e-01
  4.42817126e-02   5.20232902e-01   7.74907831e-01             nan
  5.09713702e-04   4.39454203e-03   1.71347208e-07   7.04846120e-03
  9.86469000e-01             nan]
```

Fold-3 score = 1.00000

P-values:

```
[  6.74370086e-01   4.33350741e-01   1.89513502e-01   7.00096878e-02
  9.49183283e-01             nan   1.29212696e-01   7.38351505e-01
  3.73176475e-01   8.39890807e-01   9.49225228e-01             nan
  3.25169747e-03   1.89980961e-03   8.47258623e-12   6.92489442e-01
  7.56169774e-01             nan]
TRAIN: [ 0  1  2  3  4  5  6  7  8  9 10 11 16 17 18 19]
TEST: [12 13 14 15]
```

Fold-4 score = 1.00000

p-values:

```
[  5.19664942e-01   2.46668590e-01   3.38972368e-02   8.99060709e-04
  9.53101087e-01             nan   9.74185054e-02   7.58832571e-01
  2.18354691e-01   8.04889158e-01   9.34659941e-01             nan
  1.37059105e-03   1.70096780e-03   6.32909145e-11   1.08991573e-01
  8.50330528e-01             nan]
```

Fold-5 score = 1.00000

p-values:

```
[  6.82431158e-01   4.20932344e-01   2.97538522e-01   1.17820516e-01
  9.59763682e-01             nan   1.19282133e-01   9.76120343e-01
  2.86536795e-01   9.88818822e-01   9.08824666e-01             nan
  3.80797442e-03   2.43796545e-03   2.19097120e-11   6.33472382e-01
  7.65236368e-01             nan]
```

Final Score = 1.0

Division = 4:

Fold-1 score: 1.00000

p-values:

```
 [  3.51358901e-03   1.25236608e-03   5.34624809e-02   7.17949368e-07
  2.75230950e-05             nan   3.07433538e-01   6.94086982e-02
  1.12160592e-08   9.85925753e-01   2.05609279e-03             nan
  7.80099495e-02   5.65833180e-01   2.26255210e-05   1.42556334e-02
  4.50363359e-03             nan]
```

Fold-2 score: 1.00000
p-values:
```
[ 4.34318799e-03  9.89278773e-05  3.93545305e-04  2.05404199e-04
  4.30014580e-04             nan  2.05820711e-01  1.38046953e-01
  3.11466823e-07  4.65088349e-01  1.16028128e-02             nan
  3.59146407e-02  6.63947204e-01  6.69554135e-04  9.34366028e-02
  2.22736655e-02             nan]
```

Fold-3 score =1.00000
p-values:
```
[ 6.86729235e-11  4.86993356e-11  2.72814757e-08  3.39854975e-06
  7.48561187e-04             nan  1.32542073e-02  3.43591819e-04
  5.87039425e-06  1.35143253e-01  8.25925412e-03             nan
  4.45251575e-02  2.66886684e-03  3.18531459e-04  1.90794201e-03
  1.08079690e-02             nan]
```

Fold-4 score: 1.00000
p-values:
```
[ 1.45065794e-08  8.41147241e-10  9.80453067e-08  1.03142620e-05
  3.05778382e-04             nan  1.86894882e-01  2.29139518e-03
  8.64233790e-07  4.65051124e-01  6.35676129e-03             nan
  5.28550878e-01  2.21850407e-02  1.81102268e-04  4.99631716e-03
  9.96601414e-03             nan]
```

Fold-5 score = 1.00000
p-value:
```
[ 1.05047625e-08  1.08296055e-09  2.65705147e-07  4.17780703e-06
  2.92889515e-04             nan  1.74909514e-01  2.20115585e-03
  5.95342284e-07  4.48900034e-01  5.83821601e-03             nan
  4.47890784e-01  2.08962469e-02  1.68218896e-04  8.21181587e-03
  9.86514109e-03             nan]
```
Final Score = 1.0

L-4 score = 0.9875

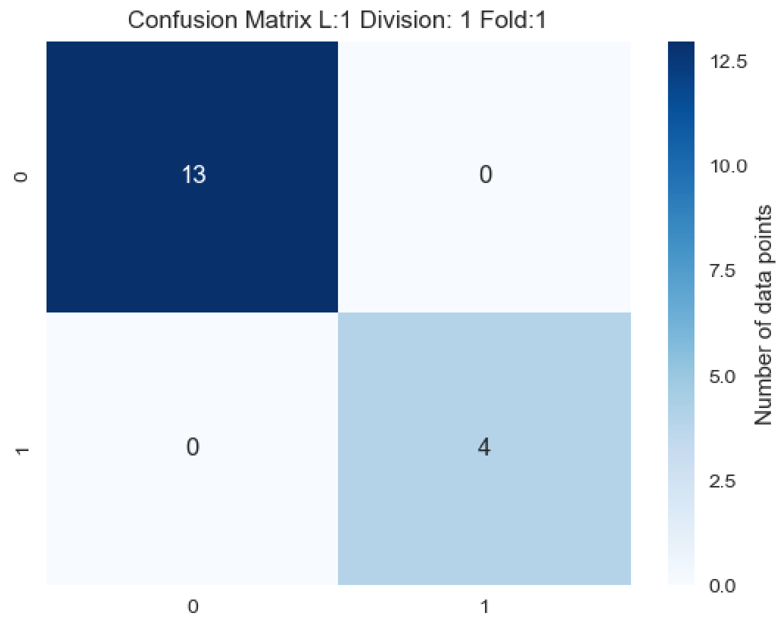
Right way to perform cross-validation:
We can select/prune variables/features before cross-validation using the whole dataset. But the right way to do it is to select/prune variables/features while k-fold cross-validation inside the loop because the dataset changes in every fold and hence the significance of each feature.

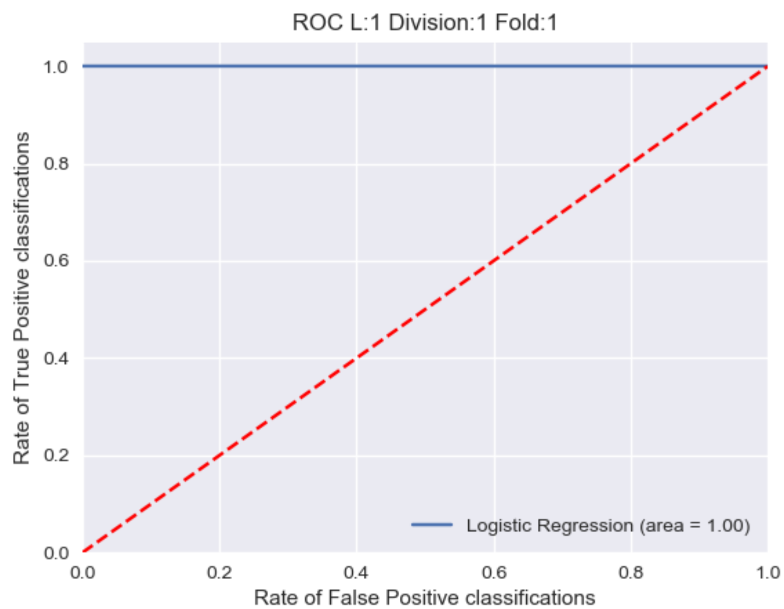
Code:

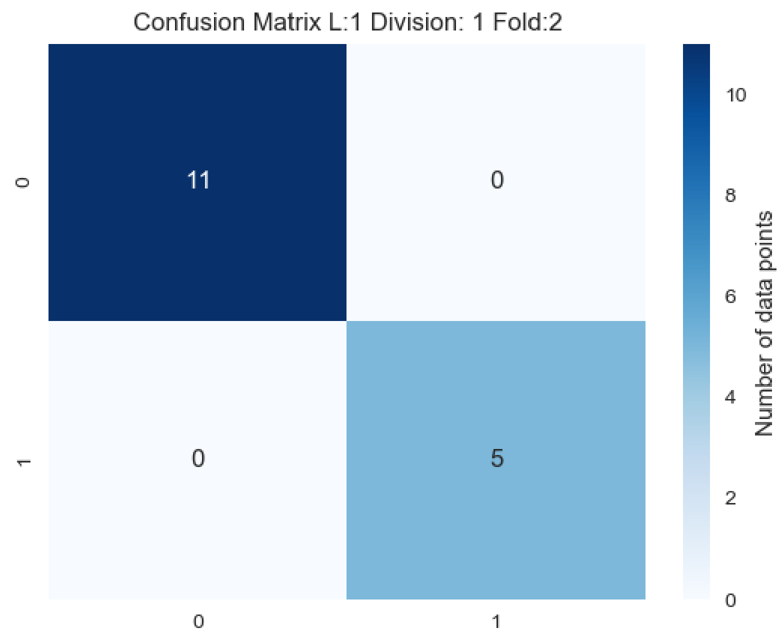
```
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest

data_size = 69
index = 1
count = 0
division = 1
division_size = data_size / division

for l in range(1,11):
    division_scores = []
    scores = []
    for div in range(1,l+1):
        count = count + 1

        data = pd.read_csv('C:/Users/Sarah
Riaz/Documents/ML/HW/HW2/AReM/Arem/AReM/my_dataset.csv').values
        x_train = data[:, 0:18]
        y_train = data[:, 18:]

        division_size = int(len(x_train) / l)
        index_end = int((division_size * count))
        index_start = int(index_end - division_size)

        divided_x_train = x_train[index_start:index_end, 0:18]
        divided_y_train = y_train[index_start:index_end, 0:18]

        k_fold = KFold(n_splits=5)
        k_fold.get_n_splits(divided_x_train)
        KFold(n_splits=2, random_state=7, shuffle=True)

        modelCV = LogisticRegression()
        x_pruned = SelectKBest(chi2, k=6).fit_transform(divided_x_train, divided_y_train)
        scores = []

        for k, (k_train_index, k_test_index) in enumerate(k_fold.split(x_pruned, divided_y_train)):

            x_train_fold, x_test_fold = divided_x_train[k_train_index], divided_x_train[k_test_index]
            y_train_fold, y_test_fold = divided_y_train[k_train_index], divided_y_train[k_test_index]

            modelCV.fit(divided_x_train[k_train_index], divided_y_train[k_train_index])

            prediction = modelCV.predict(x_test_fold)
            extra_data, p_values = chi2(x_train_fold, y_train_fold)

            scores.append(modelCV.score(divided_x_train[k_test_index], divided_y_train[k_test_index]))

        division_scores.append(np.mean(scores))
```

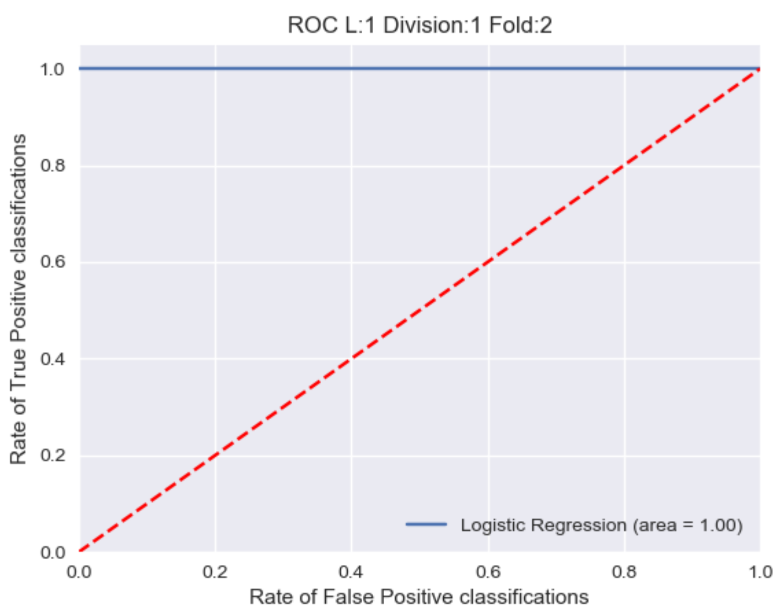
iv)

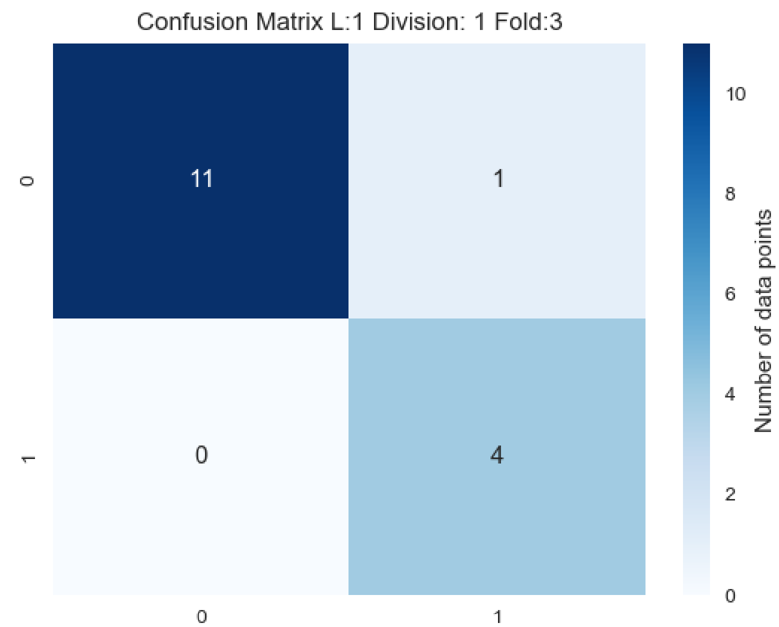
Confusion matrix and ROC:

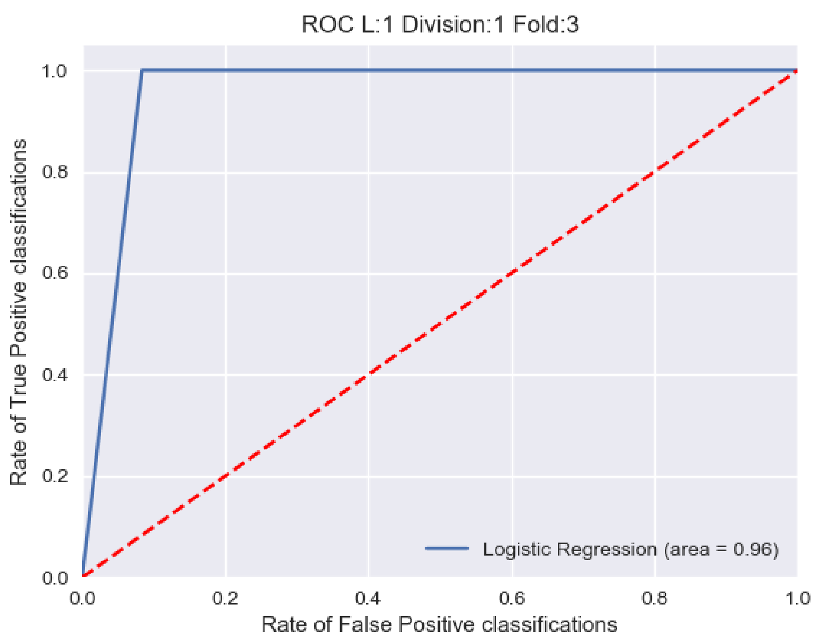


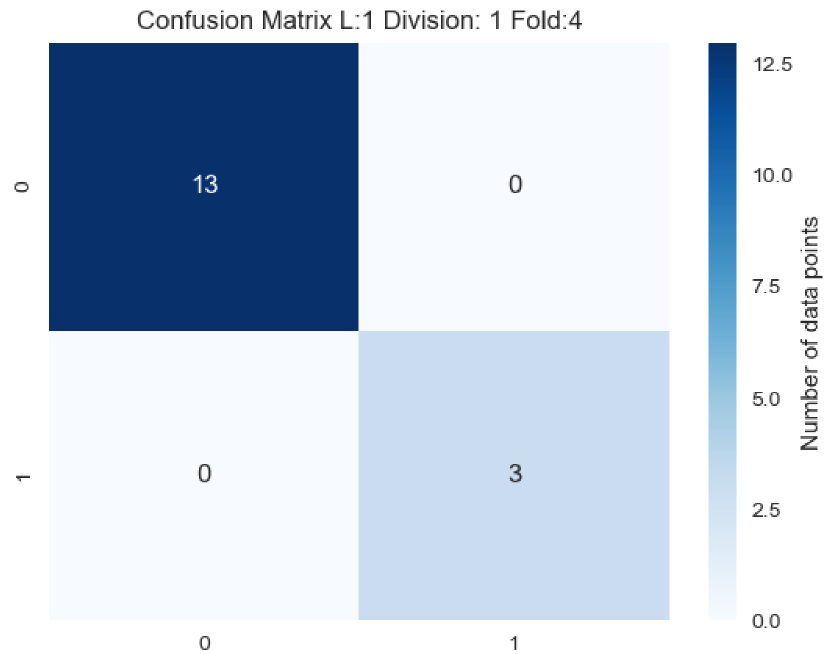



Confusion Matrix L:1 Division: 1 Fold:2
11
0
0
5
0
1
0
1
10
8
6
4
2
0
Number of data points

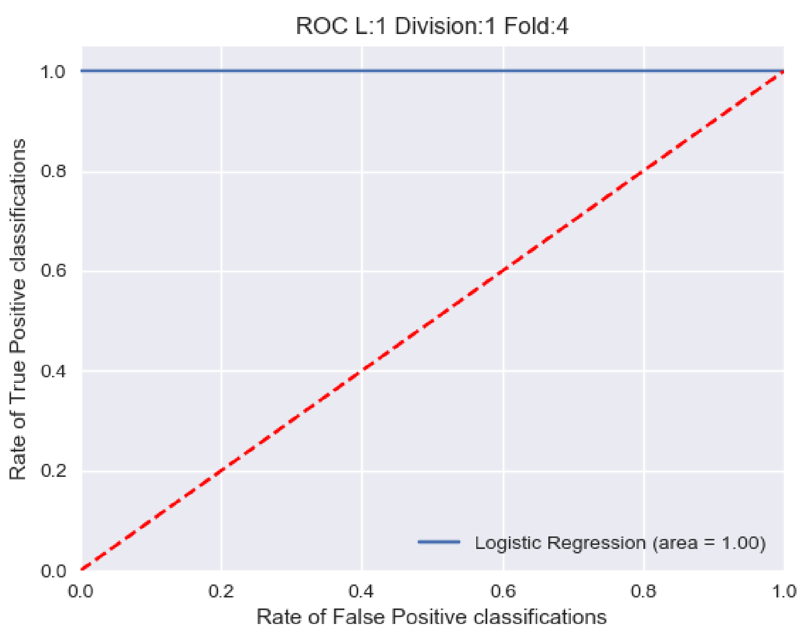

ROC L:1 Division:1 Fold:2
1.0
0.8
0.6
0.4
0.2
0.0
Rate of True Positive classifications
Logistic Regression (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0
Rate of False Positive classifications


Confusion Matrix L:1 Division: 1 Fold:3
11
1
0
4
0
1
0
1
10
8
6
4
2
0
Number of data points

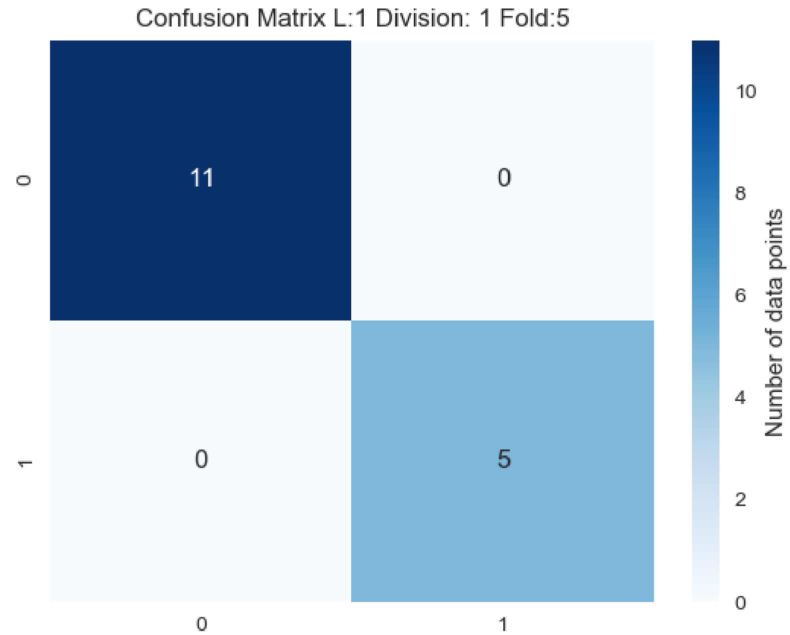

ROC L:1 Division:1 Fold:3
Logistic Regression (area = 0.96)
Rate of True Positive classifications
Rate of False Positive classifications
0.0
0.2
0.4
0.6
0.8
1.0

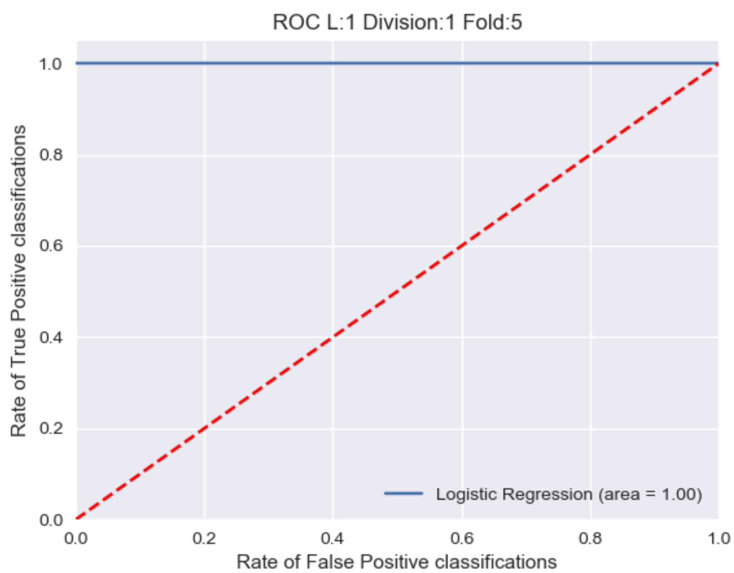


Confusion Matrix L:1 Division: 1 Fold:4
13
0
0
3
0
1
0
1
12.5
10.0
7.5
5.0
2.5
0.0
Number of data points



ROC L:1 Division:1 Fold:4
Rate of True Positive classifications
Rate of False Positive classifications
Logistic Regression (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0





Logistic regression β_i's and p-values:

L= 1:

Fold-1 score = 1.00000

Regression β_i's:

[[-1.88989006e-01 -3.31441790e-01 -1.87278877e-01 -3.84869958e-01
-1.62155439e-01 0.00000000e+00 1.47912261e-01 -2.62967806e-01
-5.25356509e-02 1.46077039e-01 -1.10366775e-01 0.00000000e+00

```
  5.39161705e-01  4.25511556e-01  7.06919125e-01 -3.79192664e-02
 -1.26477892e-01 -4.01018520e-04]]
p-values:
[ 3.92349208e-02  6.18793704e-04  7.41144085e-04  3.07791459e-03
  1.13343942e-03             nan  2.81880591e-04  2.79224872e-01
  4.16025220e-01  7.78354896e-02  5.82606166e-02             nan
  1.98207096e-10  3.63528323e-09  2.76529009e-28  7.70402802e-02
  1.34686726e-02  6.93303942e-01]

Fold-2 score = 1.00000
Regression βᵢ's:
 [[ -8.57949245e-03 -2.71480147e-01 -2.10362480e-01 -4.08587873e-01
  -1.06325146e-01  0.00000000e+00  3.49075579e-02 -3.38705745e-01
  -3.16875295e-01 -7.04519160e-02 -9.73807825e-02  0.00000000e+00
   5.91547740e-01  2.16684833e-01  5.15500706e-01 -1.08906042e-01
  -1.09140024e-01 -1.34679887e-04]]
p-values:
[ 1.17993621e-02  2.14223590e-04  2.83710707e-03  2.97720532e-06
  2.19498330e-05             nan  1.39482986e-01  6.87769284e-01
  5.85553054e-04  2.23247360e-01  1.02569697e-03             nan
  4.06012864e-08  1.12079342e-06  4.58321576e-25  3.54359362e-04
  4.45003302e-04  7.07875580e-01]

Fold-3 score = 0.93750
Regression βᵢ's:
 [[ -8.60317827e-02 -4.95600069e-01 -8.05139997e-02 -4.07712864e-01
  -2.27954727e-01  0.00000000e+00  1.34393847e-01 -3.70948694e-02
  -7.65638176e-02 -4.42610754e-04 -2.40128948e-01  0.00000000e+00
   4.92214706e-01  5.36217667e-01  2.54024642e-01 -1.67867645e-01
  -2.44817352e-01 -1.06960333e-04]]
p-values:
[ 6.24521028e-02  1.31965524e-03  3.01541196e-02  1.09623081e-06
  1.10994891e-05             nan  8.23118121e-04  1.67578549e-01
  2.25979327e-01  8.48642579e-01  1.27224713e-03             nan
  3.76430645e-13  4.42240752e-11  1.96341189e-24  6.43635564e-03
  5.29433280e-04  6.96354909e-01]

Fold-4 score = 1.00000
Regression βᵢ's:
 [[ -1.44120829e-01 -3.31059872e-01 -1.67794732e-01 -3.96498037e-01
  -1.21723519e-01  0.00000000e+00  8.68105544e-02 -2.12166139e-01
```

```
  2.46895622e-02  1.35151057e-01 -1.23383255e-01  0.00000000e+00
  6.15868357e-01  3.47078968e-01  5.85349645e-01 -4.25632700e-01
 -1.46550334e-01 -1.00683338e-04]]
p-values:
[ 7.42926219e-01  6.32242697e-01  6.87074716e-01  6.06519155e-10
  1.33673803e-05             nan  3.83573659e-05  3.01624484e-03
  6.86134011e-01  8.03987221e-01  1.28857636e-03             nan
  2.13093924e-15  1.10854530e-15  4.54785847e-41  1.28874086e-06
  2.23218174e-04  6.84883030e-01]

Fold-5 score = 1.00000
Regression βi's:
 [[-0.06740508 -0.43109773 -0.18277607 -0.4001777  -0.16158786  0.
  0.0316608  -0.26589794 -0.0574274   0.06066791 -0.17379993  0.
  0.64658364  0.44452797  0.6259898  -0.19496637 -0.19474349  0.        ]]
p-values:
[ 8.42518241e-01  2.65772070e-01  4.33510339e-01  2.51597858e-03
  2.69212313e-02             nan  6.66727786e-04  2.88556899e-03
  2.52562142e-03  9.73825019e-01  9.33670187e-02             nan
  2.19476571e-15  1.43100395e-19  7.07163278e-73  4.32744647e-03
  1.18326240e-02             nan]
```

Code:

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import chi2
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import KFold

count = 0
index = 1
Divisions = 1

for l in range(1,2):
    division_scores = []
    scores = []
    count = 0
    for div in range(1, l+1):
        count = count + 1

        data = pd.read_csv('C:/Users/Sarah 
Riaz/Documents/ML/HW/HW2/AReM/Arem/AReM/my_dataset.csv').values
```

```
    x_train = data[:, 0:18]
    y_train = data[:, 18:]

    division_size = int(len(x_train) / l)
    end_index=int((division_size * count))
    start_index=int(end_index - division_size)
    divided_x_train = x_train[start_index:end_index, 0:18]
    divided_y_train = y_train[start_index:end_index, 0:18]

    k_fold = KFold(n_splits=5)
    k_fold.get_n_splits(divided_x_train)
    KFold(n_splits=5, random_state=7, shuffle=True)
    logistic_reg_model = LogisticRegression()
    x_pruned = SelectKBest(chi2, k=6).fit_transform(divided_x_train, divided_y_train)

    scores = []

    for k, (k_train_index, k_test_index) in enumerate(k_fold.split(x_pruned, divided_y_train)):
        x_train_fold, x_test_fold = divided_x_train[k_train_index], divided_x_train[k_test_index]
        y_train_fold, y_test_fold = divided_y_train[k_train_index], divided_y_train[k_test_index]
        logistic_reg_model.fit(divided_x_train[k_train_index], divided_y_train[k_train_index])

        prediction = logistic_reg_model.predict(x_test_fold)

        scores.append(logistic_reg_model.score(divided_x_train[k_test_index],
divided_y_train[k_test_index]))

        extra_info, p_values = chi2(x_train_fold, y_train_fold)
        conf_matrix = confusion_matrix(y_test_fold, prediction)

        plt.figure()

        sns.set()
        conf_map = sns.heatmap(conf_matrix, cmap="Blues", cbar_kws={'label': 'Number of data points'},
annot=True)
        conf_title = 'Confusion Matrix L:' + str(l) + ' Division: ' + str(div) + ' Fold:' + str(k + 1)
        conf_map.set_title(conf_title)
        plt.show()

        logistic_roc_area = roc_auc_score(y_test_fold, prediction)
        fp, tp, thresholds = roc_curve(y_test_fold, prediction)
        plt.figure()

        plt_title = 'ROC L:' + str(l) + ' Division:' + str(div) + ' Fold:' + str(k+1)
        plt.title(plt_title)

        plt.plot(fp, tp, label='Logistic Regression (area = %0.2f)' % logistic_roc_area)
        plt.plot([0, 1], [0, 1], 'r--')
        plt.ylabel('Rate of True Positive classifications')
        plt.xlabel('Rate of False Positive classifications')
        plt.ylim([0.0, 1.05])
        plt.xlim([0.0, 1.0])

        plt.legend(loc="lower right")
        plt.show()
    division_scores.append(np.mean(scores))
```

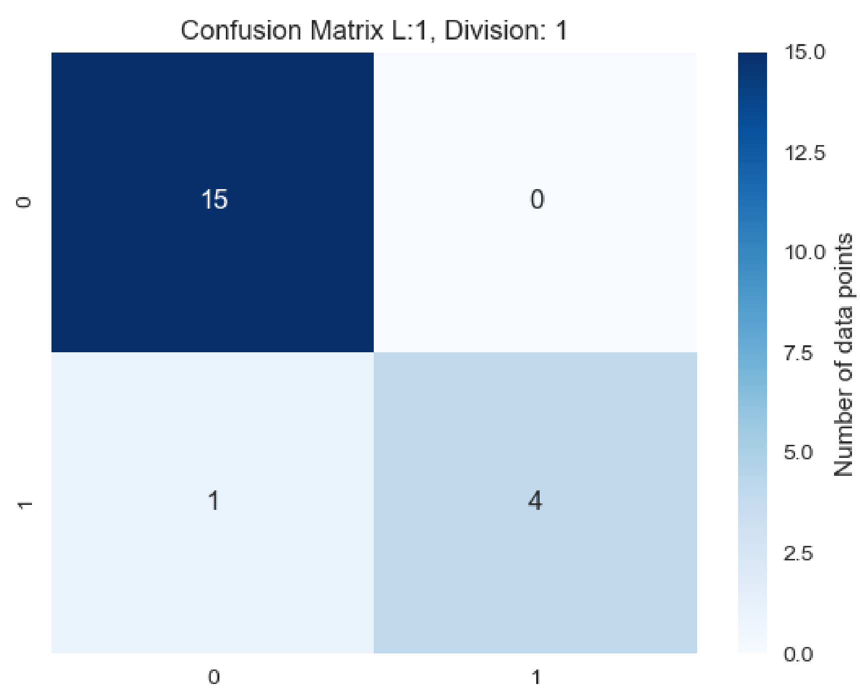
V)
Using the pruned set of time domain features and best value of l i.e. 1:

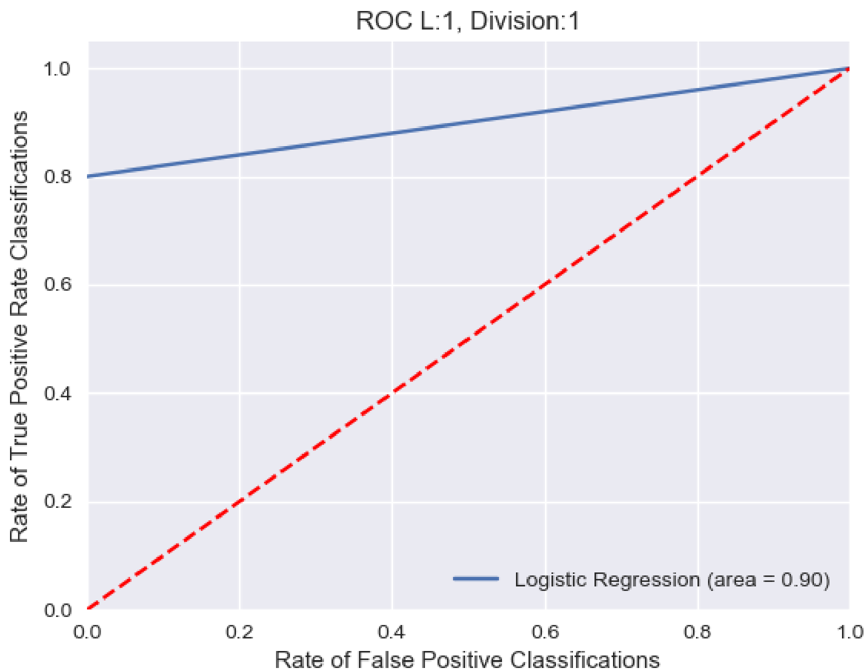
Testing (L=1):
Accuracy = 0.95

Training (L=1):
Score = 0.965

As both the scores are nearly same, we can conclude that our classifier performed well on the testing data. (Code is given below)

Vi and vii)





Only one of the classifications is misclassified. I have got AUC score of 0.90 which means there is 20% overlap in the data. So, I can conclude that there is inseparability among the classes to some extent because
higher overlapping is because of non-separability among classes.
Using randomundersampler in scikitlearn to under-sample the class with majority data points, I have tried to reduce the imbalance in the data because bending has significantly more data that others.

Code:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

data_size = 69
division = 1
count = 0
index = 1
value = data_size / division

for l in range(1,2):
    count = 0
    avg_divisions = []
    scores = []
    for inner in range(1,l+1):

        count = count + 1
```

```
        dataset=pd.read_csv('C:/Users/Sarah
Riaz/Documents/ML/HW/HW2/AReM/Arem/AReM/my_dataset_v_test.csv').values
        df=pd.read_csv('C:/Users/Sarah
Riaz/Documents/ML/HW/HW2/AReM/Arem/AReM/my_dataset_v_train.csv')

        dfset=df.values

        x_shuf=dataset[:,0:6]
        y_shuf=dataset[:,6:]
        value = int(len(dataset) / l)

        end_index=int((value * count))
        start_index=int(end_index - value)

        x_shuf_train = dfset[:, 0:6]
        y_shuf_train = dfset[:, 6:]
        value_train = int(len(df) / l)

        end_train_index = int((value_train * count))
        start_train_index = int(end_train_index - value_train)

        x_train_prime, y_train_prime = x_shuf_train, y_shuf_train
        x_test_prime,y_test_prime=x_shuf,y_shuf

        x_train= x_train_prime[start_train_index:end_train_index, 0:6]
        y_train= y_train_prime[start_train_index:end_train_index, 0:6]


        x_test= x_test_prime[start_index:end_index, 0:6]
        y_test= y_test_prime[start_index:end_index, 0:6]


        modelCV = LogisticRegression()
        scoring = 'accuracy'
        results = modelCV.fit(x_train,y_train)
        prediction =modelCV.predict(x_test)
        score = modelCV.score(x_test,y_test)
        scores.append(mean_squared_error(y_test, prediction))

        conf_matrix = confusion_matrix(y_test, prediction)
        plt.figure()
        sns.set()
        conf_map = sns.heatmap(conf_matrix, cmap="Blues", cbar_kws={'label':
'Number of data points'}, annot=True)
        conf_title = 'Confusion Matrix L:' + str(l) + ', Division: ' +
str(inner)
        conf_map.set_title(conf_title)
        plt.show()

        logit_roc_auc = roc_auc_score(y_test, prediction)
        fpr, tpr, thresholds = roc_curve(y_test, prediction)
        plt.figure()
        plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' %
logit_roc_auc)
        plt.plot([0, 1], [0, 1], 'r--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('Rate of False Positive Classifications')
        plt.ylabel('Rate of True Positive Rate Classifications')
        title = 'ROC L:' + str(l) + ', Division:' + str(inner)
        plt.title(title)
        plt.legend(loc="lower right")
        plt.show()
```

```
avg_divisions.append(np.mean(scores))
```

e)

We get the best score when L = 1.

L = {1, 2, …., 10}

L:1

Fold-1 score = 0.9286

Fold-2 score = 1

Fold-3 score = 1

Fold-4 score = 0.857

Fold-5 score = 0.846

Avg. Score = 0.9263736264

Final Scores:

L:1 = 0.9663736264

L:2 = 0.8761904762

L:3 = 0.9167

L:4 = 0.867

L:5 = 0.797

L:6 = 0.9164575624

L:7 = 0.8875964879

L:8 = 0.824598

L:9 = 0.7823236

L:10 = 0.81257

ii)

Training accuracy with p-values = 0.9625

L-1 penalization accuracy = 0.9663736264

Accuracy of L-1 penalization is better, although slightly. It also prunes the features pruning is done by giving a regression coefficient of zero to the features that are not significant for classification. Therefore, I'd say L-1 penalization is better and easier to implement.

Code:

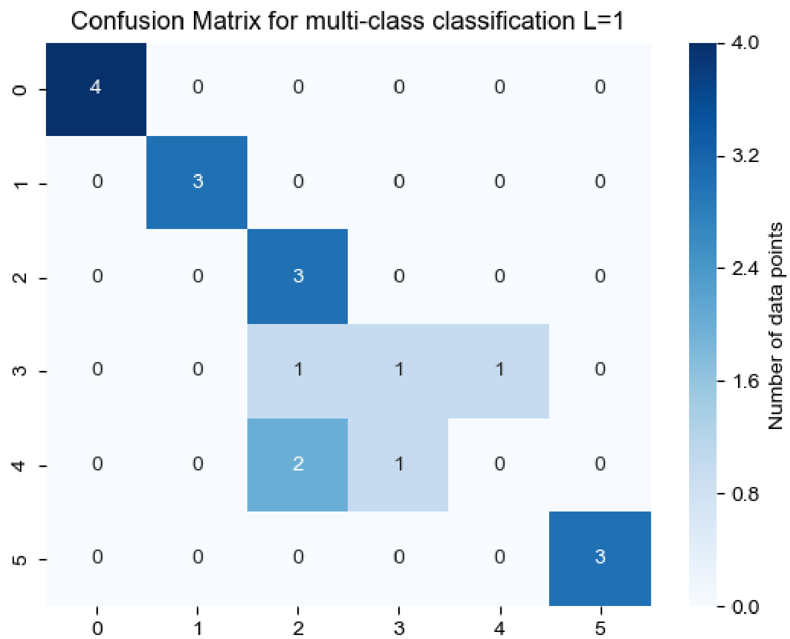
Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest

data_size = 69
index = 1
count = 0
division = 1
division_size = data_size / division

for l in range(1,11):
  division_scores = []
  scores = []
  for div in range(1,l+1):
    count = count + 1

    data = pd.read_csv('C:/Users/Sarah
Riaz/Documents/ML/HW/HW2/AReM/Arem/AReM/my_dataset.csv').values
    x_train = data[:, 0:18]
    y_train = data[:, 18:]

    division_size = int(len(x_train) / l)
    index_end = int((division_size * count))
    index_start = int(index_end - division_size)

    divided_x_train = x_train[index_start:index_end, 0:18]
    divided_y_train = y_train[index_start:index_end, 0:18]

    k_fold = KFold(n_splits=5)
    k_fold.get_n_splits(divided_x_train)
    KFold(n_splits=2, random_state=7, shuffle=True)

    model=LogisticRegressionCV(penalty='l1',Cs=10,cv=5,solver='liblinear')

    x_pruned = SelectKBest(chi2, k=6).fit_transform(divided_x_train, divided_y_train)
    scores = []

    for k, (k_train_index, k_test_index) in enumerate(k_fold.split(x_pruned, divided_y_train)):

      x_train_fold, x_test_fold = divided_x_train[k_train_index], divided_x_train[k_test_index]
      y_train_fold, y_test_fold = divided_y_train[k_train_index], divided_y_train[k_test_index]

      model.fit(divided_x_train[k_train_index], divided_y_train[k_train_index])

      prediction = model.predict(x_test_fold)
      extra_data, p_values = chi2(x_train_fold, y_train_fold)

      scores.append(modelCV.score(divided_x_train[k_test_index], divided_y_train[k_test_index]))

    division_scores.append(np.mean(scores))
```

f)

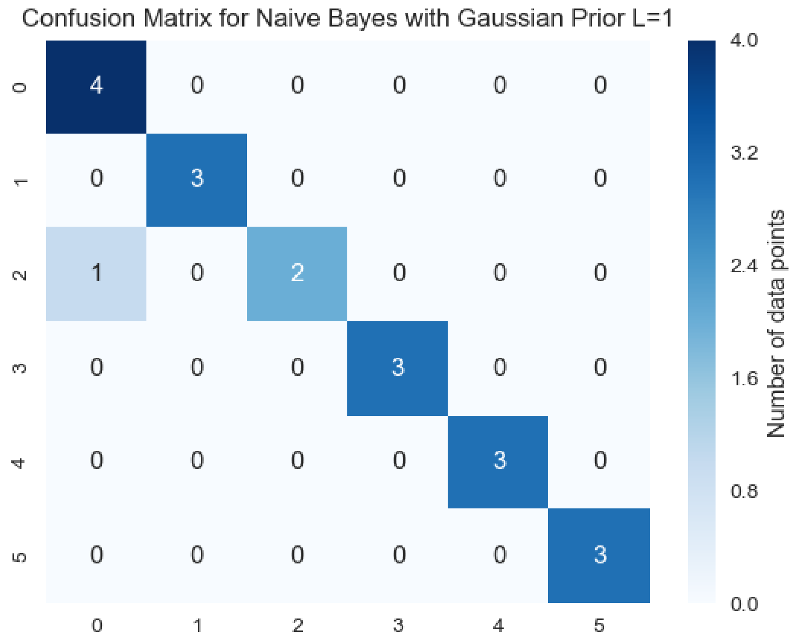
i)

Test error is 0.26316 and accuracy is 0.736842105263

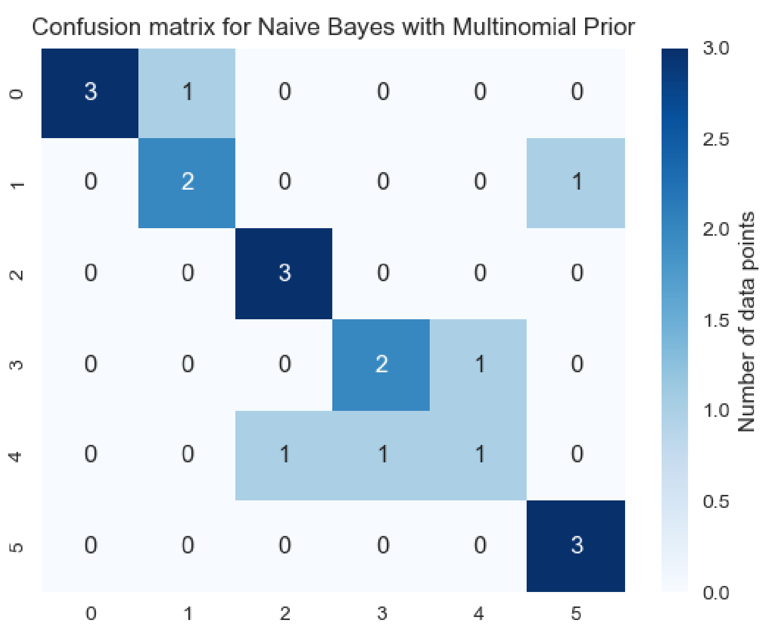


ii)

Score of Gaussian Naive Bayes for L= 1 is 0.947368421053



Score of Naive Bayes with Multinomial classifier for L= 1 is 0.736842105263



iii)

As it is evident from the scores of these classifiers Naïve Bayes classifier is better than L1-penalized multinomial regression. Gaussian Naive Bayes has the best score of 0.947368421053. Therefore, it performs best on this dataset.

Code:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import chi2
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import KFold
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
```

```
count = 0
index = 1
Divisions = 1

for l in range(1,2):
  division_scores = []
  scores = []
  count = 0
  for div in range(1, l+1):
    count = count + 1

    data = pd.read_csv('C:/Users/Sarah Riaz/Documents/ML/HW/HW2/AReM/Arem/AReM/my_dataset.csv').values
    x_train = data[:, 0:18]
    y_train = data[:, 18:]

    division_size = int(len(x_train) / l)
    end_index=int((division_size * count))
    start_index=int(end_index - division_size)
    divided_x_train = x_train[start_index:end_index, 0:18]
    divided_y_train = y_train[start_index:end_index, 0:18]

    k_fold = KFold(n_splits=5)
    k_fold.get_n_splits(divided_x_train)
    KFold(n_splits=5, random_state=7, shuffle=True)

# for L1 penalization
model=LogisticRegression(penalty='l1',solver='saga',multi_class='multinomial')

# for Naïve Bayes with gaussian prior
model=GaussianNB()
```

```
# for Naïve Bayes with multi-nomial prior
model=MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)


  x_pruned = SelectKBest(chi2, k=6).fit_transform(divided_x_train, divided_y_train)

  scores = []

  for k, (k_train_index, k_test_index) in enumerate(k_fold.split(x_pruned, divided_y_train)):
    x_train_fold, x_test_fold = divided_x_train[k_train_index], divided_x_train[k_test_index]
    y_train_fold, y_test_fold = divided_y_train[k_train_index], divided_y_train[k_test_index]
    logistic_reg_model.fit(divided_x_train[k_train_index], divided_y_train[k_train_index])

    prediction = logistic_reg_model.predict(x_test_fold)

    scores.append(logistic_reg_model.score(divided_x_train[k_test_index],
divided_y_train[k_test_index]))

    extra_info, p_values = chi2(x_train_fold, y_train_fold)
    conf_matrix = confusion_matrix(y_test_fold, prediction)

    plt.figure()

    sns.set()
    conf_map = sns.heatmap(conf_matrix, cmap="Blues", cbar_kws={'label': 'Number of data points'},
annot=True)
    conf_title = 'Confusion Matrix for multi-class classification L=1'
    conf_map.set_title(conf_title)
    plt.show()

    logistic_roc_area = roc_auc_score(y_test_fold, prediction)
    fp, tp, thresholds = roc_curve(y_test_fold, prediction)
    plt.figure()

    plt_title = 'ROC L:' + str(l) + ' Division:' + str(div) + ' Fold:' + str(k+1)
    plt.title(plt_title)

    plt.plot(fp, tp, label='Logistic Regression (area = %0.2f)' % logistic_roc_area)
    plt.plot([0, 1], [0, 1], 'r--')
    plt.ylabel('Rate of True Positive classifications')
    plt.xlabel('Rate of False Positive classifications')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])

    plt.legend(loc="lower right")
    plt.show()
  division_scores.append(np.mean(scores))
```

Question 2:

ISLR 3.7.4

. I collect a set of data ($n=100n=100$ observations) containing a single predictor and a quantitative response. I then fit a linear regression model to the data, as well as a separate cubic regression, i.e. $Y=\beta0+\beta1X+\beta2X2+\beta3X3+\varepsilon Y=\beta0+\beta1X+\beta2X2+\beta3X3+\varepsilon$.

a. Suppose that the true relationship between XX and YY is linear, i.e. $Y=\beta0+\beta1X+\varepsilon Y=\beta0+\beta1X+\varepsilon$. Consider the training residual sum of squares (RSS) for the linear regression, and also the training RSS for the cubic regression. Would we expect one to be lower than the other, would we expect them to be the same, or is there not enough information to tell? Justify your answer.

It is hard to say which training RSS is lower than the other either cubic or linear without having more detail about the training data. The least squares line may be closer to the true regression line, therefore linear regression RSS may be lower than the cubic regression RSS because true relationship between XX and YY is supposed to be linear.

b. Answer (a) using test rather than training RSS.

We don’t have enough information to say which test RSS will be lower than the other because test RSS depends on test data. But because of the overfitting from training of polynomial regression model, polynomial regression may have a higher test RSS than the linear regression RSS.

c. Suppose that the true relationship between X and Y is not linear, but we don’t know how far it is from linear. Consider the training RSS for the linear regression, and also the training RSS for the cubic regression. Would we expect one to be lower than the other, would we expect them to be the same, or is there not enough information to tell? Justify your answer.

Because of the higher flexibility of polynomial regression, we can say that it will closely follow the training data points and reduce training error regardless of the underlying true relationship and therefore will have lower training RSS.

d. Answer (c) using test rather than training RSS.

As we don’t know how far it is from linear and due to bias-variance tradeoff, we cannot say which RSS is lower than the other. Linear regression test RSS could be lower than the cubic regression test RSS if it is closer to linear than it is to cubic and vice versa.

Question 3:

DATE........../........../........ SUBJECT:

Q.3. As we have to find the posterior probability of an observation belonging to the k^{th} class then X comes from a one-dimensional normal distribution. Therefore, using Bayes' theorem:

$$P(x_k) = \frac{\pi_k f(x_k)}{\sum_{l=1}^{k} \pi_l f(x_l)}$$

As we have a Gaussian or normal distribution, ~~so~~ using the density function:

$$f(x_k) = \frac{1}{\sqrt{2\pi}\,\delta_k} e^{-\frac{(x-\mu_k)^2}{2\delta_k^2}}$$

Combining these two equations:

$$P(x_k) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\,\delta_k} e^{-\frac{(x-\mu_k)^2}{2\delta_k^2}}}{\sum_{l=1}^{k} \pi_l \frac{1}{\sqrt{2\pi}\,\delta_k} e^{-\frac{(x-\mu_l)^2}{2\delta_k^2}}}$$

As the denominator doesn't depend on k^{th} class, we can ignore it. Hence:

DATE........../........../.........   SUBJECT:

$$P(x_k) = \pi_k \frac{1}{\sqrt{2\pi}\,\delta_k} e^{-\frac{(x-\mu_k)^2}{2\delta_k^2}}$$

Now taking log of this equation:

$$\log(P(x_k)) = \log\left(\pi_k \frac{1}{\sqrt{2\pi}\,\delta_k} e^{-\frac{(x-\mu_k)^2}{2\delta_k^2}}\right)$$

$$= \log \pi_k + \log \frac{1}{\sqrt{2\pi}\,\delta_k} - \frac{(x-\mu_k)^2}{2\delta_k^2}$$

$$= \log \pi_k + \log \frac{1}{\sqrt{2\pi}\,\delta_k} - \frac{x^2+\mu_k^2-2x\mu_k}{2\delta_k^2}$$

$$= \log \pi_k + \log \frac{1}{\sqrt{2\pi}\,\delta_k} - \frac{x^2}{2\delta_k^2} + \frac{\mu_k^2}{2\delta_k^2} - \frac{2x\mu_k}{2\delta_k^2}$$

$$= \log \pi_k + \log 1 - \log \sqrt{2\pi}\,\delta_k - \frac{x^2}{2\delta_k^2} + \frac{\mu_k^2}{2\delta_k^2} - \frac{2x\mu_k}{2\delta_k^2}$$

$$= \log \pi_k - \log \delta_k - \frac{x^2}{2\delta_k^2} + \frac{\mu_k^2}{2\delta_k^2} - \frac{2x\mu_k}{2\delta_k^2}$$

We can clearly see in the equation that the equation is quadratic.

Question 4:

DATE........../.........../.........   SUBJECT:

Q.7 Given:

$x = 4$, $\bar{x} = 0$

$\delta^2 = 36$

K (Yes) = 1, $\mu_1 = 10$, $\pi_1 = \frac{80}{100}$

K (No) = 2, $\mu_2 = 0$, $\pi_2 = \frac{20}{100}$

$$P(x_k) = \frac{\pi_k \, f(x_k)}{\sum_{l=1}^{k} \pi_l \, f(x_l)}$$

where,

$$f(x_k) = \frac{1}{\sqrt{2\pi\,\delta^2}} e^{-\frac{(x-\mu_k)^2}{2\delta^2}}$$

So,

$$P(K=1 \mid x=4) = \frac{0.8 \times \frac{1}{\sqrt{2\pi \times 36}} e^{-\frac{(4-10)^2}{2\times 36}}}{0.8 \times \frac{1}{\sqrt{2\pi\times 36}} e^{-\frac{(4-10)^2}{2\times 36}} + 0.2 \times \frac{1}{\sqrt{2\pi \times 36}} e^{-\frac{(4-0)^2}{}}}$$

$$= \frac{\cancel{\frac{1}{\sqrt{2\pi\times 36}}}(0.8) \times 0.6065}{\cancel{\frac{1}{\sqrt{2\pi\times 36}}}\left((0.8 \times 0.6065) + 0.2 \times 0.801\right)}$$

$$= 0.752$$