In [2]:
```
import pandas as pd
import numpy as np
import os
import seaborn as sns
```

In [3]:
```
path='AReM/'
rootFolder=os.listdir(path)#use OS to get the path of files
trainSetPath=[]#set Train Set path
testSetPath=[]#set Test Set path
totalSetPath=[]#set of all data set for question c
bendingtrainSetPath=[]#set  bending Train Set path
notbendingtrainSetPath=[]#set not bending train Set for question d
```

In [4]:
```
#************************************************ question b **********************
for folderName in rootFolder[0:]:#delete the 2 pdf function,go though all the folder
    temp = os.path.join('%s%s' % (path, folderName))
    featurePath= os.listdir(temp+ '/')#get all the file in each fold
    featureL=len(featurePath)#get the length of folder
    if (folderName =='bending1' or folderName =='bending2'):
        for i in range(0,2):
            testSetPath.append('AReM/'+ folderName + '/dataset' + str(i+1) + '.csv')
            totalSetPath.append('AReM/'+ folderName + '/dataset' + str(i+1) + '.csv')
        for i in range(2,featureL):
            trainSetPath.append('AReM/'+ folderName + '/dataset' + str(i+1) + '.csv')
            totalSetPath.append('AReM/'+ folderName + '/dataset' + str(i+1) + '.csv')
            bendingtrainSetPath.append('AReM/'+ folderName + '/dataset' + str(i+1) + '
    elif (folderName =='cycling' or folderName =='lying' or folderName =='sitting' or
        for i in range(0,3):
            testSetPath.append('AReM/'+ folderName+'/dataset'+str(i+1)+'.csv')
            totalSetPath.append('AReM/'+ folderName + '/dataset' + str(i+1) + '.csv')
        for i in range(3,featureL):
            trainSetPath.append('AReM/'+ folderName+'/dataset'+str(i+1)+'.csv')
            totalSetPath.append('AReM/'+ folderName + '/dataset' + str(i+1) + '.csv')
            notbendingtrainSetPath.append('AReM/'+ folderName+'/dataset'+str(i+1)+'.csv
#************************************************end of question b***************
```

2 question c

2.1 c_i

mean, minimum, maximum, median, standard deviation, varience are usually used to research time series

3 c_ii

```
In [5]: #*************************** question c ******************************
        #
        instanceCol,instanceRow=43,88;
        Instance = [[0 for x in range(instanceCol)] for y in range(instanceRow)]
        for i in range(0,instanceRow):
            tempDfp = pd.read_csv(totalSetPath[i],skiprows=5,header=None)
            tempDf=tempDfp.values
            timeSeries1=tempDf[:,1]#spilt the 6 series
            timeSeries2=tempDf[:,2]
            timeSeries3=tempDf[:,3]
            timeSeries4=tempDf[:,4]
            timeSeries5=tempDf[:,5]
            timeSeries6=tempDf[:,6]

            maxtS1=np.max(timeSeries1)#get the max
            maxtS2=np.max(timeSeries2)
            maxtS3=np.max(timeSeries3)
            maxtS4=np.max(timeSeries4)
            maxtS5=np.max(timeSeries5)
            maxtS6=np.max(timeSeries6)

            mintS1=np.min(timeSeries1)#get the min
            mintS2=np.min(timeSeries2)
            mintS3=np.min(timeSeries3)
            mintS4=np.min(timeSeries4)
            mintS5=np.min(timeSeries5)
            mintS6=np.min(timeSeries6)

            meantS1=np.mean(timeSeries1)#get the mean
            meantS2=np.mean(timeSeries2)
            meantS3=np.mean(timeSeries3)
            meantS4=np.mean(timeSeries4)
            meantS5=np.mean(timeSeries5)
            meantS6=np.mean(timeSeries6)
```

```
mediantS1=np.median(timeSeries1)#get the median
mediantS2=np.median(timeSeries2)
mediantS3=np.median(timeSeries3)
mediantS4=np.median(timeSeries4)
mediantS5=np.median(timeSeries5)
mediantS6=np.median(timeSeries6)

stdtS1=np.std(timeSeries1)#get the std
stdtS2=np.std(timeSeries2)
stdtS3=np.std(timeSeries3)
stdtS4=np.std(timeSeries4)
stdtS5=np.std(timeSeries5)
stdtS6=np.std(timeSeries6)

quart_25_S1=np.percentile(timeSeries1,25)#get the first quart
quart_25_S2=np.percentile(timeSeries2,25)
quart_25_S3=np.percentile(timeSeries3,25)
quart_25_S4=np.percentile(timeSeries4,25)
quart_25_S5=np.percentile(timeSeries5,25)
quart_25_S6=np.percentile(timeSeries6,25)

quart_75_S1=np.percentile(timeSeries1,75)#get the third quart
quart_75_S2=np.percentile(timeSeries2,75)
quart_75_S3=np.percentile(timeSeries3,75)
quart_75_S4=np.percentile(timeSeries4,75)
quart_75_S5=np.percentile(timeSeries5,75)
quart_75_S6=np.percentile(timeSeries6,75)

Instance[i][0]=maxtS1 #save different data into the matrix Instance
Instance[i][1]=mintS1
Instance[i][2]=meantS1
Instance[i][3]=mediantS1
Instance[i][4]=stdtS1
Instance[i][5]=quart_25_S1
Instance[i][6]=quart_75_S1

Instance[i][7]=maxtS2
Instance[i][8]=mintS2
Instance[i][9]=meantS2
Instance[i][10]=mediantS2
Instance[i][11]=stdtS2
Instance[i][12]=quart_25_S2
Instance[i][13]=quart_75_S2

Instance[i][14]=maxtS3
Instance[i][15]=mintS3
Instance[i][16]=meantS3
Instance[i][17]=mediantS3
```

```
            Instance[i][18]=stdtS3
            Instance[i][19]=quart_25_S3
            Instance[i][20]=quart_75_S3

            Instance[i][21]=maxtS4
            Instance[i][22]=mintS4
            Instance[i][23]=meantS4
            Instance[i][24]=mediantS4
            Instance[i][25]=stdtS4
            Instance[i][26]=quart_25_S4
            Instance[i][27]=quart_75_S4

            Instance[i][28]=maxtS5
            Instance[i][29]=mintS5
            Instance[i][30]=meantS5
            Instance[i][31]=mediantS5
            Instance[i][32]=stdtS5
            Instance[i][33]=quart_25_S5
            Instance[i][34]=quart_75_S5

            Instance[i][35]=maxtS6
            Instance[i][36]=mintS6
            Instance[i][37]=meantS6
            Instance[i][38]=mediantS6
            Instance[i][39]=stdtS6
            Instance[i][40]=quart_25_S6
            Instance[i][41]=quart_75_S6
            if(0<i+1<14):
                Instance[i][42]=1 #'bending'
            if(13<i+1<29):
                Instance[i][42]=2 #'cycling'
            if(28<i+1<44):
                Instance[i][42]=3 #'lying'
            if(43<i+1<59):
                Instance[i][42]=4 #'sitting'
            if(58<i+1<74):
                Instance[i][42]=5 #'standing'
            if(73<i+1<89):
                Instance[i][42]=6 #'walking'
        instanceDf = pd.DataFrame(data=Instance,index=None,columns=None)
        #I choose min max and mean
        #**************************************************end of question c**************
```

```
In [6]: instanceDf.columns=['maxtS1', 'mintS1','meantS1','mediantS1','stdtS1','quart_25_S1','qu
```

```
In [7]: instanceDf
```

```
Out[7]:     maxtS1  mintS1    meantS1  mediantS1    stdtS1  quart_25_S1  quart_75_S1  \
        0    45.00   37.25  40.624792     40.500  1.475428      39.2500      42.0000
```

1	45.67	38.00	42.812812	42.500	1.434054	42.0000	43.6700
2	47.40	35.00	43.954500	44.330	1.557210	43.0000	45.0000
3	47.75	33.00	42.179812	43.500	3.666840	39.1500	45.0000
4	45.75	33.00	41.678063	41.750	2.241152	41.3300	42.7500
5	48.00	37.00	43.454958	43.250	1.384653	42.5000	45.0000
6	48.00	36.25	43.969125	44.500	1.616677	43.3100	44.6700
7	51.00	12.75	24.562958	24.250	3.733619	23.1875	26.5000
8	42.75	0.00	27.464604	28.000	3.579847	25.5000	30.0000
9	50.00	21.00	32.586208	33.000	6.231642	26.1875	34.5000
10	33.00	27.50	29.881938	30.000	1.152635	29.0000	30.2700
11	45.50	19.00	30.938104	29.000	7.676137	26.7500	38.0000
12	47.50	25.00	31.058250	29.710	4.824761	27.5000	31.8125
13	45.00	24.25	37.177042	36.250	3.577569	34.5000	40.2500
14	44.75	28.75	37.561187	36.875	3.223144	35.2500	40.2500
15	44.67	22.00	37.058708	36.000	3.706313	34.5000	40.0625
16	44.00	19.00	36.228396	36.000	3.524939	34.0000	39.0000
17	44.33	26.50	36.687292	36.000	3.525726	34.2500	39.3725
18	45.00	25.33	37.114313	36.250	3.706518	34.5000	40.2500
19	44.75	26.75	36.863375	36.330	3.552081	34.5000	39.7500
20	44.25	26.25	36.957458	36.290	3.431283	34.5000	40.2500
21	44.67	27.75	37.144833	36.330	3.754986	34.0000	40.5000
22	45.00	27.00	36.819521	36.000	3.896394	33.7500	40.2500
23	44.33	27.00	36.541667	36.000	4.014734	33.2500	39.8125
24	44.25	18.50	35.752354	36.000	4.609992	33.0000	39.3300
25	43.75	19.00	35.879875	36.000	4.610068	33.0000	39.5000
26	43.50	23.33	36.244083	36.750	3.818032	33.4575	39.2500
27	45.00	24.25	37.177042	36.250	3.577569	34.5000	40.2500
28	30.00	23.50	27.716375	27.500	1.440750	27.0000	29.0000
29	48.33	24.75	44.182937	48.000	7.487803	48.0000	48.0000
..	...	...	...	...	...	...	...
58	48.00	33.33	44.334729	45.000	2.474358	42.2500	46.5000
59	46.25	35.50	43.174938	43.670	1.986979	42.5000	44.5000
60	47.00	32.75	42.760562	44.500	3.395376	41.3300	45.3725
61	46.67	30.00	42.648521	42.750	2.392842	41.5000	45.0000
62	47.50	36.00	43.720021	45.000	2.381620	43.0000	45.0000
63	47.75	34.50	44.471146	45.000	1.770706	45.0000	45.2500
64	48.00	35.50	46.224938	46.000	1.746493	45.2500	48.0000
65	48.00	29.75	46.932208	47.500	1.830755	47.2375	47.7500
66	47.67	36.33	45.399625	45.500	1.326737	45.0000	46.3300
67	45.80	36.00	42.419917	42.670	2.517503	41.3300	44.6175
68	48.25	37.00	42.516958	42.500	2.193462	41.0000	44.5000
69	45.50	36.25	42.959354	42.670	1.499314	42.0000	44.3300
70	47.33	36.00	42.674583	43.670	2.381685	40.0000	44.7500
71	45.75	36.25	43.187521	44.750	2.488565	39.7500	45.0000
72	47.33	36.00	44.441187	45.000	2.415277	44.6275	45.7500
73	43.50	19.33	34.227771	35.500	4.884480	30.5000	37.7500
74	45.00	12.50	33.509729	34.125	4.845868	30.5000	36.7500
75	46.75	15.00	34.660583	35.000	5.309571	31.0000	38.2500

76	46.00	18.00	35.193333	36.000	4.746916	32.0000	38.7500
77	46.25	20.75	34.763333	35.290	4.737266	31.6700	38.2500
78	51.00	21.50	34.935812	35.500	4.641102	32.0000	38.0625
79	47.67	18.33	34.333042	34.750	4.943612	31.2500	38.0000
80	45.75	18.33	34.599875	35.125	4.726858	31.5000	38.0000
81	43.67	15.50	34.225875	34.750	4.437168	31.2500	37.2500
82	51.25	21.50	34.253521	35.000	4.935592	30.9375	37.7500
83	45.33	19.50	33.586875	34.250	4.646088	30.2500	37.0000
84	45.50	19.75	34.322750	35.250	4.747524	31.0000	38.0000
85	46.00	19.50	34.546229	35.250	4.837247	31.2500	37.8125
86	46.25	23.50	34.873229	35.250	4.526997	31.7500	38.2500
87	44.00	19.25	34.473188	35.000	4.791706	31.2500	38.0000

	maxtS2	mintS2	meantS2	...	quart_25_S5	quart_75_S5	maxtS6	mintS6	\
0	1.30	0.0	0.358604	...	33.0000	36.0000	1.92	0.00	
1	1.22	0.0	0.372437	...	32.0000	34.5000	3.11	0.00	
2	1.70	0.0	0.426250	...	35.3625	36.5000	1.79	0.00	
3	3.00	0.0	0.696042	...	30.4575	36.3300	2.18	0.00	
4	2.83	0.0	0.535979	...	28.4575	31.2500	1.79	0.00	
5	1.58	0.0	0.378083	...	22.2500	24.0000	5.26	0.00	
6	1.50	0.0	0.413125	...	20.5000	23.7500	2.96	0.00	
7	6.87	0.0	0.590833	...	20.5000	27.0000	4.97	0.00	
8	7.76	0.0	0.449708	...	15.0000	20.7500	6.76	0.00	
9	9.90	0.0	0.516125	...	17.6700	23.5000	13.61	0.00	
10	1.00	0.0	0.256437	...	17.0000	19.0000	6.40	0.00	
11	6.40	0.0	0.467167	...	15.0000	20.8125	6.73	0.00	
12	6.38	0.0	0.405458	...	9.0000	18.3125	4.92	0.00	
13	8.58	0.0	2.374208	...	17.9500	21.7500	9.34	0.00	
14	9.91	0.0	2.080687	...	18.0000	21.5000	9.62	0.00	
15	14.17	0.0	2.438146	...	16.0000	21.0000	8.55	0.00	
16	12.28	0.0	2.831687	...	14.0000	18.0625	9.98	0.00	
17	12.89	0.0	2.973042	...	14.6700	18.5000	8.19	0.00	
18	10.84	0.0	2.730000	...	14.7500	18.5000	9.50	0.00	
19	11.68	0.0	2.757312	...	15.0000	18.6700	8.81	0.00	
20	8.64	0.0	2.420083	...	14.0000	18.2500	8.34	0.00	
21	10.76	0.0	2.419062	...	15.0000	18.7500	8.75	0.00	
22	10.47	0.0	2.600146	...	15.5000	19.2700	8.99	0.00	
23	10.43	0.0	2.847958	...	15.0000	19.5000	9.18	0.00	
24	12.60	0.0	3.328104	...	14.0000	18.0625	9.39	0.00	
25	11.20	0.0	3.414312	...	14.7500	19.6900	8.50	0.00	
26	9.71	0.0	2.736021	...	15.7500	21.0000	11.15	0.00	
27	8.58	0.0	2.374208	...	17.9500	21.7500	9.34	0.00	
28	1.79	0.0	0.363687	...	5.5000	10.7500	4.50	0.00	
29	3.11	0.0	0.101875	...	2.0000	5.5425	3.91	0.00	
..	...	...	...	...	...	...	...	...	
58	3.90	0.0	0.432958	...	9.3300	17.7500	5.02	0.00	
59	2.12	0.0	0.506583	...	12.7500	16.5000	5.72	0.00	
60	3.34	0.0	0.486167	...	13.0000	18.5650	5.73	0.00	

61	2.95	0.0	0.402833	...	10.6275	14.2500	4.64	0.00
62	1.92	0.0	0.366708	...	11.3100	15.5425	6.18	0.00
63	2.18	0.0	0.290479	...	12.0000	14.8125	4.32	0.00
64	4.50	0.0	0.312354	...	12.0000	15.2500	6.00	0.00
65	4.60	0.0	0.429667	...	11.6700	15.5000	6.58	0.00
66	1.66	0.0	0.460146	...	11.2500	14.5000	4.50	0.00
67	2.12	0.0	0.460562	...	7.6275	12.0000	6.65	0.00
68	2.12	0.0	0.440687	...	12.6275	17.5000	6.85	0.00
69	2.60	0.0	0.352875	...	14.0000	16.6900	4.00	0.00
70	2.17	0.0	0.419167	...	12.7500	16.5000	3.77	0.00
71	2.83	0.0	0.271271	...	16.5000	21.0000	3.83	0.00
72	4.50	0.0	0.346604	...	11.0000	14.6700	5.91	0.00
73	14.50	0.0	3.995729	...	14.7500	18.6700	9.74	0.00
74	13.05	0.0	4.450771	...	14.6275	18.7500	8.96	0.00
75	13.44	0.0	4.200896	...	14.2500	18.5000	8.99	0.00
76	16.20	0.0	4.321021	...	14.2500	18.5000	8.50	0.00
77	12.68	0.0	4.223792	...	14.2500	18.3300	9.39	0.00
78	12.21	0.0	4.115750	...	14.2375	18.2500	10.21	0.00
79	12.48	0.0	4.396958	...	13.7500	18.0000	8.01	0.00
80	15.37	0.0	4.398833	...	14.0000	18.2500	8.86	0.00
81	17.24	0.0	4.354500	...	14.3300	18.2500	9.42	0.00
82	13.55	0.0	4.457896	...	13.7500	18.0000	8.32	0.00
83	14.67	0.0	4.576562	...	13.7300	18.2500	8.32	0.00
84	13.47	0.0	4.456333	...	13.5000	17.7500	9.67	0.00
85	12.47	0.0	4.371958	...	14.0000	17.7500	10.00	0.00
86	14.82	0.0	4.380583	...	13.7500	18.0000	9.51	0.00
87	13.86	0.0	4.359312	...	13.7300	17.7500	9.00	0.43

	meantS6	mediantS6	stdtS6	quart_25_S6	quart_75_S6	class
0	0.570583	0.430	0.582308	0.0000	1.3000	1
1	0.571083	0.430	0.600383	0.0000	1.3000	1
2	0.493292	0.430	0.512971	0.0000	0.9400	1
3	0.613521	0.500	0.523771	0.0000	1.0000	1
4	0.383292	0.430	0.388759	0.0000	0.5000	1
5	0.679646	0.500	0.621885	0.4300	0.8700	1
6	0.555312	0.490	0.487318	0.0000	0.8300	1
7	0.700188	0.500	0.692997	0.4300	0.8700	1
8	1.122125	0.830	1.011287	0.4700	1.3000	1
9	1.162042	0.830	1.331591	0.4700	1.3000	1
10	0.701625	0.710	0.480601	0.4700	0.9400	1
11	1.107354	0.830	1.079715	0.4700	1.3000	1
12	1.098104	0.940	0.830614	0.5000	1.3000	1
13	2.921729	2.500	1.850669	1.5000	3.9000	2
14	2.765896	2.450	1.767359	1.4100	3.7700	2
15	2.983750	2.570	1.813837	1.5000	4.1500	2
16	3.480688	3.340	1.825864	2.1025	4.5500	2
17	3.073313	2.690	1.627976	1.9125	4.0875	2
18	3.076354	2.770	1.822633	1.7000	4.0375	2

19	2.773313	2.590	1.568283	1.6400	3.6325	2
20	2.934625	2.525	1.629680	1.6600	4.0300	2
21	2.822437	2.590	1.635476	1.5800	3.7400	2
22	2.887563	2.525	1.721298	1.5600	3.7700	2
23	3.225458	2.870	1.767913	1.8850	4.2625	2
24	3.069667	2.770	1.746503	1.7975	4.0600	2
25	3.093021	2.930	1.624339	1.8900	4.0600	2
26	3.530500	3.110	1.961639	2.1700	4.6175	2
27	2.921729	2.500	1.850669	1.5000	3.9000	2
28	0.734271	0.710	0.613049	0.4300	1.0000	3
29	0.692771	0.500	0.675076	0.3225	0.9400	3
..	...	...	...	...	...	...
58	0.933000	0.830	0.672907	0.4700	1.2500	5
59	0.911979	0.830	0.665467	0.4700	1.2200	5
60	0.842271	0.710	0.721413	0.4300	1.0900	5
61	0.917354	0.830	0.708898	0.4700	1.1200	5
62	1.039687	0.830	0.915702	0.4700	1.2200	5
63	0.927375	0.830	0.755647	0.4700	1.2200	5
64	0.882583	0.830	0.667727	0.4700	1.1200	5
65	0.991125	0.830	0.854438	0.4700	1.2200	5
66	0.795104	0.820	0.502483	0.4700	1.0000	5
67	1.226271	1.090	0.891058	0.5000	1.5850	5
68	0.977417	0.830	0.852390	0.4700	1.2200	5
69	0.748479	0.820	0.460671	0.4300	0.9500	5
70	0.702042	0.500	0.566859	0.4300	0.9400	5
71	0.645458	0.500	0.566828	0.4300	0.8300	5
72	1.155083	0.940	0.841210	0.5000	1.5000	5
73	3.394125	3.100	1.790222	2.1050	4.4250	6
74	3.378479	3.085	1.785497	2.0600	4.4400	6
75	3.244396	3.000	1.629283	2.1200	4.2400	6
76	3.241958	3.015	1.767339	1.8850	4.4400	6
77	3.288271	3.270	1.645811	2.0500	4.3050	6
78	3.280021	3.015	1.699145	2.1200	4.5000	6
79	3.261583	2.980	1.615604	2.0500	4.3200	6
80	3.289542	3.015	1.678418	2.1200	4.2600	6
81	3.479542	3.270	1.759311	2.2400	4.5375	6
82	3.500750	3.285	1.690614	2.1800	4.5575	6
83	3.259729	3.110	1.638534	2.0500	4.3225	6
84	3.432563	3.200	1.730921	2.1575	4.5650	6
85	3.338125	3.080	1.655016	2.1600	4.3350	6
86	3.424646	3.270	1.689198	2.1700	4.5000	6
87	3.340458	3.090	1.697343	2.1200	4.3750	6

[88 rows x 43 columns]

4 end of c_ii

5 c_iii

I choose maximum,first quartile ,third quartile for the result I got in d, we can see all of these 3 feature cannot be eliminated

6 question d

6.1 d_i

In [8]:
```
#****************************** question d ******************************
#************************d_i**********************************
BendingInstance = [[0 for x in range(22)] for y in range(64)]#9
for i in range(0,9):
    tempDfp = pd.read_csv(bendingtrainSetPath[i],skiprows=5,header=None)
    tempDf=tempDfp.values
    timeSeries1=tempDf[:,1]#spilt the 6 series
    timeSeries2=tempDf[:,2]
    timeSeries6=tempDf[:,6]

    maxtS1=np.max(timeSeries1)#get the max
    maxtS2=np.max(timeSeries2)
    maxtS6=np.max(timeSeries6)

    mintS1=np.min(timeSeries1)#get the min
    mintS2=np.min(timeSeries2)
    mintS6=np.min(timeSeries6)

    meantS1=np.mean(timeSeries1)#get the mean
    meantS2=np.mean(timeSeries2)
    meantS6=np.mean(timeSeries6)

    mediantS1=np.median(timeSeries1)#get the median
    mediantS2=np.median(timeSeries2)
    mediantS6=np.median(timeSeries6)

    stdtS1=np.std(timeSeries1)#get the std
    stdtS2=np.std(timeSeries2)
    stdtS6=np.std(timeSeries6)

    quart_25_S1=np.percentile(timeSeries1,25)#get the first quart
    quart_25_S2=np.percentile(timeSeries2,25)
    quart_25_S6=np.percentile(timeSeries6,25)

    quart_75_S1=np.percentile(timeSeries1,75)#get the third quart
    quart_75_S2=np.percentile(timeSeries2,75)
```

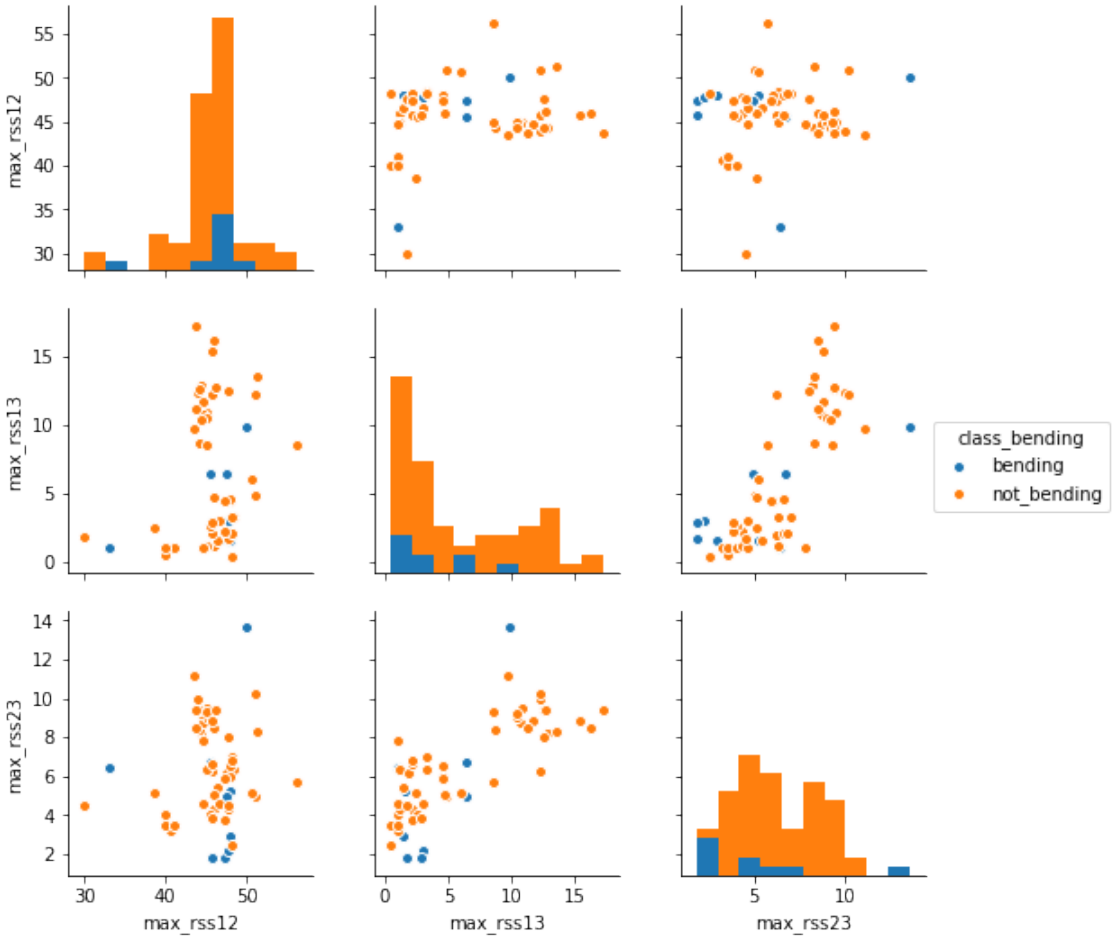
```
    quart_75_S6=np.percentile(timeSeries6,75)

    BendingInstance[i][0]=maxtS1 #save different data into the matrix Instance
    BendingInstance[i][1]=mintS1
    BendingInstance[i][2]=meantS1
    BendingInstance[i][3]=mediantS1
    BendingInstance[i][4]=stdtS1
    BendingInstance[i][5]=quart_25_S1
    BendingInstance[i][6]=quart_75_S1

    BendingInstance[i][7]=maxtS2
    BendingInstance[i][8]=mintS2
    BendingInstance[i][9]=meantS2
    BendingInstance[i][10]=mediantS2
    BendingInstance[i][11]=stdtS2
    BendingInstance[i][12]=quart_25_S2
    BendingInstance[i][13]=quart_75_S2

    BendingInstance[i][14]=maxtS6
    BendingInstance[i][15]=mintS6
    BendingInstance[i][16]=meantS6
    BendingInstance[i][17]=mediantS6
    BendingInstance[i][18]=stdtS6
    BendingInstance[i][19]=quart_25_S6
    BendingInstance[i][20]=quart_75_S6
    BendingInstance[i][21]='bending'

for i in range(9,64):
    tempDfp = pd.read_csv(notbendingtrainSetPath[i-9],skiprows=5,header=None)
    tempDf=tempDfp.values
    timeSeries1=tempDf[:,1]#spilt the 6 series
    timeSeries2=tempDf[:,2]
    timeSeries6=tempDf[:,6]

    maxtS1=np.max(timeSeries1)#get the max
    maxtS2=np.max(timeSeries2)
    maxtS6=np.max(timeSeries6)

    mintS1=np.min(timeSeries1)#get the min
    mintS2=np.min(timeSeries2)
    mintS6=np.min(timeSeries6)

    meantS1=np.mean(timeSeries1)#get the mean
    meantS2=np.mean(timeSeries2)
    meantS6=np.mean(timeSeries6)

    mediantS1=np.median(timeSeries1)#get the median
    mediantS2=np.median(timeSeries2)
```

```
            mediantS6=np.median(timeSeries6)

            stdtS1=np.std(timeSeries1)#get the std
            stdtS2=np.std(timeSeries2)
            stdtS6=np.std(timeSeries6)

            quart_25_S1=np.percentile(timeSeries1,25)#get the first quart
            quart_25_S2=np.percentile(timeSeries2,25)
            quart_25_S6=np.percentile(timeSeries6,25)

            quart_75_S1=np.percentile(timeSeries1,75)#get the third quart
            quart_75_S2=np.percentile(timeSeries2,75)
            quart_75_S6=np.percentile(timeSeries6,75)

            BendingInstance[i][0]=maxtS1 #save different data into the matrix Instance
            BendingInstance[i][1]=mintS1
            BendingInstance[i][2]=meantS1
            BendingInstance[i][3]=mediantS1
            BendingInstance[i][4]=stdtS1
            BendingInstance[i][5]=quart_25_S1
            BendingInstance[i][6]=quart_75_S1

            BendingInstance[i][7]=maxtS2
            BendingInstance[i][8]=mintS2
            BendingInstance[i][9]=meantS2
            BendingInstance[i][10]=mediantS2
            BendingInstance[i][11]=stdtS2
            BendingInstance[i][12]=quart_25_S2
            BendingInstance[i][13]=quart_75_S2

            BendingInstance[i][14]=maxtS6
            BendingInstance[i][15]=mintS6
            BendingInstance[i][16]=meantS6
            BendingInstance[i][17]=mediantS6
            BendingInstance[i][18]=stdtS6
            BendingInstance[i][19]=quart_25_S6
            BendingInstance[i][20]=quart_75_S6
            BendingInstance[i][21]='not_bending'

        BendingInstanceDf = pd.DataFrame(data=BendingInstance,index=None,columns=None)
        BendingInstanceDf.columns=['max_rss12', 'min_rss12','mean_rss12','median_rss12','std_rs

        #get the dataframe of bending instance

In [78]: #BendingInstanceDf

In [10]: max_Bending=BendingInstanceDf[['max_rss12','max_rss13','max_rss23','class_bending']]
         mean_Bending=BendingInstanceDf[['mean_rss12','mean_rss13','mean_rss23','class_bending
```

```
std_Bending=BendingInstanceDf[['std_rss12','std_rss13','std_rss23','class_bending']]
median_Bending=BendingInstanceDf[['median_rss12','median_rss13','median_rss23','class_
quart25_Bending=BendingInstanceDf[['quart25_rss12','quart25_rss13','quart25_rss23','cl
quart75_Bending=BendingInstanceDf[['quart75_rss12','quart75_rss13','quart75_rss23','cl
```

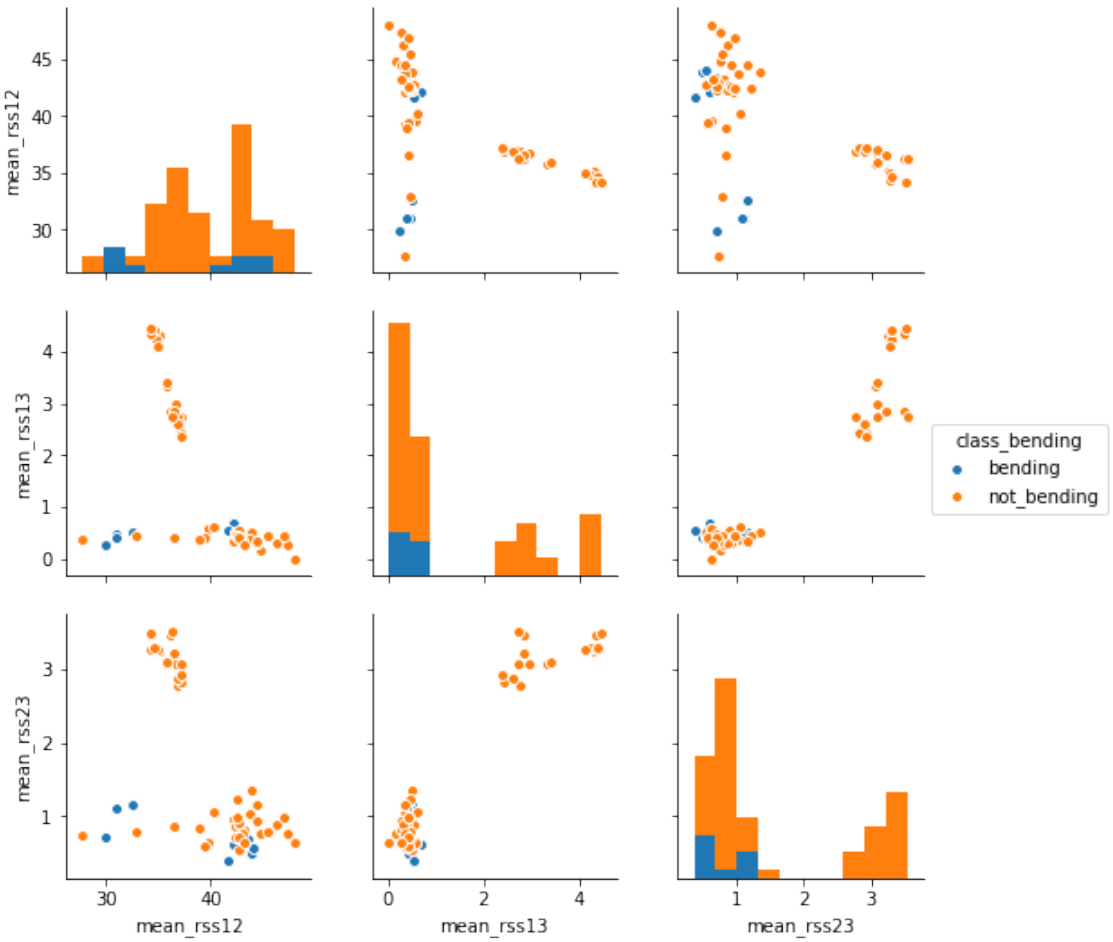
In [11]:
```
sns.pairplot(max_Bending,hue="class_bending")
```

Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x1b4825ec780>
```



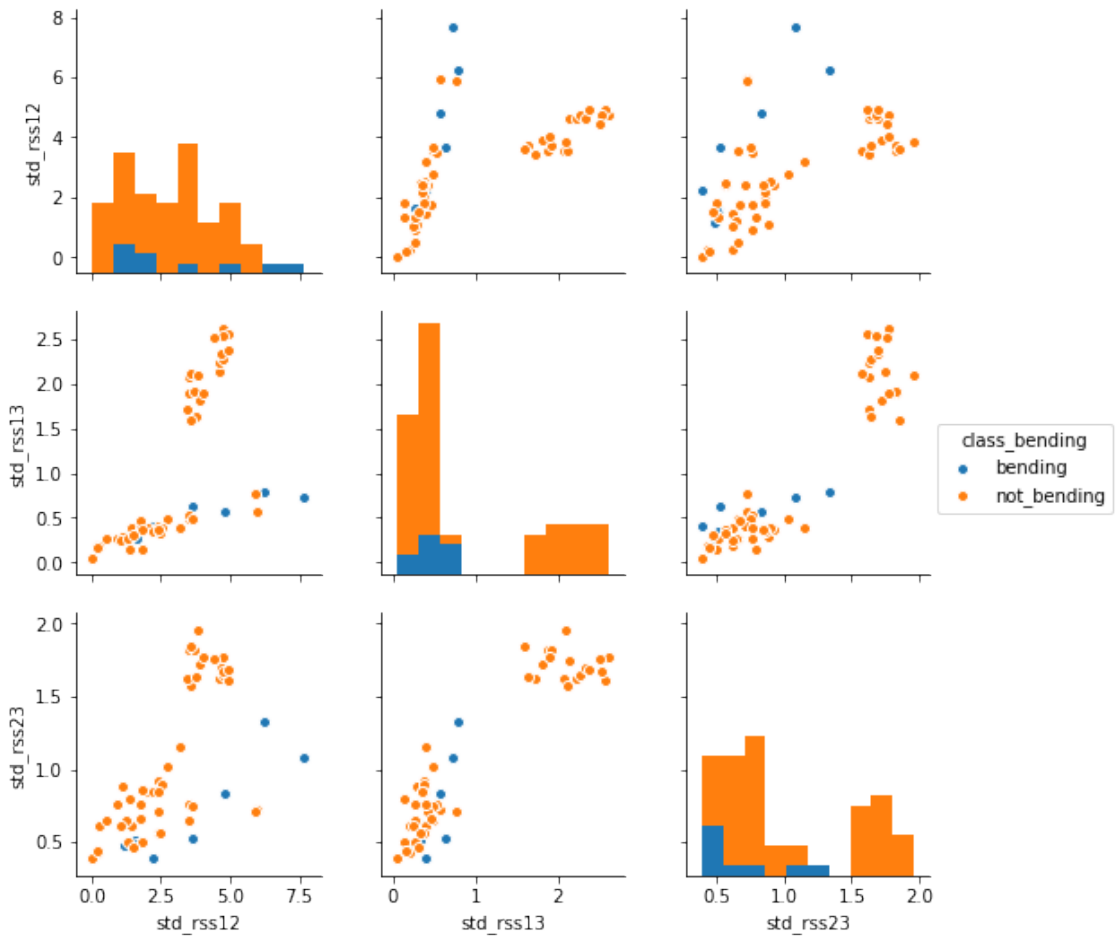
In [11]:
```
sns.pairplot(mean_Bending,hue="class_bending")
```

Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x25a10b97860>
```



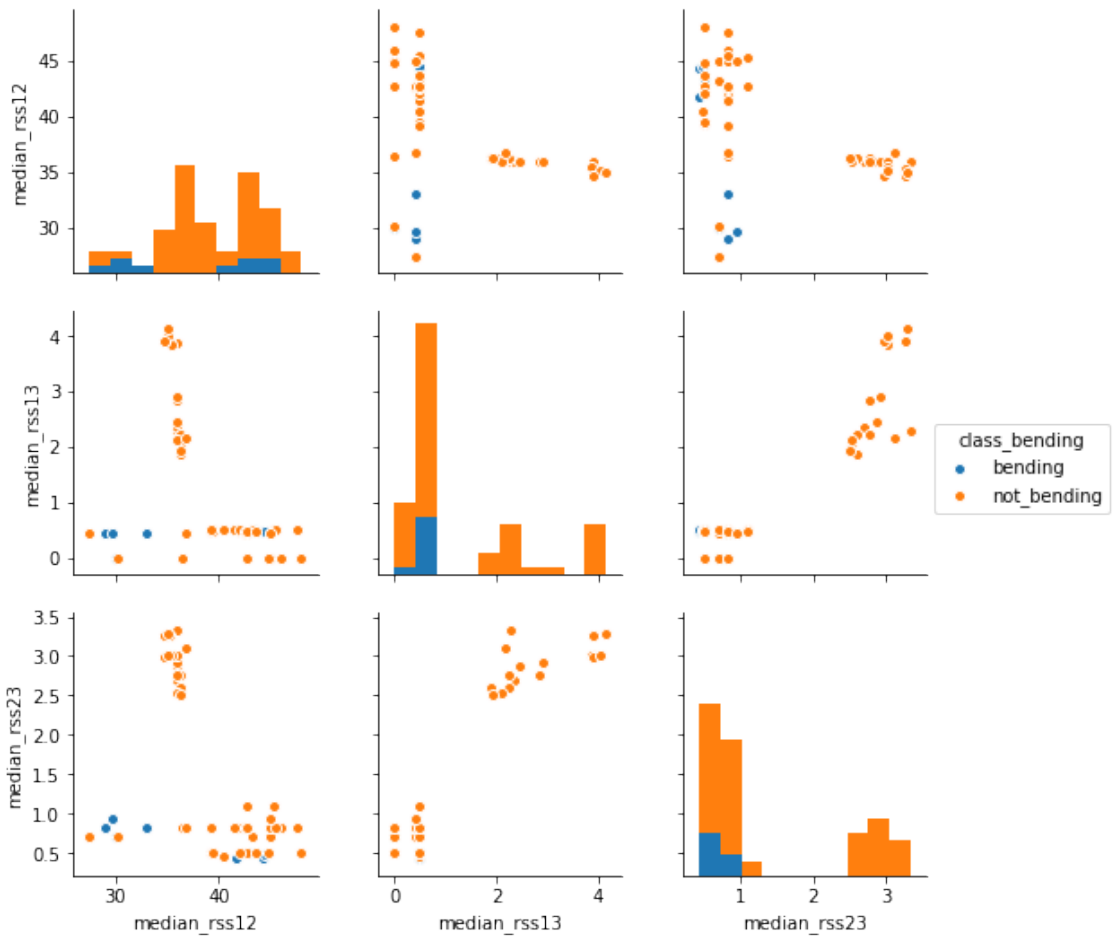

```
In [12]: sns.pairplot(std_Bending,hue="class_bending")
```

```
Out[12]: <seaborn.axisgrid.PairGrid at 0x25a112142b0>
```



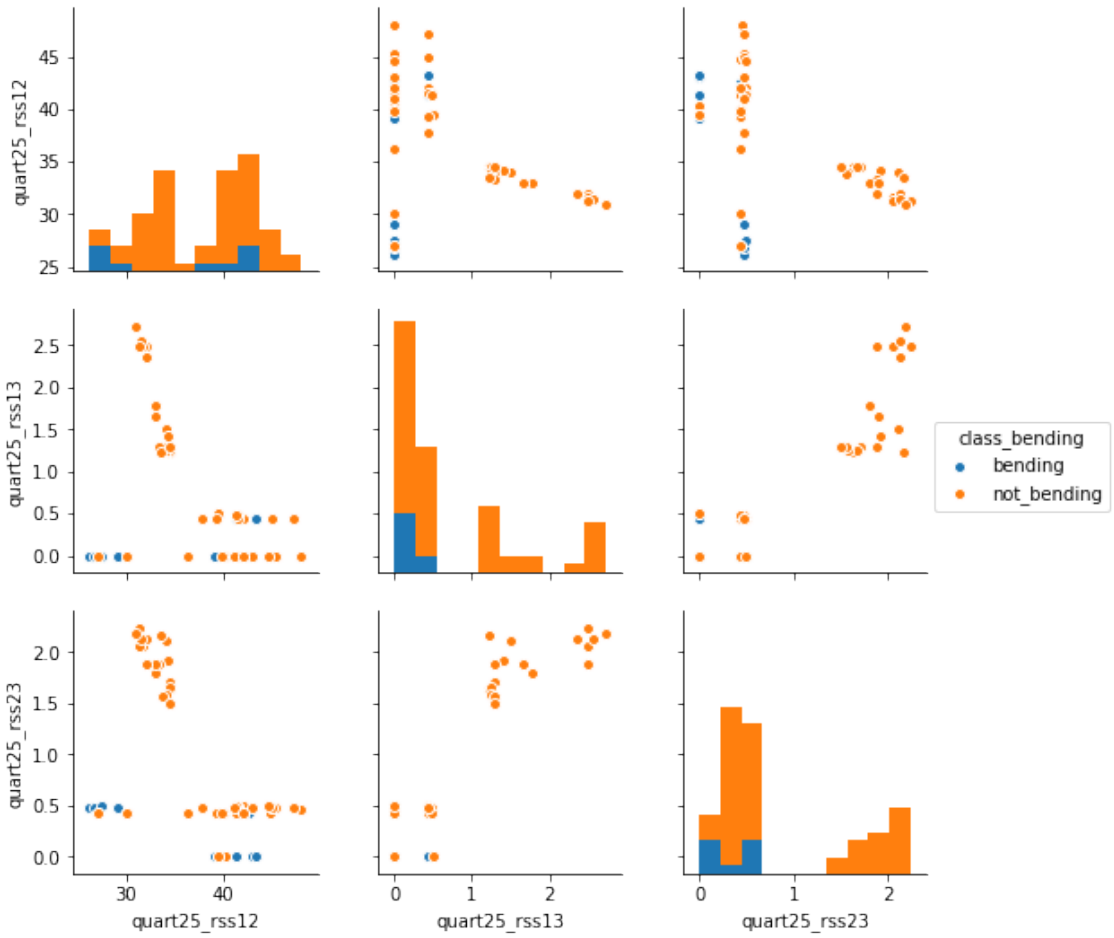

```
In [13]: sns.pairplot(median_Bending,hue="class_bending")
```

```
Out[13]: <seaborn.axisgrid.PairGrid at 0x25a119dfc18>
```



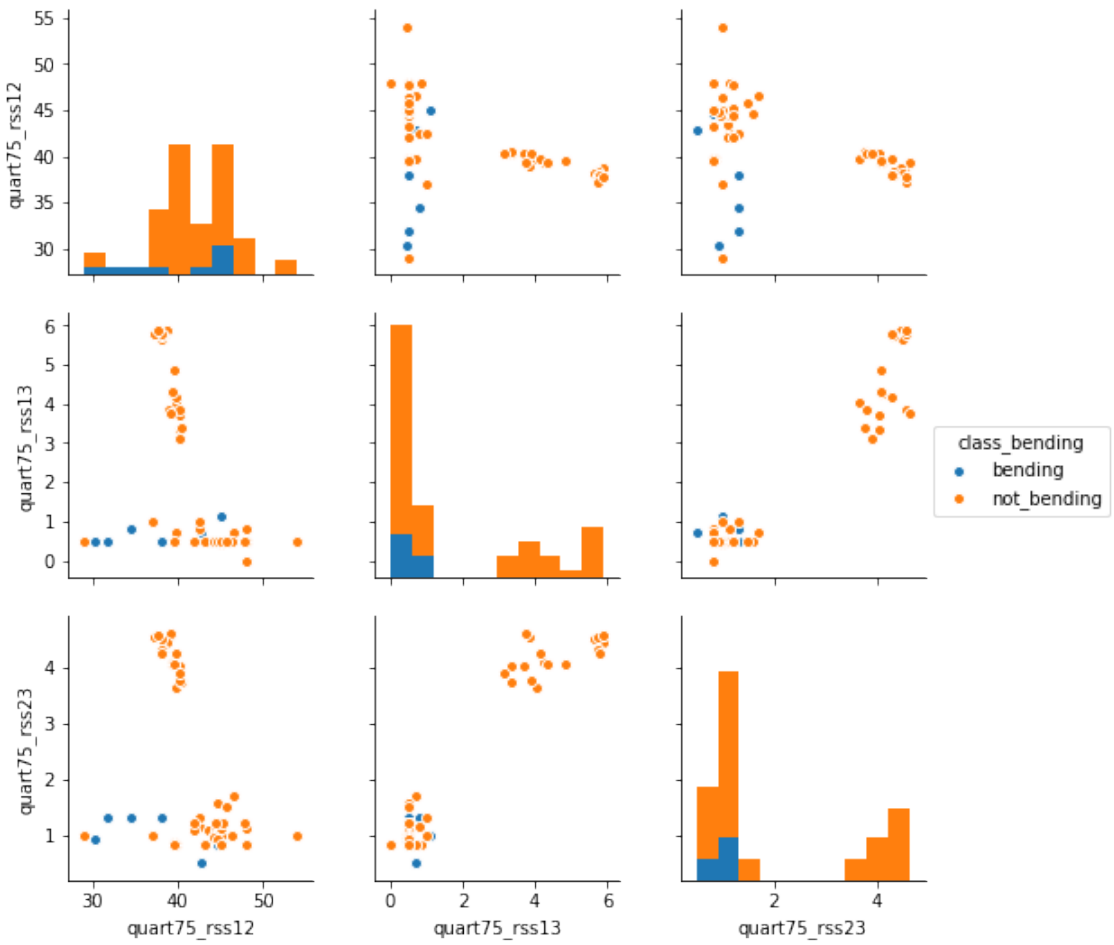

```
In [14]: sns.pairplot(quart25_Bending,hue="class_bending")
```

```
Out[14]: <seaborn.axisgrid.PairGrid at 0x25a11ff8978>
```




```
In [15]: sns.pairplot(quart75_Bending,hue="class_bending")
         #**********end of question d_i*********
```

```
Out[15]: <seaborn.axisgrid.PairGrid at 0x25a129cc080>
```



6.2 end of d_i

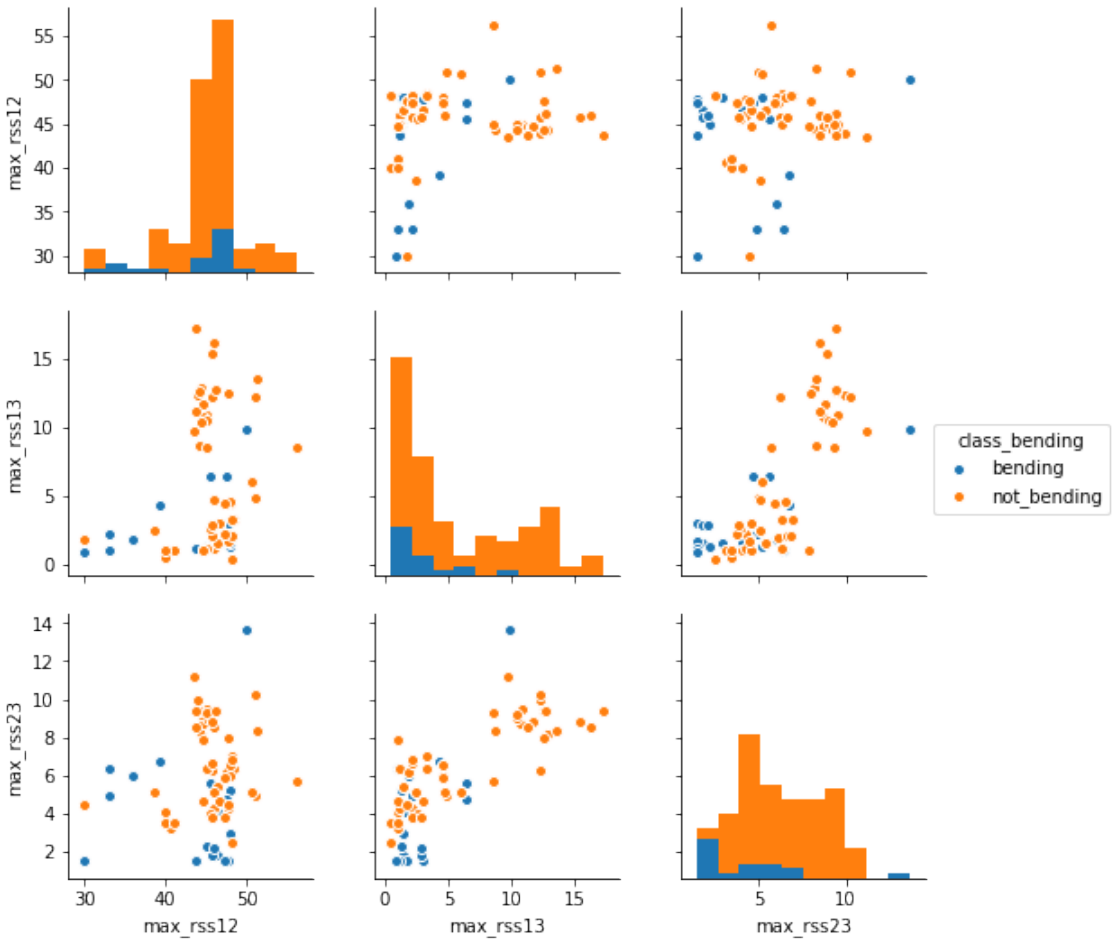
7 d_ii

```
In [12]: ######question d_ii######
         def generate_split_l(z):
             split=int(480/z)
             BendingInstance_l = [[0 for x in range(23)] for y in range(64*z)]
             for l in range(1,z+1):
                 for i in range(0,9):
                     tempDfp = pd.read_csv(bendingtrainSetPath[i],skiprows=5,header=None)
                     tempDf=tempDfp.values
                     rangeStart=(l-1)*split
                     rangeEnd=l*split
                     timeSeries1=tempDf[rangeStart:rangeEnd,1]#spilt the 6 series
                     timeSeries2=tempDf[rangeStart:rangeEnd,2]
                     timeSeries6=tempDf[rangeStart:rangeEnd,6]
```

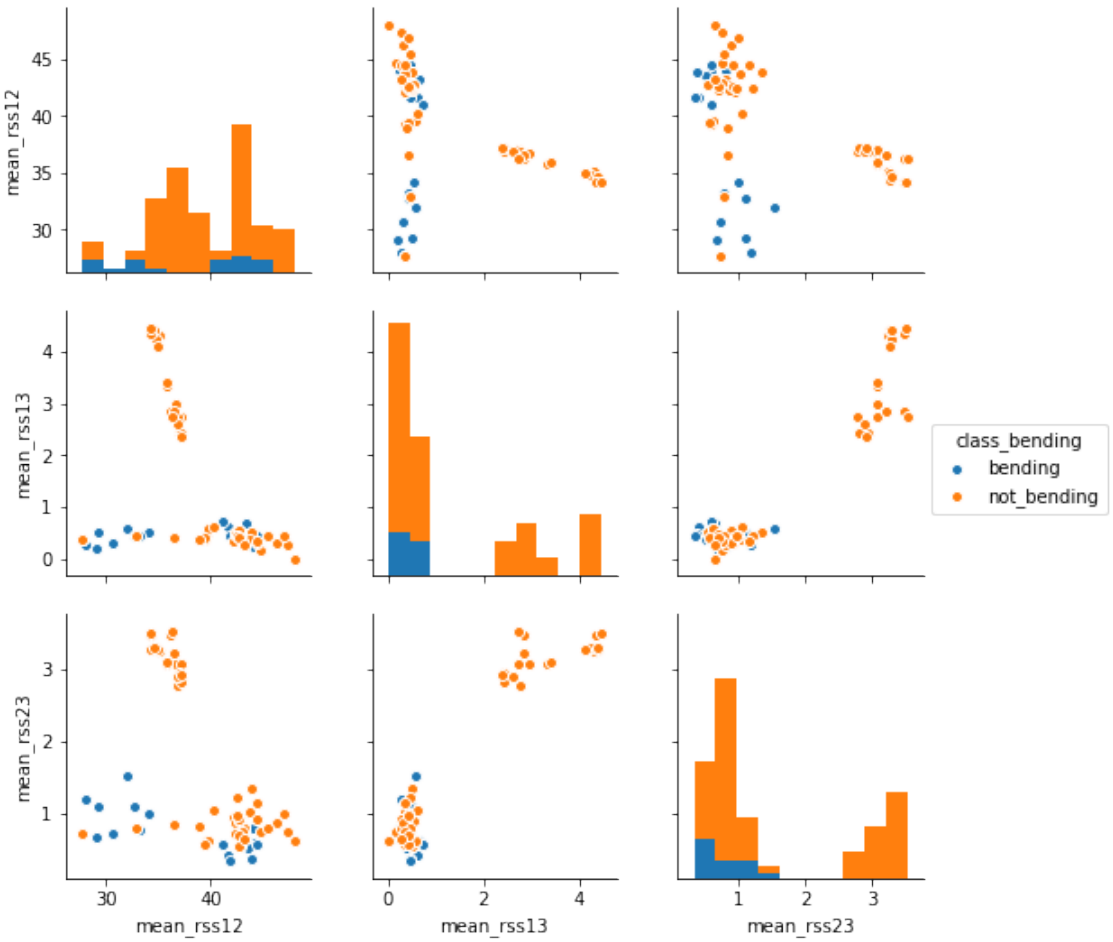
```
maxtS1=np.max(timeSeries1)#get the max
maxtS2=np.max(timeSeries2)
maxtS6=np.max(timeSeries6)

mintS1=np.min(timeSeries1)#get the min
mintS2=np.min(timeSeries2)
mintS6=np.min(timeSeries6)

meantS1=np.mean(timeSeries1)#get the mean
meantS2=np.mean(timeSeries2)
meantS6=np.mean(timeSeries6)

mediantS1=np.median(timeSeries1)#get the median
mediantS2=np.median(timeSeries2)
mediantS6=np.median(timeSeries6)

stdtS1=np.std(timeSeries1)#get the std
stdtS2=np.std(timeSeries2)
stdtS6=np.std(timeSeries6)

quart_25_S1=np.percentile(timeSeries1,25)#get the first quart
quart_25_S2=np.percentile(timeSeries2,25)
quart_25_S6=np.percentile(timeSeries6,25)

quart_75_S1=np.percentile(timeSeries1,75)#get the third quart
quart_75_S2=np.percentile(timeSeries2,75)
quart_75_S6=np.percentile(timeSeries6,75)

BendingInstance_l[i+(l-1)*64][0]=maxtS1 #save different data into the mat
BendingInstance_l[i+(l-1)*64][1]=mintS1
BendingInstance_l[i+(l-1)*64][2]=meantS1
BendingInstance_l[i+(l-1)*64][3]=mediantS1
BendingInstance_l[i+(l-1)*64][4]=stdtS1
BendingInstance_l[i+(l-1)*64][5]=quart_25_S1
BendingInstance_l[i+(l-1)*64][6]=quart_75_S1

BendingInstance_l[i+(l-1)*64][7]=maxtS2
BendingInstance_l[i+(l-1)*64][8]=mintS2
BendingInstance_l[i+(l-1)*64][9]=meantS2
BendingInstance_l[i+(l-1)*64][10]=mediantS2
BendingInstance_l[i+(l-1)*64][11]=stdtS2
BendingInstance_l[i+(l-1)*64][12]=quart_25_S2
BendingInstance_l[i+(l-1)*64][13]=quart_75_S2

BendingInstance_l[i+(l-1)*64][14]=maxtS6
BendingInstance_l[i+(l-1)*64][15]=mintS6
BendingInstance_l[i+(l-1)*64][16]=meantS6
```

```
            BendingInstance_l[i+(l-1)*64][17]=mediantS6
            BendingInstance_l[i+(l-1)*64][18]=stdtS6
            BendingInstance_l[i+(l-1)*64][19]=quart_25_S6
            BendingInstance_l[i+(l-1)*64][20]=quart_75_S6
            BendingInstance_l[i+(l-1)*64][21]='bending'
            BendingInstance_l[i+(l-1)*64][22]=str('l'+str(z)+'_'+str(l))

        for i in range(9,64):
            tempDfp = pd.read_csv(notbendingtrainSetPath[i-9],skiprows=5,header=None)
            tempDf=tempDfp.values
            timeSeries1=tempDf[:,1]#spilt the 6 series
            timeSeries2=tempDf[:,2]
            timeSeries6=tempDf[:,6]

            maxtS1=np.max(timeSeries1)#get the max
            maxtS2=np.max(timeSeries2)
            maxtS6=np.max(timeSeries6)

            mintS1=np.min(timeSeries1)#get the min
            mintS2=np.min(timeSeries2)
            mintS6=np.min(timeSeries6)

            meantS1=np.mean(timeSeries1)#get the mean
            meantS2=np.mean(timeSeries2)
            meantS6=np.mean(timeSeries6)

            mediantS1=np.median(timeSeries1)#get the median
            mediantS2=np.median(timeSeries2)
            mediantS6=np.median(timeSeries6)

            stdtS1=np.std(timeSeries1)#get the std
            stdtS2=np.std(timeSeries2)
            stdtS6=np.std(timeSeries6)

            quart_25_S1=np.percentile(timeSeries1,25)#get the first quart
            quart_25_S2=np.percentile(timeSeries2,25)
            quart_25_S6=np.percentile(timeSeries6,25)

            quart_75_S1=np.percentile(timeSeries1,75)#get the third quart
            quart_75_S2=np.percentile(timeSeries2,75)
            quart_75_S6=np.percentile(timeSeries6,75)

            BendingInstance_l[i+(l-1)*64][0]=maxtS1 #save different data into the mat
            BendingInstance_l[i+(l-1)*64][1]=mintS1
            BendingInstance_l[i+(l-1)*64][2]=meantS1
            BendingInstance_l[i+(l-1)*64][3]=mediantS1
            BendingInstance_l[i+(l-1)*64][4]=stdtS1
            BendingInstance_l[i+(l-1)*64][5]=quart_25_S1
```

```
                    BendingInstance_l[i+(l-1)*64][6]=quart_75_S1

                    BendingInstance_l[i+(l-1)*64][7]=maxtS2
                    BendingInstance_l[i+(l-1)*64][8]=mintS2
                    BendingInstance_l[i+(l-1)*64][9]=meantS2
                    BendingInstance_l[i+(l-1)*64][10]=mediantS2
                    BendingInstance_l[i+(l-1)*64][11]=stdtS2
                    BendingInstance_l[i+(l-1)*64][12]=quart_25_S2
                    BendingInstance_l[i+(l-1)*64][13]=quart_75_S2

                    BendingInstance_l[i+(l-1)*64][14]=maxtS6
                    BendingInstance_l[i+(l-1)*64][15]=mintS6
                    BendingInstance_l[i+(l-1)*64][16]=meantS6
                    BendingInstance_l[i+(l-1)*64][17]=mediantS6
                    BendingInstance_l[i+(l-1)*64][18]=stdtS6
                    BendingInstance_l[i+(l-1)*64][19]=quart_25_S6
                    BendingInstance_l[i+(l-1)*64][20]=quart_75_S6
                    BendingInstance_l[i+(l-1)*64][21]='not_bending'
                    BendingInstance_l[i+(l-1)*64][22]=str('l'+str(z)+'_'+str(l))
            BendingInstanceDf_l = pd.DataFrame(data=BendingInstance_l,index=None,columns=None)
            BendingInstanceDf_l.columns=['max_rss12', 'min_rss12','mean_rss12','median_rss12'
            return BendingInstanceDf_l
        #get the dataframe of bending instance

In [13]: split_l_2=generate_split_l(2)

In [14]: max_Bending=split_l_2[['max_rss12','max_rss13','max_rss23','class_bending']]
         mean_Bending=split_l_2[['mean_rss12','mean_rss13','mean_rss23','class_bending']]
         std_Bending=split_l_2[['std_rss12','std_rss13','std_rss23','class_bending']]
         median_Bending=split_l_2[['median_rss12','median_rss13','median_rss23','class_bending
         quart25_Bending=split_l_2[['quart25_rss12','quart25_rss13','quart25_rss23','class_ben
         quart75_Bending=split_l_2[['quart75_rss12','quart75_rss13','quart75_rss23','class_ben

In [15]: sns.pairplot(max_Bending,hue="class_bending")

Out[15]: <seaborn.axisgrid.PairGrid at 0x1b4826de080>
```

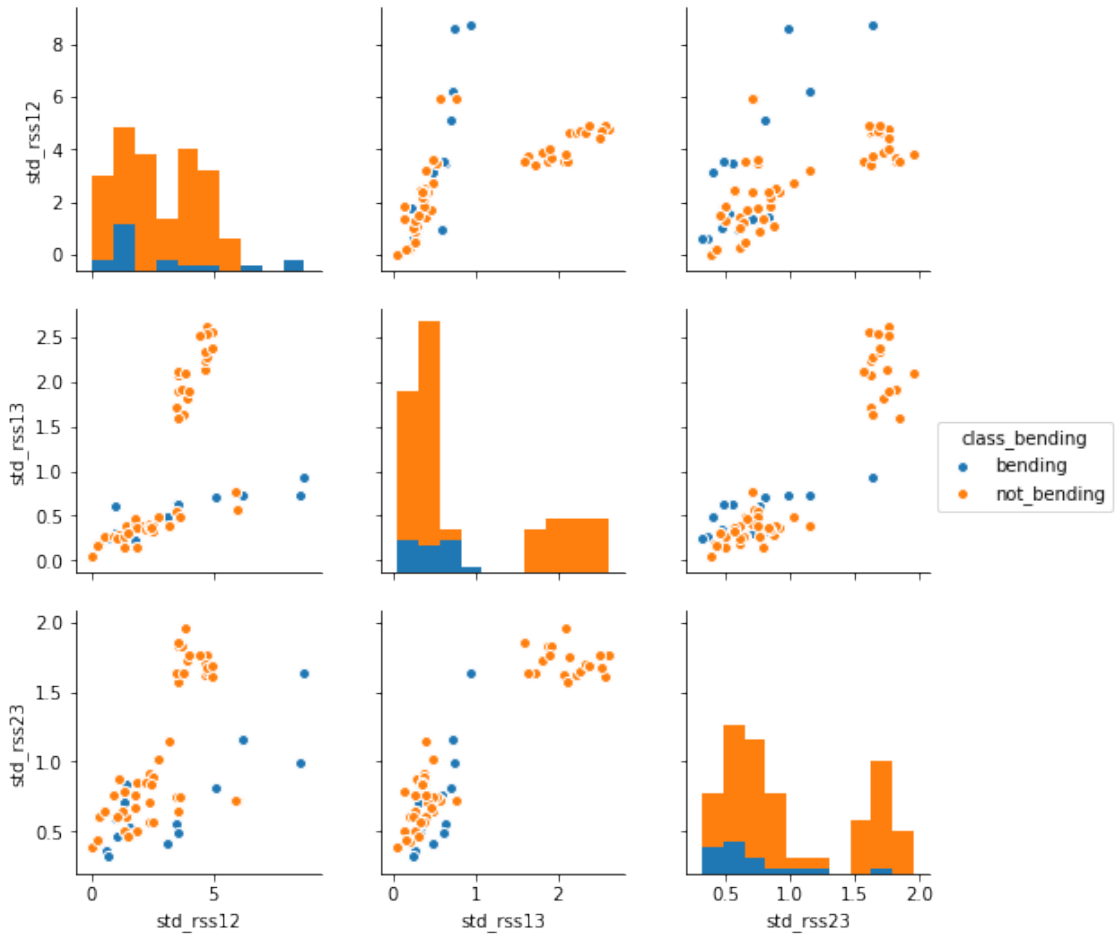



```
In [20]: sns.pairplot(mean_Bending,hue="class_bending")

Out[20]: <seaborn.axisgrid.PairGrid at 0x25a13326898>
```



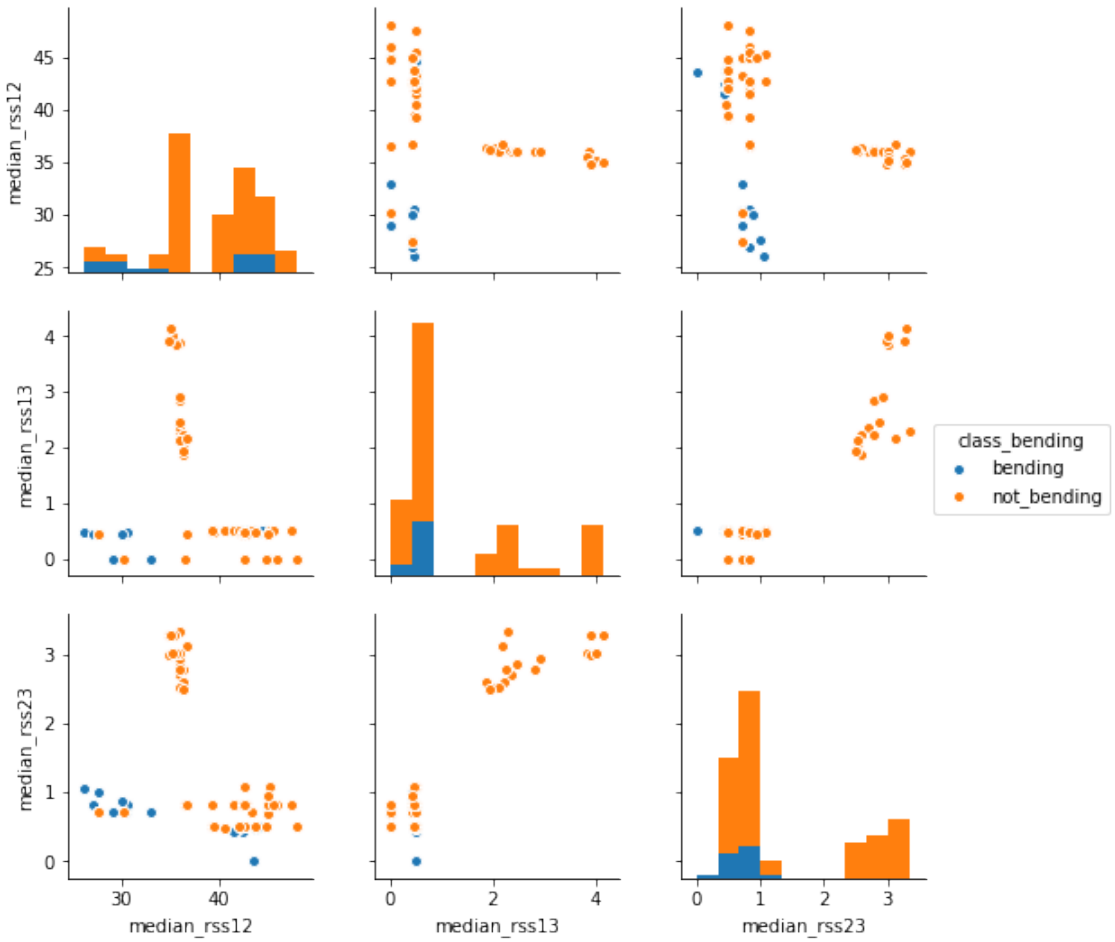

```
In [21]: sns.pairplot(std_Bending,hue="class_bending")
```

```
Out[21]: <seaborn.axisgrid.PairGrid at 0x25a15164160>
```



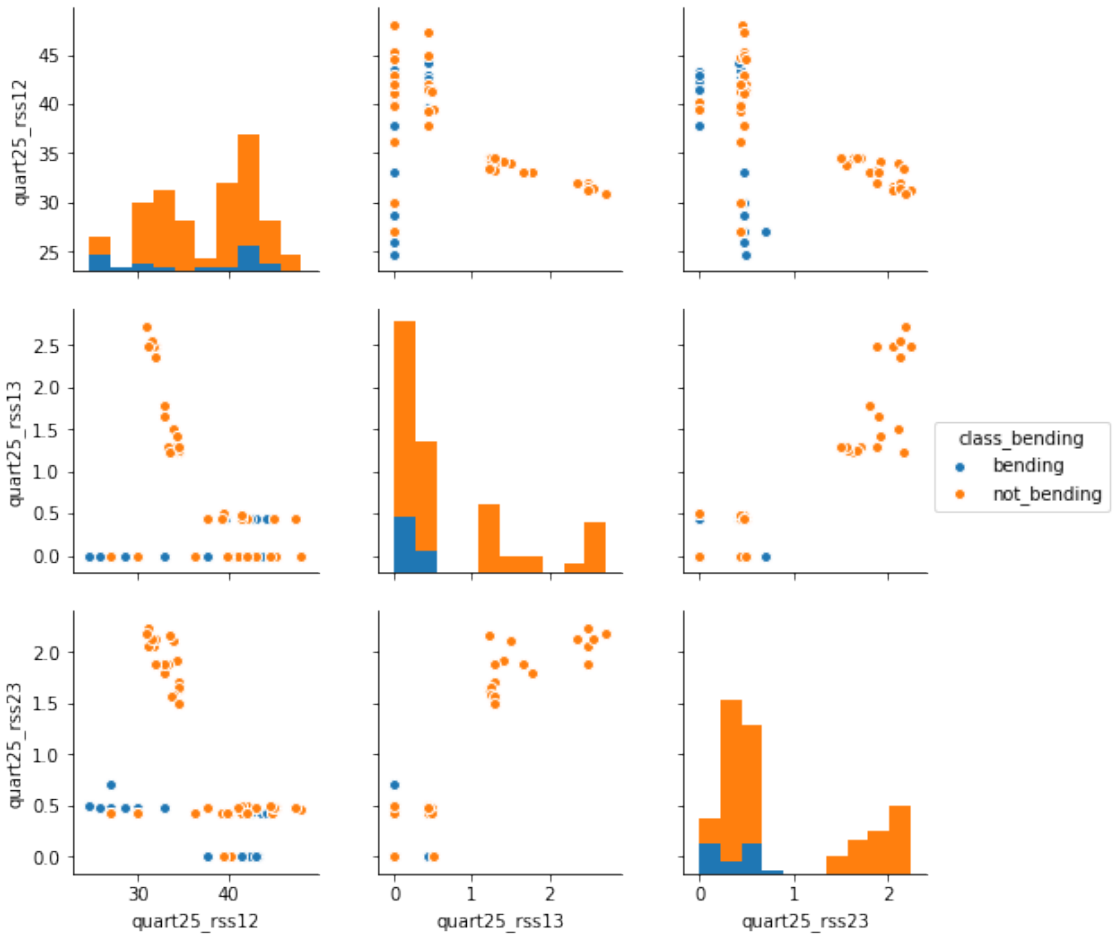

```
In [22]: sns.pairplot(median_Bending,hue="class_bending")
```

```
Out[22]: <seaborn.axisgrid.PairGrid at 0x25a15431208>
```



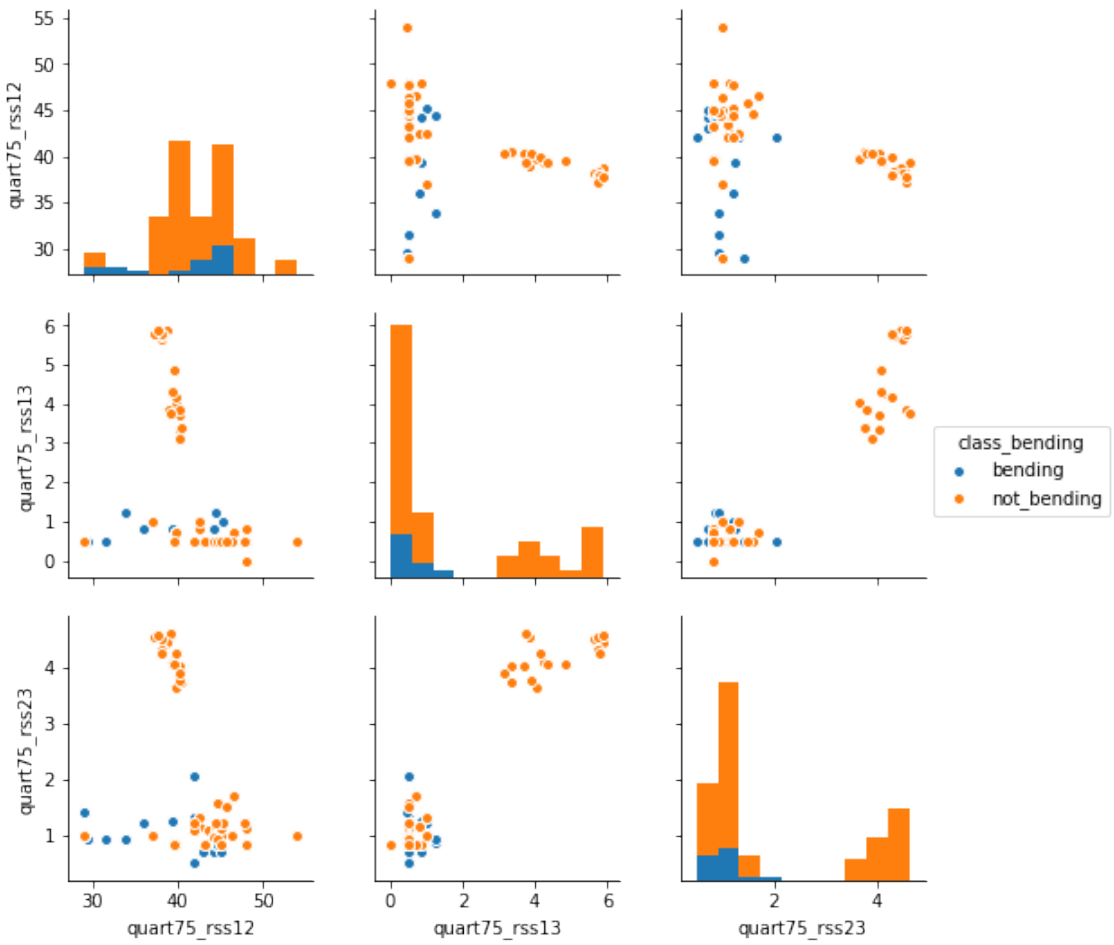

```
In [23]: sns.pairplot(quart25_Bending,hue="class_bending")
```

```
Out[23]: <seaborn.axisgrid.PairGrid at 0x25a15164dd8>
```




```
In [24]: sns.pairplot(quart75_Bending,hue="class_bending")
         #********************end of d_ii*****************
```

```
Out[24]: <seaborn.axisgrid.PairGrid at 0x25a16922518>
```



7.1 We can see the point doubled but there are only little difference compare with before

8 end of d_ii

9 d_iii

```
In [16]: #******************d_iii**********************
         import statsmodels.api as sm
         from sklearn.linear_model.logistic import LogisticRegression
         from sklearn.feature_selection import f_regression
         from sklearn.model_selection import cross_val_score
         import matplotlib.pyplot as plt

In [17]: def generate_split_l_training(z):#split data l to 1-20
             split=int(480/z)
             Instance_l = [[0 for x in range(23)] for y in range(88*z)]
```

```
for l in range(1,z+1):
    for i in range(0,88):
        tempDfp = pd.read_csv(totalSetPath[i],skiprows=5,header=None)
        tempDf=tempDfp.values
        rangeStart=(l-1)*split
        rangeEnd=l*split
        timeSeries1=tempDf[rangeStart:rangeEnd,1]#spilt the 6 series
        timeSeries2=tempDf[rangeStart:rangeEnd,2]
        timeSeries6=tempDf[rangeStart:rangeEnd,6]

        maxtS1=np.max(timeSeries1)#get the max
        maxtS2=np.max(timeSeries2)
        maxtS6=np.max(timeSeries6)

        mintS1=np.min(timeSeries1)#get the min
        mintS2=np.min(timeSeries2)
        mintS6=np.min(timeSeries6)

        meantS1=np.mean(timeSeries1)#get the mean
        meantS2=np.mean(timeSeries2)
        meantS6=np.mean(timeSeries6)

        mediantS1=np.median(timeSeries1)#get the median
        mediantS2=np.median(timeSeries2)
        mediantS6=np.median(timeSeries6)

        stdtS1=np.std(timeSeries1)#get the std
        stdtS2=np.std(timeSeries2)
        stdtS6=np.std(timeSeries6)

        quart_25_S1=np.percentile(timeSeries1,25)#get the first quart
        quart_25_S2=np.percentile(timeSeries2,25)
        quart_25_S6=np.percentile(timeSeries6,25)

        quart_75_S1=np.percentile(timeSeries1,75)#get the third quart
        quart_75_S2=np.percentile(timeSeries2,75)
        quart_75_S6=np.percentile(timeSeries6,75)

        Instance_l[i+(l-1)*88][0]=maxtS1 #save different data into the matrix Ins
        Instance_l[i+(l-1)*88][1]=mintS1
        Instance_l[i+(l-1)*88][2]=meantS1
        Instance_l[i+(l-1)*88][3]=mediantS1
        Instance_l[i+(l-1)*88][4]=stdtS1
        Instance_l[i+(l-1)*88][5]=quart_25_S1
        Instance_l[i+(l-1)*88][6]=quart_75_S1

        Instance_l[i+(l-1)*88][7]=maxtS2
        Instance_l[i+(l-1)*88][8]=mintS2
```

```
                    Instance_l[i+(l-1)*88][9]=meantS2
                    Instance_l[i+(l-1)*88][10]=mediantS2
                    Instance_l[i+(l-1)*88][11]=stdtS2
                    Instance_l[i+(l-1)*88][12]=quart_25_S2
                    Instance_l[i+(l-1)*88][13]=quart_75_S2

                    Instance_l[i+(l-1)*88][14]=maxtS6
                    Instance_l[i+(l-1)*88][15]=mintS6
                    Instance_l[i+(l-1)*88][16]=meantS6
                    Instance_l[i+(l-1)*88][17]=mediantS6
                    Instance_l[i+(l-1)*88][18]=stdtS6
                    Instance_l[i+(l-1)*88][19]=quart_25_S6
                    Instance_l[i+(l-1)*88][20]=quart_75_S6
                    Instance_l[i+(l-1)*88][22]=str('l'+str(z)+'_'+str(l))
                    if(0<i+1<14):
                        Instance_l[i+(l-1)*88][21]=1 #'bending'
                    if(13<i+1<29):
                        Instance_l[i+(l-1)*88][21]=0 #'cycling'
                    if(28<i+1<44):
                        Instance_l[i+(l-1)*88][21]=0 #'lying'
                    if(43<i+1<59):
                        Instance_l[i+(l-1)*88][21]=0 #'sitting'
                    if(58<i+1<74):
                        Instance_l[i+(l-1)*88][21]=0 #'standing'
                    if(73<i+1<89):
                        Instance_l[i+(l-1)*88][21]=0 #'walking'
            InstanceDf_l = pd.DataFrame(data=Instance_l,index=None,columns=None)
            InstanceDf_l.columns=['max_rss12', 'min_rss12','mean_rss12','median_rss12','std_rs
            return InstanceDf_l
        #get the dataframe of instance

In [18]: l_value=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
         accruacy_value_before=[]

In [19]: #before feature selection get the accrucy of 5 cv LR
         for l in range(1,21):
             temp_df=generate_split_l_training(l)
             trainSetX=temp_df[['max_rss12', 'min_rss12','mean_rss12','median_rss12','std_rss1
             trainSetY=temp_df['class']
             LR_classify=LogisticRegression()#use LogisticRegression to predict the value
             train_result=LR_classify.fit(trainSetX,trainSetY)
             c=cross_val_score(LR_classify, trainSetX, trainSetY,cv=5)#cross validation=5
             accrucy=np.mean(c)
             print('The predict accrucy of l= '+str(l)+' is '+accrucy.astype('str'))
             accruacy_value_before.append(accrucy)

The predict accrucy of l= 1 is 0.8633986928104574
The predict accrucy of l= 2 is 0.86984126984127
```

```
The predict accrucy of l= 3 is 0.8560957910014515
The predict accrucy of l= 4 is 0.863702213279678
The predict accrucy of l= 5 is 0.8545454545454547
The predict accrucy of l= 6 is 0.8541419586702605
The predict accrucy of l= 7 is 0.8555337004982952
The predict accrucy of l= 8 is 0.8593617021276595
The predict accrucy of l= 9 is 0.8674388981768969
The predict accrucy of l= 10 is 0.8659090909090909
The predict accrucy of l= 11 is 0.8625874686181294
The predict accrucy of l= 12 is 0.8645846373960475
The predict accrucy of l= 13 is 0.8619053091243393
The predict accrucy of l= 14 is 0.856344425792436
The predict accrucy of l= 15 is 0.8643939393939395
The predict accrucy of l= 16 is 0.8600817748163851
The predict accrucy of l= 17 is 0.8522764771460423
The predict accrucy of l= 18 is 0.854176815876692
The predict accrucy of l= 19 is 0.8594423094110286
The predict accrucy of l= 20 is 0.8602272727272726
```

```
In [20]: np.max(accruacy_value_before)#highest predict rate before feature selection
```

```
Out[20]: 0.86984126984127
```

we use all the all the feature to do cross validition I think it's wrong way of cross validition. Let I use p_value for feature selection.

```
In [23]: temp_df=generate_split_l_training(8)
         trainSetX=temp_df[['max_rss12', 'min_rss12','mean_rss12','median_rss12','std_rss12','
         trainSetY=temp_df['class']
         p_model=sm.Logit(trainSetY, trainSetX).fit(method='bfgs')
         pvalues=p_model.pvalues
         pvalues#get p_value for feature selction, every p_value higher than 50% we drop it
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.266080
         Iterations: 35
         Function evaluations: 40
         Gradient evaluations: 40
```

```
Out[23]: max_rss12       0.001081
         min_rss12       0.163573
         mean_rss12      0.131694
         median_rss12    0.044159
         std_rss12       0.060160
         quart25_rss12   0.296792
         quart75_rss12   0.054719
         max_rss13       0.394571
```

```
min_rss13        0.809965
mean_rss13       0.860762
median_rss13     0.676833
std_rss13        0.712441
quart25_rss13    0.141672
quart75_rss13    0.353580
max_rss23        0.239245
min_rss23        0.989344
mean_rss23       0.735656
median_rss23     0.589484
std_rss23        0.594711
quart25_rss23    0.010286
quart75_rss23    0.201453
dtype: float64
```

form p value we see max_rss13 min_rss13 mean_rss13 median_rss13 std_rss13 quart75_rss13 min_rss23 mean_rss23 median_rss23 std_rss23 has too high p value so we drop these feature

```
In [24]: l_value=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
         accruacy_value=[]
```

```
In [25]: for l in range(1,21):
             temp_df=generate_split_l_training(l)
             trainSetX=temp_df[['max_rss12', 'min_rss12','mean_rss12','median_rss12','quart25_r
             #feature after selection
             trainSetY=temp_df['class']
             LR_classify=LogisticRegression()#use LogisticRegression to predict the value
             train_result=LR_classify.fit(trainSetX,trainSetY)
             c=cross_val_score(LR_classify, trainSetX, trainSetY,cv=5)#cross validation=5
             accrucy=np.mean(c)
             accruacy_value.append(accrucy)
             print('The predict accrucy of l= '+str(l)+' is '+accrucy.astype('str'))
```

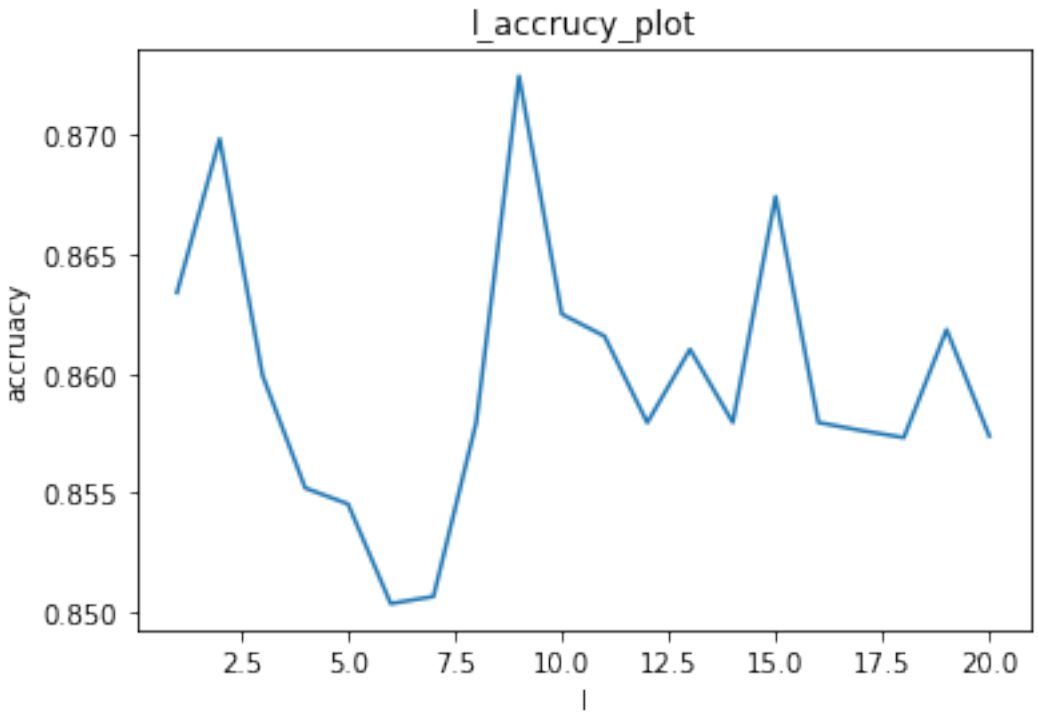
```
The predict accrucy of l= 1 is 0.8633986928104574
The predict accrucy of l= 2 is 0.86984126984127
The predict accrucy of l= 3 is 0.8599419448476052
The predict accrucy of l= 4 is 0.8552112676056337
The predict accrucy of l= 5 is 0.8545454545454545
The predict accrucy of l= 6 is 0.8503683737646002
The predict accrucy of l= 7 is 0.8506687647521638
The predict accrucy of l= 8 is 0.8579635258358662
The predict accrucy of l= 9 is 0.8724703447177772
The predict accrucy of l= 10 is 0.8625
The predict accrucy of l= 11 is 0.8615725655680787
The predict accrucy of l= 12 is 0.8579495663060002
The predict accrucy of l= 13 is 0.8610242856048419
The predict accrucy of l= 14 is 0.8579605674599255
The predict accrucy of l= 15 is 0.8674242424242425
The predict accrucy of l= 16 is 0.8579591630700891
```

```
The predict accrucy of l= 17 is 0.8576209587513934
The predict accrucy of l= 18 is 0.8573254003114643
The predict accrucy of l= 19 is 0.8618482438108858
The predict accrucy of l= 20 is 0.8573863636363637
```

```
In [26]: np.max(accruacy_value)

Out[26]: 0.8724703447177772
```

At this time we have higher predict accruacy after feature selection So we can know the wrong way to do cross validition is do it before feature selection, the wrong way is do it after feature selection *****in this time we get l=9 wiht highest accrucy of 87.24%*****

```
In [27]: plt.figure()
         plt.plot(l_value,accruacy_value)
         plt.xlabel('l')
         plt.ylabel('accruacy')
         plt.title('l_accrucy_plot')#get the plot of accrucy vs l
         #********************end of d_iii************************

Out[27]: Text(0.5,1,'l_accrucy_plot')
```



10 end of d_iii

11 d_iv

```
In [81]: #******************d_iv**************************
         from sklearn.metrics import confusion_matrix
         temp_df=generate_split_l_training(9)#choose the best performace l
         trainSetX=temp_df[['max_rss12', 'min_rss12','mean_rss12','median_rss12','quart25_rss1
         trainSetY=temp_df['class']
         trainSetX.columns.values
         #the feature I use to do logistic regression with l =9

Out[81]: array(['max_rss12', 'min_rss12', 'mean_rss12', 'median_rss12',
                'quart25_rss12', 'quart75_rss12', 'max_rss13', 'max_rss23',
                'min_rss23', 'quart25_rss23', 'quart75_rss23'], dtype=object)

In [82]: p_model=sm.Logit(trainSetY, trainSetX).fit(method='bfgs')
         pvalues=p_model.pvalues
         pvalues
         #p_value of the parameter

Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.278789
         Iterations: 35
         Function evaluations: 40
         Gradient evaluations: 40


Out[82]: max_rss12        2.430657e-05
         min_rss12        6.233161e-01
         mean_rss12       1.397305e-03
         median_rss12     2.041815e-01
         quart25_rss12    5.621480e-02
         quart75_rss12    1.255204e-03
         max_rss13        5.435988e-02
         max_rss23        8.226471e-02
         min_rss23        8.981710e-01
         quart25_rss23    4.540790e-11
         quart75_rss23    4.637750e-02
         dtype: float64

In [84]: train_result=LR_classify.fit(trainSetX,trainSetY)
         c=cross_val_score(LR_classify, trainSetX, trainSetY,cv=5)#cross validation=5
         predict_result=train_result.predict(trainSetX)
         tn, fp, fn, tp = confusion_matrix(predict_result,trainSetY).ravel()#get the con
         print ('tn:'+str(tn))
         print ('fp:'+str(fp))
         print ('fn:'+str(fn))
         print ('tp:'+str(tp))
```

```
        #confusion matrix
        confusion_matrix(predict_result,trainSetY)
```

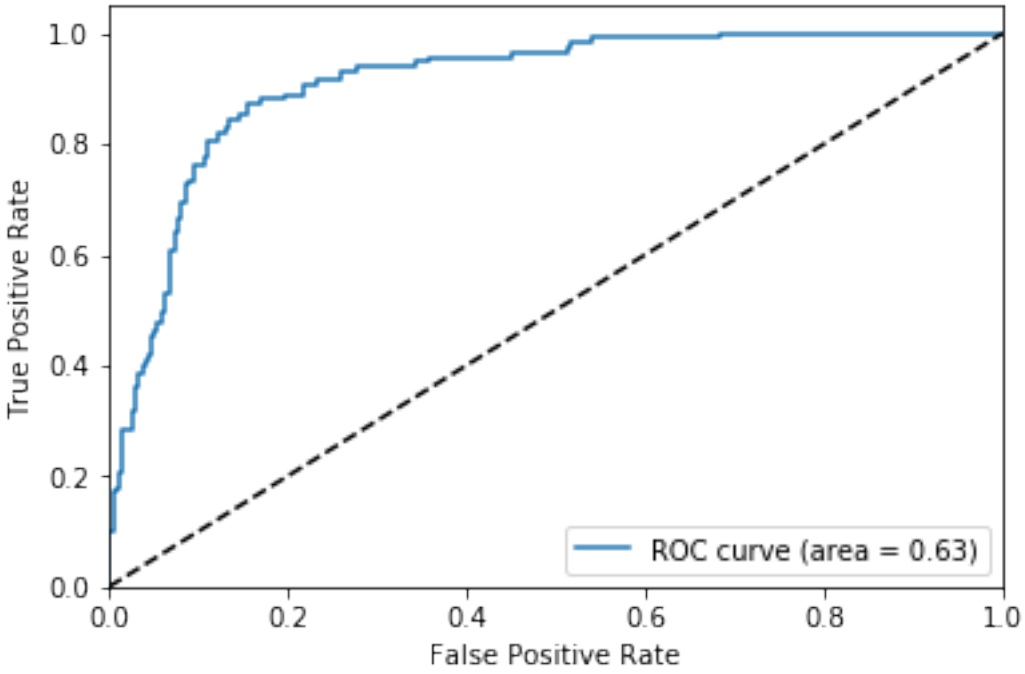
```
tn:663
fp:84
fn:12
tp:33
```

```
Out[84]: array([[663,  84],
                [ 12,  33]], dtype=int64)
```

```
In [31]: for i in range(0,11):
             coefficient=str(train_result.coef_[0][i])
             featurename=str(trainSetX.columns.values[i])
             print('the coefficient of ' +featurename+' is ' +coefficient)
         #coefficient of feature
```

```
the coefficient of max_rss12 is 0.29054082990310554
the coefficient of min_rss12 is -0.013096224238079034
the coefficient of mean_rss12 is -1.3231456305106981
the coefficient of median_rss12 is 0.1275660161769023
the coefficient of quart25_rss12 is 0.18903634665755484
the coefficient of quart75_rss12 is 0.6368027158252618
the coefficient of max_rss13 is -0.25191841425330147
the coefficient of max_rss23 is -0.26427744251146335
the coefficient of min_rss23 is 0.0005329345480167791
the coefficient of quart25_rss23 is -3.04303528258379
the coefficient of quart75_rss23 is 0.2265003029205297
```

```
In [32]: from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve, auc
         logit=roc_auc_score(trainSetY,predict_result)
         y_scores=LR_classify.decision_function(trainSetX)
         fp_rate,tp_rate,Th=roc_curve(trainSetY,y_scores)
         plt.figure()
         plt.plot(fp_rate,tp_rate,label='ROC curve (area = %0.2f)'%logit)
         plt.plot([0, 1], [0, 1], 'k--')
         plt.xlim([0.0, 1.0])
         plt.ylim([0.0, 1.05])
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.legend(loc="lower right")
         plt.show()
         #ROC_plot
```



12 end of d_iv

```
In [33]: #******************d_v*************************
         def generate_split_l_testing(z):#split train data l to 1-20
             split=int(480/z)
             Instance_l = [[0 for x in range(23)] for y in range(19*z)]
             for l in range(1,z+1):
                 for i in range(0,19):
                     tempDfp = pd.read_csv(testSetPath[i],skiprows=5,header=None)
                     tempDf=tempDfp.values
                     rangeStart=(l-1)*split
                     rangeEnd=l*split
                     timeSeries1=tempDf[rangeStart:rangeEnd,1]#spilt the 6 series
                     timeSeries2=tempDf[rangeStart:rangeEnd,2]
                     timeSeries6=tempDf[rangeStart:rangeEnd,6]

                     maxtS1=np.max(timeSeries1)#get the max
                     maxtS2=np.max(timeSeries2)
                     maxtS6=np.max(timeSeries6)

                     mintS1=np.min(timeSeries1)#get the min
                     mintS2=np.min(timeSeries2)
                     mintS6=np.min(timeSeries6)
```

```
meantS1=np.mean(timeSeries1)#get the mean
meantS2=np.mean(timeSeries2)
meantS6=np.mean(timeSeries6)

mediantS1=np.median(timeSeries1)#get the median
mediantS2=np.median(timeSeries2)
mediantS6=np.median(timeSeries6)

stdtS1=np.std(timeSeries1)#get the std
stdtS2=np.std(timeSeries2)
stdtS6=np.std(timeSeries6)

quart_25_S1=np.percentile(timeSeries1,25)#get the first quart
quart_25_S2=np.percentile(timeSeries2,25)
quart_25_S6=np.percentile(timeSeries6,25)

quart_75_S1=np.percentile(timeSeries1,75)#get the third quart
quart_75_S2=np.percentile(timeSeries2,75)
quart_75_S6=np.percentile(timeSeries6,75)

Instance_l[i+(l-1)*19][0]=maxtS1 #save different data into the matrix Ins
Instance_l[i+(l-1)*19][1]=mintS1
Instance_l[i+(l-1)*19][2]=meantS1
Instance_l[i+(l-1)*19][3]=mediantS1
Instance_l[i+(l-1)*19][4]=stdtS1
Instance_l[i+(l-1)*19][5]=quart_25_S1
Instance_l[i+(l-1)*19][6]=quart_75_S1

Instance_l[i+(l-1)*19][7]=maxtS2
Instance_l[i+(l-1)*19][8]=mintS2
Instance_l[i+(l-1)*19][9]=meantS2
Instance_l[i+(l-1)*19][10]=mediantS2
Instance_l[i+(l-1)*19][11]=stdtS2
Instance_l[i+(l-1)*19][12]=quart_25_S2
Instance_l[i+(l-1)*19][13]=quart_75_S2

Instance_l[i+(l-1)*19][14]=maxtS6
Instance_l[i+(l-1)*19][15]=mintS6
Instance_l[i+(l-1)*19][16]=meantS6
Instance_l[i+(l-1)*19][17]=mediantS6
Instance_l[i+(l-1)*19][18]=stdtS6
Instance_l[i+(l-1)*19][19]=quart_25_S6
Instance_l[i+(l-1)*19][20]=quart_75_S6
Instance_l[i+(l-1)*19][22]=str('l'+str(z)+'_'+str(l))
if(0<i+1<5):
    Instance_l[i+(l-1)*19][21]=1 #'bending'
if(4<i+1<20):
    Instance_l[i+(l-1)*19][21]=0 #'not_bending'
```

```
            InstanceDf_l = pd.DataFrame(data=Instance_l,index=None,columns=None)
            InstanceDf_l.columns=['max_rss12', 'min_rss12','mean_rss12','median_rss12','std_rs
            return InstanceDf_l
```

```
In [35]: for l in range(1,21):
            temp_df=generate_split_l_testing(l)
            trainSetX_test=temp_df[['max_rss12', 'min_rss12','mean_rss12','median_rss12','quar
            #feature after selection
            trainSetY_test=temp_df['class']
            LR_classify=LogisticRegression()#use LogisticRegression to predict the value
            train_result=LR_classify.fit(trainSetX_test,trainSetY_test)
            c=cross_val_score(LR_classify, trainSetX_test, trainSetY_test,cv=5)#cross validat
            accrucy=np.mean(c)
            accruacy_value.append(accrucy)
            print('The predict accrucy of l= '+str(l)+' is '+accrucy.astype('str'))
```

```
The predict accrucy of l= 1 is 0.95
The predict accrucy of l= 2 is 0.8428571428571429
The predict accrucy of l= 3 is 0.7924242424242424
The predict accrucy of l= 4 is 0.7908333333333334
The predict accrucy of l= 5 is 0.8105263157894737
The predict accrucy of l= 6 is 0.7810276679841899
The predict accrucy of l= 7 is 0.8202279202279202
The predict accrucy of l= 8 is 0.8161290322580644
The predict accrucy of l= 9 is 0.7897478991596637
The predict accrucy of l= 10 is 0.8105263157894737
The predict accrucy of l= 11 is 0.8229965156794427
The predict accrucy of l= 12 is 0.8248309178743962
The predict accrucy of l= 13 is 0.8018775510204081
The predict accrucy of l= 14 is 0.8046820405310973
The predict accrucy of l= 15 is 0.8175438596491228
The predict accrucy of l= 16 is 0.8092349726775956
The predict accrucy of l= 17 is 0.8174038461538462
The predict accrucy of l= 18 is 0.8130008525149188
The predict accrucy of l= 19 is 0.8034627092846269
The predict accrucy of l= 20 is 0.8184210526315789
```

While we use test set. We have higher accurecy=95% while l=1

13 end of d_v

14 d_vi

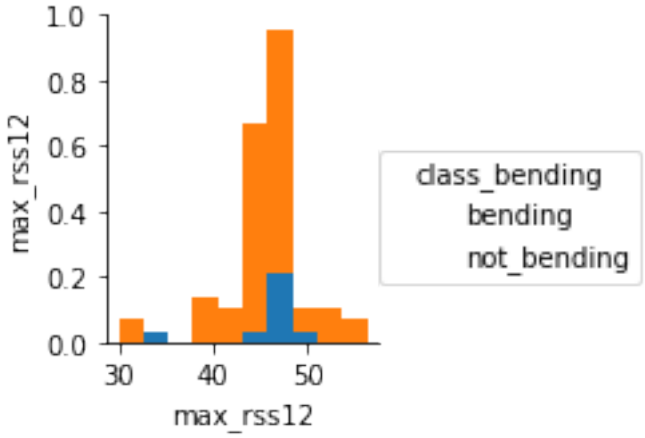
'max_rss12','min_rss12','mean_rss12','median_rss12','quart25_rss12','quart75_rss12','max_rss13','max_rss23', 'min_rss23','quart25_rss23','quart75_rss23' is the feature we use we can plot them one by one

```
In [36]: max_rss12_p=BendingInstanceDf[['max_rss12','class_bending']]
         min_rss12_p=BendingInstanceDf[['min_rss12','class_bending']]
```

```
median_rss12_p=BendingInstanceDf[['median_rss12','class_bending']]
quart25_rss12_p=BendingInstanceDf[['quart25_rss12','class_bending']]
quart75_rss12_p=BendingInstanceDf[['quart75_rss12','class_bending']]
max_rss13_p=BendingInstanceDf[['max_rss13','class_bending']]
max_rss23_p=BendingInstanceDf[['max_rss23','class_bending']]
min_rss23_p=BendingInstanceDf[['min_rss23','class_bending']]
quart25_rss23_p=BendingInstanceDf[['quart25_rss23','class_bending']]
quart75_rss23_p=BendingInstanceDf[['quart75_rss23','class_bending']]
```

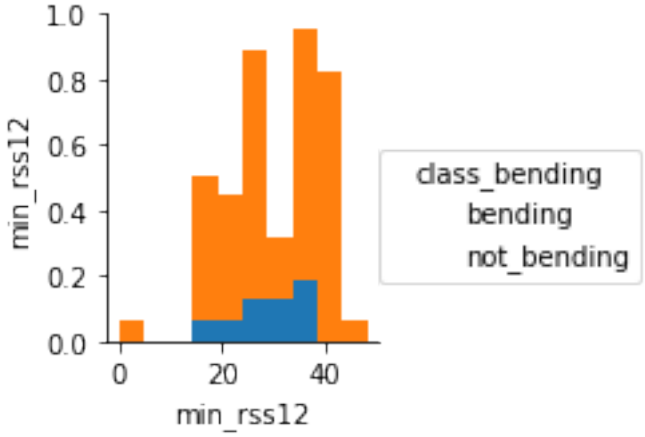
In [37]:
```
sns.pairplot(max_rss12_p,hue="class_bending")
```

Out[37]:
```
<seaborn.axisgrid.PairGrid at 0x1b486501a20>
```

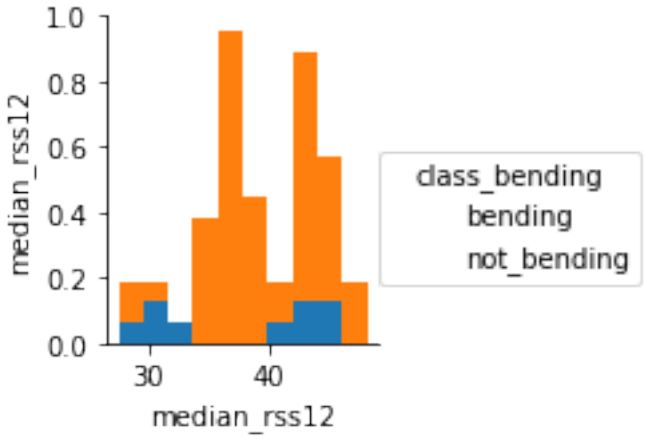


In [251]:
```
sns.pairplot(min_rss12_p,hue="class_bending")
```

Out[251]:
```
<seaborn.axisgrid.PairGrid at 0x22c39aa6780>
```



```
In [250]: sns.pairplot(median_rss12_p,hue="class_bending")

Out[250]: <seaborn.axisgrid.PairGrid at 0x22c395f1a58>
```



I only show 3 of them, it's obvious that the feature is not well-separated

15 end of d_vi

16 d_vii

```
In [255]: print ('tn:'+str(tn))
          print ('fp:'+str(fp))
          print ('fn:'+str(fn))
          print ('tp:'+str(tp))

tn:663
fp:84
fn:12
tp:33
```

we can know form the confusion matrix.It's imbalance class

17 end of d_vii

18 question e

19 e_i

```
In [58]: #*******************question e**************************
         accruacy_value=[]
```

```
        temp_df=generate_split_l_training(l)
        #use l1-penalized LogisticRegression to predict the value
        for l in range(2,21):
            c=0.01
            temp_df=generate_split_l_training(l)
            trainSetX=temp_df[['max_rss12', 'min_rss12','mean_rss12','median_rss12','std_rss1
            #use all the feature
            trainSetY=temp_df['class']
            while c<101:
                L1_classify=LogisticRegression(C=c,penalty='l1')
                train_result=L1_classify.fit(trainSetX,trainSetY)
                result_list=cross_val_score(L1_classify, trainSetX, trainSetY,cv=5)#cross val
                accrucy=np.mean(result_list)
                accruacy_value.append(accrucy)
                print('The predict accrucy of l= '+str(l)+' c= '+str(c)+' is '+accrucy.astype
                c=c*10
```

```
The predict accrucy of l= 2 c= 0.01 is 0.8523809523809524
The predict accrucy of l= 2 c= 0.1 is 0.8523809523809524
The predict accrucy of l= 2 c= 1.0 is 0.8755555555555556
The predict accrucy of l= 2 c= 10.0 is 0.8473015873015873
The predict accrucy of l= 2 c= 100.0 is 0.8473015873015873
The predict accrucy of l= 3 c= 0.01 is 0.8523222060957909
The predict accrucy of l= 3 c= 0.1 is 0.8485486211901307
The predict accrucy of l= 3 c= 1.0 is 0.8562409288824384
The predict accrucy of l= 3 c= 10.0 is 0.837155297532656
The predict accrucy of l= 3 c= 100.0 is 0.8333817126269956
The predict accrucy of l= 4 c= 0.01 is 0.8523138832997988
The predict accrucy of l= 4 c= 0.1 is 0.8523138832997988
The predict accrucy of l= 4 c= 1.0 is 0.8579879275653923
The predict accrucy of l= 4 c= 10.0 is 0.8751710261569418
The predict accrucy of l= 4 c= 100.0 is 0.863702213279678
The predict accrucy of l= 5 c= 0.01 is 0.8522727272727273
The predict accrucy of l= 5 c= 0.1 is 0.8590909090909091
The predict accrucy of l= 5 c= 1.0 is 0.8590909090909091
The predict accrucy of l= 5 c= 10.0 is 0.8681818181818182
The predict accrucy of l= 5 c= 100.0 is 0.8727272727272727
The predict accrucy of l= 6 c= 0.01 is 0.8522911051212938
The predict accrucy of l= 6 c= 0.1 is 0.8541958670260558
The predict accrucy of l= 6 c= 1.0 is 0.8485175202156334
The predict accrucy of l= 6 c= 10.0 is 0.8447079964061096
The predict accrucy of l= 6 c= 100.0 is 0.842785265049416
The predict accrucy of l= 7 c= 0.01 is 0.8522816679779701
The predict accrucy of l= 7 c= 0.1 is 0.8474298452661946
The predict accrucy of l= 7 c= 1.0 is 0.8441646997115132
The predict accrucy of l= 7 c= 10.0 is 0.8571334906897455
The predict accrucy of l= 7 c= 100.0 is 0.8619984264358772
The predict accrucy of l= 8 c= 0.01 is 0.8522796352583587
```

```
The predict accrucy of l= 8 c= 0.1 is 0.8593819655521784
The predict accrucy of l= 8 c= 1.0 is 0.8622188449848025
The predict accrucy of l= 8 c= 10.0 is 0.8608105369807497
The predict accrucy of l= 8 c= 100.0 is 0.8679128672745694
The predict accrucy of l= 9 c= 0.01 is 0.8522808693575353
The predict accrucy of l= 9 c= 0.1 is 0.8661651142424966
The predict accrucy of l= 9 c= 1.0 is 0.8762678130722076
The predict accrucy of l= 9 c= 10.0 is 0.8813311042114481
The predict accrucy of l= 9 c= 100.0 is 0.8838547886314785
The predict accrucy of l= 10 c= 0.01 is 0.8522727272727273
The predict accrucy of l= 10 c= 0.1 is 0.8625
The predict accrucy of l= 10 c= 1.0 is 0.8625
The predict accrucy of l= 10 c= 10.0 is 0.8625
The predict accrucy of l= 10 c= 100.0 is 0.8602272727272726
The predict accrucy of l= 11 c= 0.01 is 0.8522781902676139
The predict accrucy of l= 11 c= 0.1 is 0.8564179263928209
The predict accrucy of l= 11 c= 1.0 is 0.8698253298434914
The predict accrucy of l= 11 c= 10.0 is 0.8687837188184393
The predict accrucy of l= 11 c= 100.0 is 0.8667165215533359
The predict accrucy of l= 12 c= 0.01 is 0.8522757757310202
The predict accrucy of l= 12 c= 0.1 is 0.8551238486989181
The predict accrucy of l= 12 c= 1.0 is 866480371993204
The predict accrucy of l= 12 c= 10.0 is 0.8778413663596529
The predict accrucy of l= 12 c= 100.0 is 0.8749977644639184
The predict accrucy of l= 13 c= 0.01 is 0.8522753390025282
The predict accrucy of l= 13 c= 0.1 is 0.8601432620853442
The predict accrucy of l= 13 c= 1.0 is 859289052325136
The predict accrucy of l= 13 c= 10.0 is 0.8610319466789245
The predict accrucy of l= 13 c= 100.0 is 0.8601624147705508
The predict accrucy of l= 14 c= 0.01 is 0.8522760936111385
The predict accrucy of l= 14 c= 0.1 is 0.8595832921891973
The predict accrucy of l= 14 c= 1.0 is 0.8587735755900068
The predict accrucy of l= 14 c= 10.0 is 866893782298147
The predict accrucy of l= 14 c= 100.0 is 0.8677034988973371
The predict accrucy of l= 15 c= 0.01 is 0.8522727272727273
The predict accrucy of l= 15 c= 0.1 is 0.8636363636363636
The predict accrucy of l= 15 c= 1.0 is 0.8636363636363636
The predict accrucy of l= 15 c= 10.0 is 865909090909091
The predict accrucy of l= 15 c= 100.0 is 0.8674242424242424
The predict accrucy of l= 16 c= 0.01 is 852275308548497
The predict accrucy of l= 16 c= 0.1 is 0.8579591630700889
The predict accrucy of l= 16 c= 1.0 is 0.8643471896216652
The predict accrucy of l= 16 c= 10.0 is 859380126700487
The predict accrucy of l= 16 c= 100.0 is 859385174528659
The predict accrucy of l= 17 c= 0.01 is 0.8522742474916388
The predict accrucy of l= 17 c= 0.1 is 0.8582853957636567
The predict accrucy of l= 17 c= 1.0 is 0.8569565217391304
The predict accrucy of l= 17 c= 10.0 is 0.8582898550724638
```

```
The predict accrucy of l= 17 c= 100.0 is 0.8576209587513937
The predict accrucy of l= 18 c= 0.01 is 0.8522740885676636
The predict accrucy of l= 18 c= 0.1 is 0.861108892704548
The predict accrucy of l= 18 c= 1.0 is 0.8516531565706984
The predict accrucy of l= 18 c= 10.0 is 0.857960308269776
The predict accrucy of l= 18 c= 100.0 is 0.8585912230962744
The predict accrucy of l= 19 c= 0.01 is 0.852274555366878
The predict accrucy of l= 19 c= 0.1 is 0.8624398963267496
The predict accrucy of l= 19 c= 1.0 is 0.8624398963267496
The predict accrucy of l= 19 c= 10.0 is 0.8618446688712128
The predict accrucy of l= 19 c= 100.0 is 0.8618482438108858
The predict accrucy of l= 20 c= 0.01 is 0.8522727272727273
The predict accrucy of l= 20 c= 0.1 is 0.8630681818181818
The predict accrucy of l= 20 c= 1.0 is 0.8596590909090909
The predict accrucy of l= 20 c= 10.0 is 0.8642045454545455
The predict accrucy of l= 20 c= 100.0 is 0.8636363636363636
```

```
In [59]: np.max(accruacy_value)

Out[59]: 0.8838547886314785
```

we can know with l =9 c =100, we have best predict accruacy of 88.39%

20 e_ii

with p-values variable selection we get best predict accrucy of 87.24% when l =9

with L1-penalized we get best predict accruacy of 88.39% when l =9 c =100

So L1-penalized performed better and it's easiler to implement, because we don't need to get p-value and drop feature

21 end of question e

22 question f

22.1 f_i

```
In [38]: #******************d_v*************************
         def generate_split_l_multraing(z):#split data l to 1-20 for multclass
             split=int(480/z)
             Instance_l = [[0 for x in range(23)] for y in range(88*z)]
             for l in range(1,z+1):
                 for i in range(0,88):
                     tempDfp = pd.read_csv(totalSetPath[i],skiprows=5,header=None)
                     tempDf=tempDfp.values
                     rangeStart=(l-1)*split
                     rangeEnd=l*split
```

```
timeSeries1=tempDf[rangeStart:rangeEnd,1]#spilt the 6 series
timeSeries2=tempDf[rangeStart:rangeEnd,2]
timeSeries6=tempDf[rangeStart:rangeEnd,6]

maxtS1=np.max(timeSeries1)#get the max
maxtS2=np.max(timeSeries2)
maxtS6=np.max(timeSeries6)

mintS1=np.min(timeSeries1)#get the min
mintS2=np.min(timeSeries2)
mintS6=np.min(timeSeries6)

meantS1=np.mean(timeSeries1)#get the mean
meantS2=np.mean(timeSeries2)
meantS6=np.mean(timeSeries6)

mediantS1=np.median(timeSeries1)#get the median
mediantS2=np.median(timeSeries2)
mediantS6=np.median(timeSeries6)

stdtS1=np.std(timeSeries1)#get the std
stdtS2=np.std(timeSeries2)
stdtS6=np.std(timeSeries6)

quart_25_S1=np.percentile(timeSeries1,25)#get the first quart
quart_25_S2=np.percentile(timeSeries2,25)
quart_25_S6=np.percentile(timeSeries6,25)

quart_75_S1=np.percentile(timeSeries1,75)#get the third quart
quart_75_S2=np.percentile(timeSeries2,75)
quart_75_S6=np.percentile(timeSeries6,75)

Instance_l[i+(l-1)*88][0]=maxtS1 #save different data into the matrix Ins
Instance_l[i+(l-1)*88][1]=mintS1
Instance_l[i+(l-1)*88][2]=meantS1
Instance_l[i+(l-1)*88][3]=mediantS1
Instance_l[i+(l-1)*88][4]=stdtS1
Instance_l[i+(l-1)*88][5]=quart_25_S1
Instance_l[i+(l-1)*88][6]=quart_75_S1

Instance_l[i+(l-1)*88][7]=maxtS2
Instance_l[i+(l-1)*88][8]=mintS2
Instance_l[i+(l-1)*88][9]=meantS2
Instance_l[i+(l-1)*88][10]=mediantS2
Instance_l[i+(l-1)*88][11]=stdtS2
Instance_l[i+(l-1)*88][12]=quart_25_S2
Instance_l[i+(l-1)*88][13]=quart_75_S2
```

```
                    Instance_l[i+(l-1)*88][14]=maxtS6
                    Instance_l[i+(l-1)*88][15]=mintS6
                    Instance_l[i+(l-1)*88][16]=meantS6
                    Instance_l[i+(l-1)*88][17]=mediantS6
                    Instance_l[i+(l-1)*88][18]=stdtS6
                    Instance_l[i+(l-1)*88][19]=quart_25_S6
                    Instance_l[i+(l-1)*88][20]=quart_75_S6
                    Instance_l[i+(l-1)*88][22]=str('l'+str(z)+'_'+str(l))
                    if(0<i+1<8):
                        Instance_l[i+(l-1)*88][21]=1 #'bending1'
                    if(7<i+1<14):
                        Instance_l[i+(l-1)*88][21]=2 #'bending2'
                    if(13<i+1<29):
                        Instance_l[i+(l-1)*88][21]=3 #'cycling'
                    if(28<i+1<44):
                        Instance_l[i+(l-1)*88][21]=4 #'lying'
                    if(43<i+1<59):
                        Instance_l[i+(l-1)*88][21]=5 #'sitting'
                    if(58<i+1<74):
                        Instance_l[i+(l-1)*88][21]=6 #'standing'
                    if(73<i+1<89):
                        Instance_l[i+(l-1)*88][21]=7 #'walking'
            InstanceDf_l = pd.DataFrame(data=Instance_l,index=None,columns=None)
            InstanceDf_l.columns=['max_rss12', 'min_rss12','mean_rss12','median_rss12','std_rs
            return InstanceDf_l
        #get the dataframe of instance
```

In [67]:
```
        #*******************question f_i**************************
        accruacy_value=[]
        #use l1-penalized LogisticRegression to predict the  mult class value
        for l in range(2,21):
            c=0.01
            temp_df=generate_split_l_multraing(l)
            trainSetX=temp_df[['max_rss12', 'min_rss12','mean_rss12','median_rss12','std_rss1
            #use all the feature
            trainSetY=temp_df['class']
            while c<101:
                L1_multclassify=LogisticRegression(C=c,penalty='l1',multi_class='multinomial'
                train_result=L1_multclassify.fit(trainSetX,trainSetY)
                result_list=cross_val_score(L1_multclassify, trainSetX, trainSetY,cv=5)#cross
                accrucy=np.mean(result_list)
                accruacy_value.append(accrucy)
                print('The predict accrucy of l= '+str(l)+' c= '+str(c)+' is '+accrucy.astype
                c=c*10
```

```
The predict accrucy of l= 2 c= 0.01 is 0.41537815126050415
The predict accrucy of l= 2 c= 0.1 is 0.618608776844071
The predict accrucy of l= 2 c= 1.0 is 0.65812324929972
```

```
The predict accrucy of l= 2 c= 10.0 is 0.6636788048552755
The predict accrucy of l= 2 c= 100.0 is 0.65812324929972
The predict accrucy of l= 3 c= 0.01 is 0.5077621888942644
The predict accrucy of l= 3 c= 0.1 is 0.6435709294199861
The predict accrucy of l= 3 c= 1.0 is 0.6548244906735473
The predict accrucy of l= 3 c= 10.0 is 0.6471321829812396
The predict accrucy of l= 3 c= 100.0 is 0.6471321829812396
The predict accrucy of l= 4 c= 0.01 is 0.5028968010964336
The predict accrucy of l= 4 c= 0.1 is 0.6590073775989269
The predict accrucy of l= 4 c= 1.0 is 0.6730516431924882
The predict accrucy of l= 4 c= 10.0 is 0.6701542588866531
The predict accrucy of l= 4 c= 100.0 is 0.6758685446009389
The predict accrucy of l= 5 c= 0.01 is 0.5363636363636364
The predict accrucy of l= 5 c= 0.1 is 0.6340909090909091
The predict accrucy of l= 5 c= 1.0 is 0.6522727272727272
The predict accrucy of l= 5 c= 10.0 is 0.6568181818181819
The predict accrucy of l= 5 c= 100.0 is 0.6568181818181819
The predict accrucy of l= 6 c= 0.01 is 0.5396960307663886
The predict accrucy of l= 6 c= 0.1 is 0.6153151791487182
The predict accrucy of l= 6 c= 1.0 is 0.6438150657900261
The predict accrucy of l= 6 c= 10.0 is 0.6400055419805023
The predict accrucy of l= 6 c= 100.0 is 0.6400055419805023
The predict accrucy of l= 7 c= 0.01 is 0.5228450555261765
The predict accrucy of l= 7 c= 0.1 is 0.62654357611794
The predict accrucy of l= 7 c= 1.0 is 0.6330609691607229
The predict accrucy of l= 7 c= 10.0 is 0.6330609691607229
The predict accrucy of l= 7 c= 100.0 is 0.6330609691607229
The predict accrucy of l= 8 c= 0.01 is 0.534113903277823
The predict accrucy of l= 8 c= 0.1 is 0.6193034804572114
The predict accrucy of l= 8 c= 1.0 is 0.6278647773163806
The predict accrucy of l= 8 c= 10.0 is 0.6321001184411433
The predict accrucy of l= 8 c= 100.0 is 0.6306816787248313
The predict accrucy of l= 9 c= 0.01 is 0.5505144575997793
The predict accrucy of l= 9 c= 0.1 is 0.6300556215604158
The predict accrucy of l= 9 c= 1.0 is 0.6464638551131624
The predict accrucy of l= 9 c= 10.0 is 0.647729779313891
The predict accrucy of l= 9 c= 100.0 is 0.647729779313891
The predict accrucy of l= 10 c= 0.01 is 0.5465909090909091
The predict accrucy of l= 10 c= 0.1 is 0.6090909090909091
The predict accrucy of l= 10 c= 1.0 is 0.6227272727272727
The predict accrucy of l= 10 c= 10.0 is 0.6238636363636363
The predict accrucy of l= 10 c= 100.0 is 0.6227272727272728
The predict accrucy of l= 11 c= 0.01 is 0.5507397835147421
The predict accrucy of l= 11 c= 0.1 is 0.6012190615502404
The predict accrucy of l= 11 c= 1.0 is 0.615668240382732
The predict accrucy of l= 11 c= 10.0 is 0.6146213425482695
The predict accrucy of l= 11 c= 100.0 is 0.6146213425482695
The predict accrucy of l= 12 c= 0.01 is 0.5445100769450228
```

```
The predict accrucy of l= 12 c= 0.1 is 0.5984360615389857
The predict accrucy of l= 12 c= 1.0 is 0.6202373074777617
The predict accrucy of l= 12 c= 10.0 is 0.6211807037041768
The predict accrucy of l= 12 c= 100.0 is 0.6221285710027551
The predict accrucy of l= 13 c= 0.01 is 0.5489814768552289
The predict accrucy of l= 13 c= 0.1 is 0.6136032029951469
The predict accrucy of l= 13 c= 1.0 is 0.6214979065282343
The predict accrucy of l= 13 c= 10.0 is 0.6249990173839762
The predict accrucy of l= 13 c= 100.0 is 0.624121857710538
The predict accrucy of l= 14 c= 0.01 is 0.5373558662133758
The predict accrucy of l= 14 c= 0.1 is 0.6022751061518712
The predict accrucy of l= 14 c= 1.0 is 0.6161063876523257
The predict accrucy of l= 14 c= 10.0 is 0.6152933795222444
The predict accrucy of l= 14 c= 100.0 is 0.6152933795222444
The predict accrucy of l= 15 c= 0.01 is 0.5545454545454545
The predict accrucy of l= 15 c= 0.1 is 0.6053030303030303
The predict accrucy of l= 15 c= 1.0 is 0.6128787878787879
The predict accrucy of l= 15 c= 10.0 is 0.615151515151515
The predict accrucy of l= 15 c= 100.0 is 0.6143939393939393
The predict accrucy of l= 16 c= 0.01 is 0.5553825232594736
The predict accrucy of l= 16 c= 0.1 is 0.598680626141168
The predict accrucy of l= 16 c= 1.0 is 0.6072315757170212
The predict accrucy of l= 16 c= 10.0 is 0.605800534267128
The predict accrucy of l= 16 c= 100.0 is 0.6050887904948861
The predict accrucy of l= 17 c= 0.01 is 0.5548096787950888
The predict accrucy of l= 17 c= 0.1 is 0.6022904536374043
The predict accrucy of l= 17 c= 1.0 is 0.6190107966151153
The predict accrucy of l= 17 c= 10.0 is 0.6196819375547126
The predict accrucy of l= 17 c= 100.0 is 0.6210197301968531
The predict accrucy of l= 18 c= 0.01 is 0.5656513689535759
The predict accrucy of l= 18 c= 0.1 is 0.6110934350150445
The predict accrucy of l= 18 c= 1.0 is 0.6281401775110332
The predict accrucy of l= 18 c= 10.0 is 0.6319256915840603
The predict accrucy of l= 18 c= 100.0 is 0.6319236950181537
The predict accrucy of l= 19 c= 0.01 is 0.5520441811012907
The predict accrucy of l= 19 c= 0.1 is 0.5998841741017091
The predict accrucy of l= 19 c= 1.0 is 0.6040740785472217
The predict accrucy of l= 19 c= 10.0 is 0.6058669107931777
The predict accrucy of l= 19 c= 100.0 is 0.6070627281137603
The predict accrucy of l= 20 c= 0.01 is 0.5505681818181818
The predict accrucy of l= 20 c= 0.1 is 0.5948863636363637
The predict accrucy of l= 20 c= 1.0 is 0.6005681818181817
The predict accrucy of l= 20 c= 10.0 is 0.6034090909090908
The predict accrucy of l= 20 c= 100.0 is 0.6039772727272726
```

```
In [69]: np.max(accruacy_value)
```

```
Out[69]: 0.6758685446009389
```

with l=4 c=100, we have best predict accrucy of 67.58%

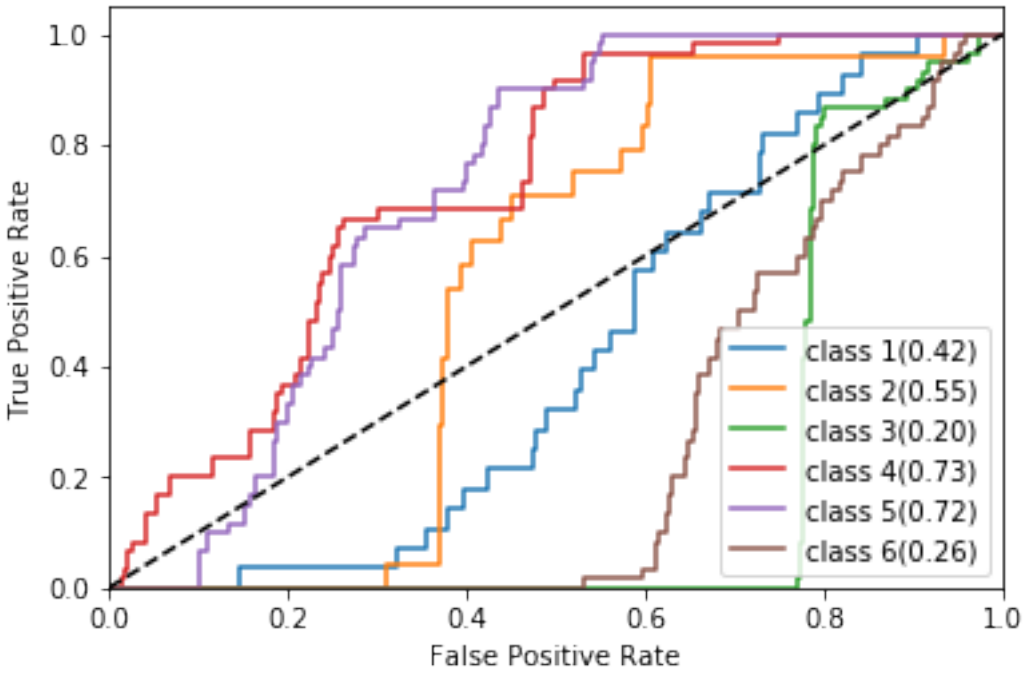
```
In [72]: temp_df=generate_split_l_multraing(4)
         trainSetX=temp_df[['max_rss12', 'min_rss12','mean_rss12','median_rss12','std_rss12','c
         trainSetY=temp_df['class']
         L1_multclassify=LogisticRegression(C=100,penalty='l1',multi_class='multinomial',solver
         train_result=L1_multclassify.fit(trainSetX,trainSetY)
         predict_result=train_result.predict(trainSetX)

         #use c=100 l=4 to generate the best fit model

In [73]: confusion_matrix(predict_result, trainSetY)

Out[73]: array([[10,  0,  0,  2,  0,  0,  0],
                [ 0, 10,  0,  2,  3,  0,  0],
                [ 0,  1, 57,  0,  0,  0,  0],
                [14,  5,  0, 37, 14, 13,  0],
                [ 1,  8,  0, 15, 32,  8,  0],
                [ 3,  0,  0,  4, 11, 39,  0],
                [ 0,  0,  3,  0,  0,  0, 60]], dtype=int64)
```

above is the confusion matrix of mult class
the diagonal means the correct prediction, i.e class 1 is classifyed as 1, class 2 predict as 2
the other element means the wrong prediction, e.g array[4,1] ,means class 4 is classifyed as 1

from the confusion matrix we can know the test error is the sum of number outside the digonal which is 57

```
In [94]: from sklearn.metrics import roc_auc_score
         from sklearn.metrics import roc_curve, auc
         y_scores=L1_multclassify.decision_function(trainSetX)
         fp_rate=dict()
         tp_rate=dict()
         R_Auc=dict()
         for i in range(1,7):
             fp_rate[i], tp_rate[i], _= roc_curve(trainSetY.values, y_scores[:, i],pos_label=i)
             R_Auc[i] = auc(fp_rate[i], tp_rate[i])

In [101]: plt.figure()
          plt.plot(fp_rate[1],tp_rate[1],label='class 1(%0.2f)'%R_Auc[1])
          plt.plot(fp_rate[2],tp_rate[2],label='class 2(%0.2f)'%R_Auc[2])
          plt.plot(fp_rate[3],tp_rate[3],label='class 3(%0.2f)'%R_Auc[3])
          plt.plot(fp_rate[4],tp_rate[4],label='class 4(%0.2f)'%R_Auc[4])
          plt.plot(fp_rate[5],tp_rate[5],label='class 5(%0.2f)'%R_Auc[5])
          plt.plot(fp_rate[6],tp_rate[6],label='class 6(%0.2f)'%R_Auc[6])
          plt.plot([0, 1], [0, 1], 'k--')
          plt.xlim([0.0, 1.0])
          plt.ylim([0.0, 1.05])
          plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.show()
#ROC_plot
```



23 end of f_i

24 f_ii

```
In [39]: from sklearn.naive_bayes import GaussianNB
         from sklearn.naive_bayes import MultinomialNB

In [68]: #********************question f_ii**************************
         accruacy_value=[]
         for l in range(2,21):
             temp_df=generate_split_l_multraing(l)
             trainSetX=temp_df[['max_rss12', 'min_rss12','mean_rss12','median_rss12','std_rss1
             #use all the feature
             trainSetY=temp_df['class']
             GNB_classify=GaussianNB()#Gsussian
             GNB_classify.fit(trainSetX,trainSetY)
             accrucy=GNB_classify.score(trainSetX,trainSetY)
             accruacy_value.append(accrucy)
             print('The predict accrucy of l= '+str(l)+' is '+accrucy.astype('str'))
```

```
The predict accrucy of l= 2 is 0.8125
The predict accrucy of l= 3 is 0.7765151515151515
The predict accrucy of l= 4 is 0.7471590909090909
The predict accrucy of l= 5 is 0.7318181818181818
The predict accrucy of l= 6 is 0.7310606060606061
The predict accrucy of l= 7 is 0.7337662337662337
The predict accrucy of l= 8 is 0.7272727272727273
The predict accrucy of l= 9 is 0.6919191919191919
The predict accrucy of l= 10 is 0.6806818181818182
The predict accrucy of l= 11 is 0.643595041322314
The predict accrucy of l= 12 is 0.6212121212121212
The predict accrucy of l= 13 is 0.6223776223776224
The predict accrucy of l= 14 is 0.6931818181818182
The predict accrucy of l= 15 is 0.6196969696969697
The predict accrucy of l= 16 is 0.6832386363636364
The predict accrucy of l= 17 is 0.6042780748663101
The predict accrucy of l= 18 is 0.5972222222222222
The predict accrucy of l= 19 is 0.666267942583732
The predict accrucy of l= 20 is 0.6568181818181819
```

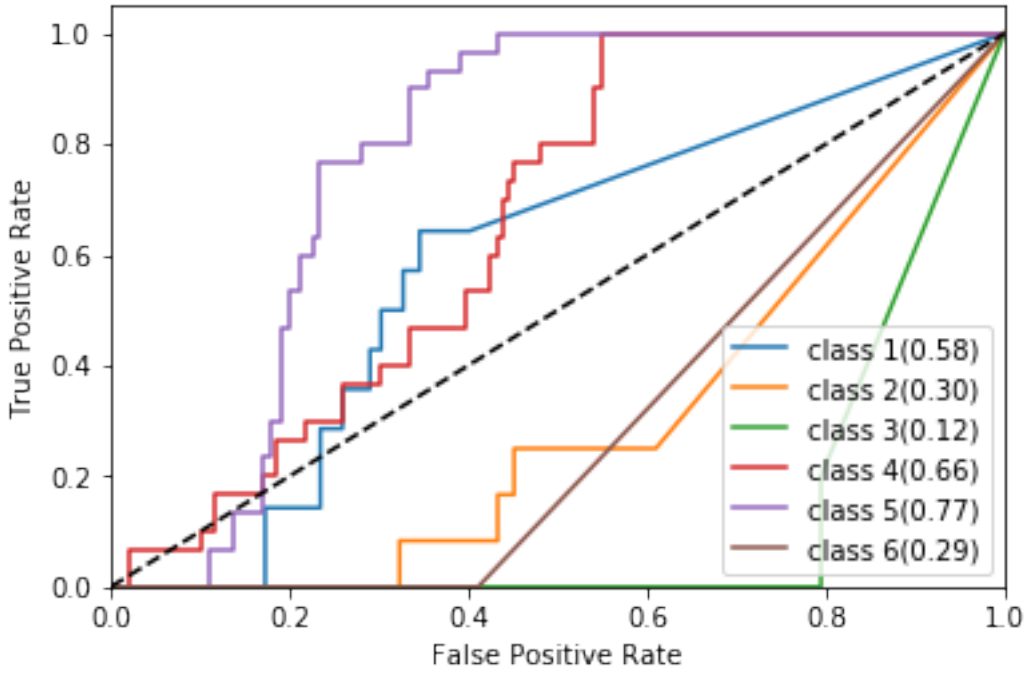
24.0.1 while using Gaussian Naive Bayes' We can get best predit result of 81.25% with l=2

```
In [70]: temp_df=generate_split_l_multraing(2)#use the best result
         trainSetX=temp_df[['max_rss12', 'min_rss12','mean_rss12','median_rss12','std_rss12','
         #use all the feature
         trainSetY=temp_df['class']
         GNB_classify=GaussianNB()
         GNB_classify.fit(trainSetX,trainSetY)
         predict_result=GNB_classify.predict(trainSetX)
         confusion_matrix(predict_result, trainSetY)
         #confusion matrix of Gaussian Naive Bayes classifier

Out[70]: array([[13,  0,  0,  1,  1,  2,  0],
                [ 0, 12,  0,  6,  2,  0,  0],
                [ 0,  0, 30,  0,  0,  0,  0],
                [ 0,  0,  0, 18,  3,  0,  0],
                [ 0,  0,  0,  5, 15,  3,  0],
                [ 1,  0,  0,  0,  9, 25,  0],
                [ 0,  0,  0,  0,  0,  0, 30]], dtype=int64)
```

From the confusion matrix we can know the test error is the sum outside diagonal, which is 28

```
In [74]: y_scores=GNB_classify.predict_proba(trainSetX)
         fp_rate=dict()
         tp_rate=dict()
         R_Auc=dict()
         for i in range(1,7):
             fp_rate[i], tp_rate[i], _= roc_curve(trainSetY.values, y_scores[:, i],pos_label=i
```

```
        R_Auc[i] = auc(fp_rate[i], tp_rate[i])
    plt.figure()
    plt.plot(fp_rate[1],tp_rate[1],label='class 1(%0.2f)'%R_Auc[1])
    plt.plot(fp_rate[2],tp_rate[2],label='class 2(%0.2f)'%R_Auc[2])
    plt.plot(fp_rate[3],tp_rate[3],label='class 3(%0.2f)'%R_Auc[3])
    plt.plot(fp_rate[4],tp_rate[4],label='class 4(%0.2f)'%R_Auc[4])
    plt.plot(fp_rate[5],tp_rate[5],label='class 5(%0.2f)'%R_Auc[5])
    plt.plot(fp_rate[6],tp_rate[6],label='class 6(%0.2f)'%R_Auc[6])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc="lower right")
    plt.show()
    #ROC curve of Gaussian Naive Bayes classifier
```



In [75]:
```
accruacy_value=[]
for l in range(2,21):
    temp_df=generate_split_l_multraing(l)
    trainSetX=temp_df[['max_rss12', 'min_rss12','mean_rss12','median_rss12','std_rss1
    #use all the feature
    trainSetY=temp_df['class']
    MNB_classify=MultinomialNB()#Multinomial
    MNB_classify.fit(trainSetX,trainSetY)
```

```
            accrucy=MNB_classify.score(trainSetX,trainSetY)
            accruacy_value.append(accrucy)
            print('The predict accrucy of l= '+str(l)+' is '+accrucy.astype('str'))
```

```
The predict accrucy of l= 2 is 0.6534090909090909
The predict accrucy of l= 3 is 0.6325757575757576
The predict accrucy of l= 4 is 0.6448863636363636
The predict accrucy of l= 5 is 0.6272727272727273
The predict accrucy of l= 6 is 0.6287878787878788
The predict accrucy of l= 7 is 0.6314935064935064
The predict accrucy of l= 8 is 0.625
The predict accrucy of l= 9 is 0.6136363636363636
The predict accrucy of l= 10 is 0.5977272727272728
The predict accrucy of l= 11 is 0.609504132231405
The predict accrucy of l= 12 is 0.5776515151515151
The predict accrucy of l= 13 is 0.583916083916084
The predict accrucy of l= 14 is 0.588474025974026
The predict accrucy of l= 15 is 0.5856060606060606
The predict accrucy of l= 16 is 0.5703125
The predict accrucy of l= 17 is 0.5828877005347594
The predict accrucy of l= 18 is 0.5820707070707071
The predict accrucy of l= 19 is 0.5669856459330144
The predict accrucy of l= 20 is 0.5596590909090909
```

24.0.2 while using Multi-class Classification Naive Bayes' We can get best predit result of 65.34% with l=2

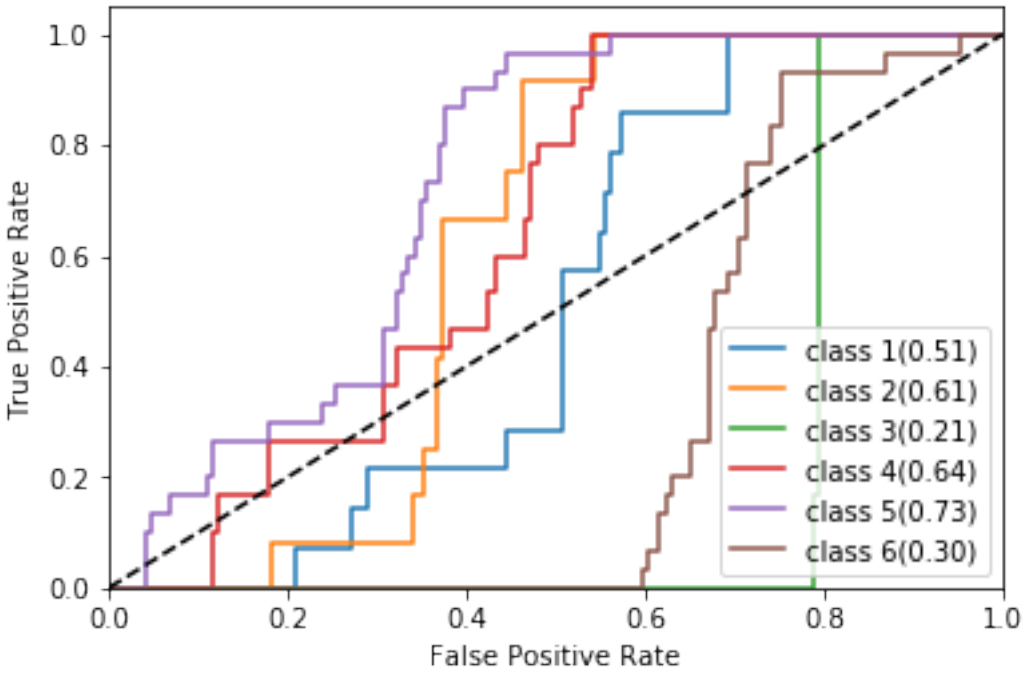
```
In [76]: temp_df=generate_split_l_multraing(2)#use the best result
         trainSetX=temp_df[['max_rss12', 'min_rss12','mean_rss12','median_rss12','std_rss12','
         #use all the feature
         trainSetY=temp_df['class']
         MNB_classify=MultinomialNB()
         MNB_classify.fit(trainSetX,trainSetY)
         predict_result=MNB_classify.predict(trainSetX)
         confusion_matrix(predict_result, trainSetY)
         #confusion matrix of Multinomial Naive Bayes classifier
```

```
Out[76]: array([[ 5,  0,  0,  0,  0,  0,  0],
                [ 0,  7,  0,  1,  2,  0,  0],
                [ 0,  0, 28,  0,  0,  0,  0],
                [ 9,  1,  0, 17, 10,  7,  0],
                [ 0,  4,  0,  3, 11,  6,  0],
                [ 0,  0,  0,  9,  7, 17,  0],
                [ 0,  0,  2,  0,  0,  0, 30]], dtype=int64)
```

From the confusion matrix we can know the test error is the sum outside diagonal, which is 54

```
In [77]: y_scores=MNB_classify.predict_proba(trainSetX)
         fp_rate=dict()
```

```
tp_rate=dict()
R_Auc=dict()
for i in range(1,7):
    fp_rate[i], tp_rate[i], _= roc_curve(trainSetY.values, y_scores[:, i],pos_label=i)
    R_Auc[i] = auc(fp_rate[i], tp_rate[i])
plt.figure()
plt.plot(fp_rate[1],tp_rate[1],label='class 1(%0.2f)'%R_Auc[1])
plt.plot(fp_rate[2],tp_rate[2],label='class 2(%0.2f)'%R_Auc[2])
plt.plot(fp_rate[3],tp_rate[3],label='class 3(%0.2f)'%R_Auc[3])
plt.plot(fp_rate[4],tp_rate[4],label='class 4(%0.2f)'%R_Auc[4])
plt.plot(fp_rate[5],tp_rate[5],label='class 5(%0.2f)'%R_Auc[5])
plt.plot(fp_rate[6],tp_rate[6],label='class 6(%0.2f)'%R_Auc[6])
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.show()
#ROC curve of Multinomial Naive Bayes classifier
```



25 end of f_ii

26 f_iii

While using L1-penalized Multinomial regression,We have best predict accrucy of 67.58%
While using Gaussian Naive Bayes,We have best predict accrucy of 81.25%
While using Multinomial Naive Bayes,We have best predict accrucy of 65.34%

So Gaussian Naive Bayes is better for multi-class classification in this problem

27 end of f_iii