HW34

June 26, 2019

```
In [2]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from pandas import DataFrame, Series
        import seaborn as sns
        from seaborn import scatterplot
        from sklearn import neighbors
        from sklearn import datasets
        from sklearn.metrics import confusion_matrix
        import sklearn.metrics as metr
        from sklearn import preprocessing
        import math
        import os
        # import statsmodels.api as sm
        from sklearn.feature_selection import RFE
        from sklearn.linear_model import LinearRegression
        from sklearn.feature_selection import RFE
        from sklearn.feature_selection import RFECV
        from sklearn.linear_model import LogisticRegression
        from numpy import linalg as la
        from sklearn.model_selection import StratifiedKFold
        from sklearn.feature_selection import RFECV
        from sklearn.feature_selection import chi2
        from sklearn.linear_model import LogisticRegressionCV
        from sklearn.metrics import roc_curve, auc
        from sklearn.naive_bayes import GaussianNB, MultinomialNB
        from sklearn.preprocessing import normalize
        from sklearn.preprocessing import label_binarize
```

```
In [4]: path = 'C:\\Users\\HP\\Desktop\\EE559\\HW34\\AReM'
```

```
In [5]: def create_samples(path):
            filename_whole = []
            for filename in os.listdir(path):
                if ('.pdf' in filename) == False:
                    filename_whole.append(filename)
            filename_whole.sort
            test = []
```

```python
train = []
data = []
for i in range(len(filename_whole)):
    path_1 = path + '\\' + filename_whole[i]
    if i <= 1:
        # print(path_1)
        filename1 = []
        for filename in os.listdir(path_1):
            filename1.append(filename)
        filename1.sort()
        for j in range(0, 2, 1):
            path1_1 = path_1 + '\\' + filename1[j]
            a = pd.read_csv(path1_1, sep=None, engine='python', skiprows=range(0,
            b = a.values
            test.append(b)
            data.append(b)
            # print(path1_1)
        for j in range(2, len(filename1), 1):
            if ((i == 1) & (j == 3)) == True:
                path1_1 = path_1 + '\\' + filename1[j]
                a = pd.read_csv(path1_1, sep=' ', engine='python', skiprows=range(
                a = a.drop('Unnamed: 7', axis=1)
                b = a.values
                train.append(b)
                data.append(b)
                # print(path1_1)
            else:
                path1_1 = path_1 + '\\' + filename1[j]
                a = pd.read_csv(path1_1, sep=None, engine='python', skiprows=range
                b = a.values
                train.append(b)
                data.append(b)
                # print(path1_1)
    else:
        # print(path_1)
        filename1 = []
        for filename in os.listdir(path_1):
            filename1.append(filename)
        filename1.sort()
        for j in range(0, 3, 1):
            path1_1 = path_1 + '\\' + 'dataset' + str(j + 1) + '.csv'
            # print(path1_1)

            a = pd.read_csv(path1_1, sep=None, engine='python', skiprows=range(0,
            b = a.values
            test.append(b)
            data.append(b)
```

```
                for j in range(3, len(filename1), 1):
                    path1_1 = path_1 + '\\' + 'dataset' + str(j + 1) + '.csv'
                    a = pd.read_csv(path1_1, sep=',', engine='python', skiprows=range(0, 4
                    b = a.values
                    train.append(b)
                    data.append(b)
                    # print(path1_1)

        # print(len(test))
        # print(len(train))
        # print(len(data))
        return test, train, data
```

```
In [6]: def create_samples_multi(path):
        filename_whole = []
        for filename in os.listdir(path):
            if ('.pdf' in filename) == False:
                filename_whole.append(filename)
        filename_whole.sort
        test = []
        train = []
        data = []
        test_classes = []
        train_classes = []
        for i in range(len(filename_whole)):
            path_1 = path + '\\' + filename_whole[i]
            if i <= 1:
                # print(path_1)
                filename1 = []
                for filename in os.listdir(path_1):
                    filename1.append(filename)
                filename1.sort()
                for j in range(0, 2, 1):
                    path1_1 = path_1 + '\\' + filename1[j]
                    a = pd.read_csv(path1_1, sep=None, engine='python', skiprows=range(0, 4
                    b = a.values
                    test.append(b)
                    data.append(b)
                    # print(path1_1)
                    test_classes.append(filename_whole[i])
                for j in range(2, len(filename1), 1):
                    if ((i == 1) & (j == 3)) == True:
                        path1_1 = path_1 + '\\' + filename1[j]
                        a = pd.read_csv(path1_1, sep=' ', engine='python', skiprows=range((
                        a = a.drop('Unnamed: 7', axis=1)
                        b = a.values
                        train.append(b)
                        data.append(b)
```

```
                    train_classes.append(filename_whole[i])
                    # print(path1_1)
                else:
                    path1_1 = path_1 + '\\' + filename1[j]
                    a = pd.read_csv(path1_1, sep=None, engine='python', skiprows=range
                    b = a.values
                    train.append(b)
                    data.append(b)
                    train_classes.append(filename_whole[i])
                    # print(path1_1)
        else:
            # print(path_1)
            filename1 = []
            for filename in os.listdir(path_1):
                filename1.append(filename)
            filename1.sort()
            for j in range(0, 3, 1):
                path1_1 = path_1 + '\\' + 'dataset' + str(j + 1) + '.csv'
                # print(path1_1)

                a = pd.read_csv(path1_1, sep=None, engine='python', skiprows=range(0,
                b = a.values
                test.append(b)
                data.append(b)
                test_classes.append(filename_whole[i])

            for j in range(3, len(filename1), 1):
                path1_1 = path_1 + '\\' + 'dataset' + str(j + 1) + '.csv'
                a = pd.read_csv(path1_1, sep=',', engine='python', skiprows=range(0, 4
                b = a.values
                train.append(b)
                data.append(b)
                train_classes.append(filename_whole[i])
                # print(path1_1)
    return test, train, data, train_classes, test_classes
```

```
In [8]: def c_1():
            print('There are a lot of features used in time series')
            print('like correlation structure, distributionentropystationarityscaling properti
        c_1()
```

```
There are a lot of features used in time series
like correlation structure, distributionentropystationarityscaling properties
```

```
In [10]: def feature_extraction(data):
             aa = []
             for i in range(6):
```

```
            for j in range(8):
                if j == 1:
                    aa.append('min_' + str(i + 1))
                elif j == 2:
                    aa.append('max_' + str(i + 1))
                elif j == 3:
                    aa.append('mean_' + str(i + 1))
                elif j == 4:
                    aa.append('median_' + str(i + 1))
                elif j == 5:
                    aa.append('standard deviation_' + str(i + 1))
                elif j == 6:
                    aa.append('1st quart_' + str(i + 1))
                elif j == 7:
                    aa.append('3rd quart_' + str(i + 1))
                else:
                    pass

        result = pd.DataFrame(columns=aa)
        for k in range(88):
            bb = []
            for i in range(6):
                for j in range(7):
                    if j == 0:
                        bb.append(np.min(data[k][:, i + 1]))
                    elif j == 1:
                        bb.append(np.max(data[k][:, i + 1]))
                    elif j == 2:
                        bb.append(np.mean(data[k][:, i + 1]))
                    elif j == 3:
                        bb.append(np.median(data[k][:, i + 1]))
                    elif j == 4:
                        bb.append(np.std(data[k][:, i + 1]))
                    elif j == 5:
                        bb.append(np.percentile(data[k][:, i + 1], 25))
                    elif j == 6:
                        bb.append(np.percentile(data[k][:, i + 1], 75))
                    else:
                        pass
            result.loc[k] = bb
        print(result)
    test_data, train_data, whole_data = create_samples(path)
    feature_extraction(whole_data)
```

```
   min_1  max_1     mean_1  median_1  standard deviation_1  1st quart_1  \
0  37.25  45.00  40.624792    40.500              1.475428      39.2500
1  38.00  45.67  42.812812    42.500              1.434054      42.0000
2  35.00  47.40  43.954500    44.330              1.557210      43.0000
```

3	33.00	47.75	42.179812	43.500	3.666840	39.1500
4	33.00	45.75	41.678063	41.750	2.241152	41.3300
5	37.00	48.00	43.454958	43.250	1.384653	42.5000
6	36.25	48.00	43.969125	44.500	1.616677	43.3100
7	12.75	51.00	24.562958	24.250	3.733619	23.1875
8	0.00	42.75	27.464604	28.000	3.579847	25.5000
9	21.00	50.00	32.586208	33.000	6.231642	26.1875
10	27.50	33.00	29.876472	30.000	1.147607	29.0000
11	19.00	45.50	30.938104	29.000	7.676137	26.7500
12	25.00	47.50	31.058250	29.710	4.824761	27.5000
13	24.25	45.00	37.177042	36.250	3.577569	34.5000
14	28.75	44.75	37.561187	36.875	3.223144	35.2500
15	22.00	44.67	37.058708	36.000	3.706313	34.5000
16	19.00	44.00	36.228396	36.000	3.524939	34.0000
17	26.50	44.33	36.687292	36.000	3.525726	34.2500
18	25.33	45.00	37.114313	36.250	3.706518	34.5000
19	26.75	44.75	36.863375	36.330	3.552081	34.5000
20	26.25	44.25	36.957458	36.290	3.431283	34.5000
21	27.75	44.67	37.144833	36.330	3.754986	34.0000
22	27.00	45.00	36.819521	36.000	3.896394	33.7500
23	27.00	44.33	36.541667	36.000	4.014734	33.2500
24	18.50	44.25	35.752354	36.000	4.609992	33.0000
25	19.00	43.75	35.879875	36.000	4.610068	33.0000
26	23.33	43.50	36.244083	36.750	3.818032	33.4575
27	24.25	45.00	37.177042	36.250	3.577569	34.5000
28	23.50	30.00	27.716375	27.500	1.440750	27.0000
29	24.75	48.33	44.182937	48.000	7.487803	48.0000
..	...	...	...	...	...	...
58	33.33	48.00	44.334729	45.000	2.474358	42.2500
59	35.50	46.25	43.174938	43.670	1.986979	42.5000
60	32.75	47.00	42.760562	44.500	3.395376	41.3300
61	30.00	46.67	42.648521	42.750	2.392842	41.5000
62	36.00	47.50	43.720021	45.000	2.381620	43.0000
63	34.50	47.75	44.471146	45.000	1.770706	45.0000
64	35.50	48.00	46.224938	46.000	1.746493	45.2500
65	29.75	48.00	46.932208	47.500	1.830755	47.2375
66	36.33	47.67	45.399625	45.500	1.326737	45.0000
67	36.00	45.80	42.419917	42.670	2.517503	41.3300
68	37.00	48.25	42.516958	42.500	2.193462	41.0000
69	36.25	45.50	42.959354	42.670	1.499314	42.0000
70	36.00	47.33	42.674583	43.670	2.381685	40.0000
71	36.25	45.75	43.187521	44.750	2.488565	39.7500
72	36.00	47.33	44.441187	45.000	2.415277	44.6275
73	19.33	43.50	34.227771	35.500	4.884480	30.5000
74	12.50	45.00	33.509729	34.125	4.845868	30.5000
75	15.00	46.75	34.660583	35.000	5.309571	31.0000
76	18.00	46.00	35.193333	36.000	4.746916	32.0000
77	20.75	46.25	34.763333	35.290	4.737266	31.6700

78	21.50	51.00	34.935812	35.500		4.641102	32.0000
79	18.33	47.67	34.333042	34.750		4.943612	31.2500
80	18.33	45.75	34.599875	35.125		4.726858	31.5000
81	15.50	43.67	34.225875	34.750		4.437168	31.2500
82	21.50	51.25	34.253521	35.000		4.935592	30.9375
83	19.50	45.33	33.586875	34.250		4.646088	30.2500
84	19.75	45.50	34.322750	35.250		4.747524	31.0000
85	19.50	46.00	34.546229	35.250		4.837247	31.2500
86	23.50	46.25	34.873229	35.250		4.526997	31.7500
87	19.25	44.00	34.473188	35.000		4.791706	31.2500

	3rd quart_1	min_2	max_2	mean_2	...	standard deviation_5	\
0	42.0000	0.0	1.30	0.358604	...	2.186168	
1	43.6700	0.0	1.22	0.372437	...	1.993175	
2	45.0000	0.0	1.70	0.426250	...	1.997520	
3	45.0000	0.0	3.00	0.696042	...	3.845436	
4	42.7500	0.0	2.83	0.535979	...	2.408514	
5	45.0000	0.0	1.58	0.378083	...	2.486268	
6	44.6700	0.0	1.50	0.413125	...	3.314843	
7	26.5000	0.0	6.87	0.590833	...	3.689936	
8	30.0000	0.0	7.76	0.449708	...	5.048375	
9	34.5000	0.0	9.90	0.516125	...	5.027179	
10	30.2500	0.0	1.00	0.255929	...	1.745504	
11	38.0000	0.0	6.40	0.467167	...	5.839819	
12	31.8125	0.0	6.38	0.405458	...	7.845242	
13	40.2500	0.0	8.58	2.374208	...	2.887335	
14	40.2500	0.0	9.91	2.080687	...	2.724534	
15	40.0625	0.0	14.17	2.438146	...	3.533457	
16	39.0000	0.0	12.28	2.831687	...	3.163354	
17	39.3725	0.0	12.89	2.973042	...	2.975134	
18	40.2500	0.0	10.84	2.730000	...	2.844908	
19	39.7500	0.0	11.68	2.757312	...	2.653138	
20	40.2500	0.0	8.64	2.420083	...	2.848701	
21	40.5000	0.0	10.76	2.419062	...	2.686488	
22	40.2500	0.0	10.47	2.600146	...	2.778132	
23	39.8125	0.0	10.43	2.847958	...	3.084922	
24	39.3300	0.0	12.60	3.328104	...	3.116805	
25	39.5000	0.0	11.20	3.414312	...	3.533948	
26	39.2500	0.0	9.71	2.736021	...	3.613931	
27	40.2500	0.0	8.58	2.374208	...	2.887335	
28	29.0000	0.0	1.79	0.363687	...	4.070265	
29	48.0000	0.0	3.11	0.101875	...	3.271126	
..	...	...	...	...	...	...	
58	46.5000	0.0	3.90	0.432958	...	5.396165	
59	44.5000	0.0	2.12	0.506583	...	2.980866	
60	45.3725	0.0	3.34	0.486167	...	4.292096	
61	45.0000	0.0	2.95	0.402833	...	3.138405	
62	45.0000	0.0	1.92	0.366708	...	3.285710	

63	45.2500	0.0	2.18	0.290479	...	2.609667
64	48.0000	0.0	4.50	0.312354	...	2.928525
65	47.7500	0.0	4.60	0.429667	...	3.131555
66	46.3300	0.0	1.66	0.460146	...	3.370579
67	44.6175	0.0	2.12	0.460562	...	3.718195
68	44.5000	0.0	2.12	0.440687	...	3.619780
69	44.3300	0.0	2.60	0.352875	...	2.699788
70	44.7500	0.0	2.17	0.419167	...	3.258218
71	45.0000	0.0	2.83	0.271271	...	3.562322
72	45.7500	0.0	4.50	0.346604	...	3.410896
73	37.7500	0.0	14.50	3.995729	...	3.088871
74	36.7500	0.0	13.05	4.450771	...	3.130299
75	38.2500	0.0	13.44	4.200896	...	3.151727
76	38.7500	0.0	16.20	4.321021	...	3.204299
77	38.2500	0.0	12.68	4.223792	...	3.171372
78	38.0625	0.0	12.21	4.115750	...	3.188731
79	38.0000	0.0	12.48	4.396958	...	2.997366
80	38.0000	0.0	15.37	4.398833	...	2.902659
81	37.2500	0.0	17.24	4.354500	...	2.989801
82	37.7500	0.0	13.55	4.457896	...	3.113379
83	37.0000	0.0	14.67	4.576562	...	3.280561
84	38.0000	0.0	13.47	4.456333	...	3.116605
85	37.8125	0.0	12.47	4.371958	...	2.820182
86	38.2500	0.0	14.82	4.380583	...	3.127813
87	38.0000	0.0	13.86	4.359312	...	3.153030

	1st quart_5	3rd quart_5	min_6	max_6	mean_6	median_6	\
0	33.0000	36.0000	0.00	1.92	0.570583	0.430	
1	32.0000	34.5000	0.00	3.11	0.571083	0.430	
2	35.3625	36.5000	0.00	1.79	0.493292	0.430	
3	30.4575	36.3300	0.00	2.18	0.613521	0.500	
4	28.4575	31.2500	0.00	1.79	0.383292	0.430	
5	22.2500	24.0000	0.00	5.26	0.679646	0.500	
6	20.5000	23.7500	0.00	2.96	0.555312	0.490	
7	20.5000	27.0000	0.00	4.97	0.700188	0.500	
8	15.0000	20.7500	0.00	6.76	1.122125	0.830	
9	17.6700	23.5000	0.00	13.61	1.162042	0.830	
10	17.0000	19.0000	0.00	6.40	0.701002	0.710	
11	15.0000	20.8125	0.00	6.73	1.107354	0.830	
12	9.0000	18.3125	0.00	4.92	1.098104	0.940	
13	17.9500	21.7500	0.00	9.34	2.921729	2.500	
14	18.0000	21.5000	0.00	9.62	2.765896	2.450	
15	16.0000	21.0000	0.00	8.55	2.983750	2.570	
16	14.0000	18.0625	0.00	9.98	3.480688	3.340	
17	14.6700	18.5000	0.00	8.19	3.073313	2.690	
18	14.7500	18.5000	0.00	9.50	3.076354	2.770	
19	15.0000	18.6700	0.00	8.81	2.773313	2.590	
20	14.0000	18.2500	0.00	8.34	2.934625	2.525	

21	15.0000	18.7500	0.00	8.75	2.822437	2.590
22	15.5000	19.2700	0.00	8.99	2.887563	2.525
23	15.0000	19.5000	0.00	9.18	3.225458	2.870
24	14.0000	18.0625	0.00	9.39	3.069667	2.770
25	14.7500	19.6900	0.00	8.50	3.093021	2.930
26	15.7500	21.0000	0.00	11.15	3.530500	3.110
27	17.9500	21.7500	0.00	9.34	2.921729	2.500
28	5.5000	10.7500	0.00	4.50	0.734271	0.710
29	2.0000	5.5425	0.00	3.91	0.692771	0.500
..	...	...	...	...	...	...
58	9.3300	17.7500	0.00	5.02	0.933000	0.830
59	12.7500	16.5000	0.00	5.72	0.911979	0.830
60	13.0000	18.5650	0.00	5.73	0.842271	0.710
61	10.6275	14.2500	0.00	4.64	0.917354	0.830
62	11.3100	15.5425	0.00	6.18	1.039687	0.830
63	12.0000	14.8125	0.00	4.32	0.927375	0.830
64	12.0000	15.2500	0.00	6.00	0.882583	0.830
65	11.6700	15.5000	0.00	6.58	0.991125	0.830
66	11.2500	14.5000	0.00	4.50	0.795104	0.820
67	7.6275	12.0000	0.00	6.65	1.226271	1.090
68	12.6275	17.5000	0.00	6.85	0.977417	0.830
69	14.0000	16.6900	0.00	4.00	0.748479	0.820
70	12.7500	16.5000	0.00	3.77	0.702042	0.500
71	16.5000	21.0000	0.00	3.83	0.645458	0.500
72	11.0000	14.6700	0.00	5.91	1.155083	0.940
73	14.7500	18.6700	0.00	9.74	3.394125	3.100
74	14.6275	18.7500	0.00	8.96	3.378479	3.085
75	14.2500	18.5000	0.00	8.99	3.244396	3.000
76	14.2500	18.5000	0.00	8.50	3.241958	3.015
77	14.2500	18.3300	0.00	9.39	3.288271	3.270
78	14.2375	18.2500	0.00	10.21	3.280021	3.015
79	13.7500	18.0000	0.00	8.01	3.261583	2.980
80	14.0000	18.2500	0.00	8.86	3.289542	3.015
81	14.3300	18.2500	0.00	9.42	3.479542	3.270
82	13.7500	18.0000	0.00	8.32	3.500750	3.285
83	13.7300	18.2500	0.00	8.32	3.259729	3.110
84	13.5000	17.7500	0.00	9.67	3.432563	3.200
85	14.0000	17.7500	0.00	10.00	3.338125	3.080
86	13.7500	18.0000	0.00	9.51	3.424646	3.270
87	13.7300	17.7500	0.43	9.00	3.340458	3.090

	standard deviation_6	1st quart_6	3rd quart_6
0	0.582308	0.0000	1.3000
1	0.600383	0.0000	1.3000
2	0.512971	0.0000	0.9400
3	0.523771	0.0000	1.0000
4	0.388759	0.0000	0.5000
5	0.621885	0.4300	0.8700

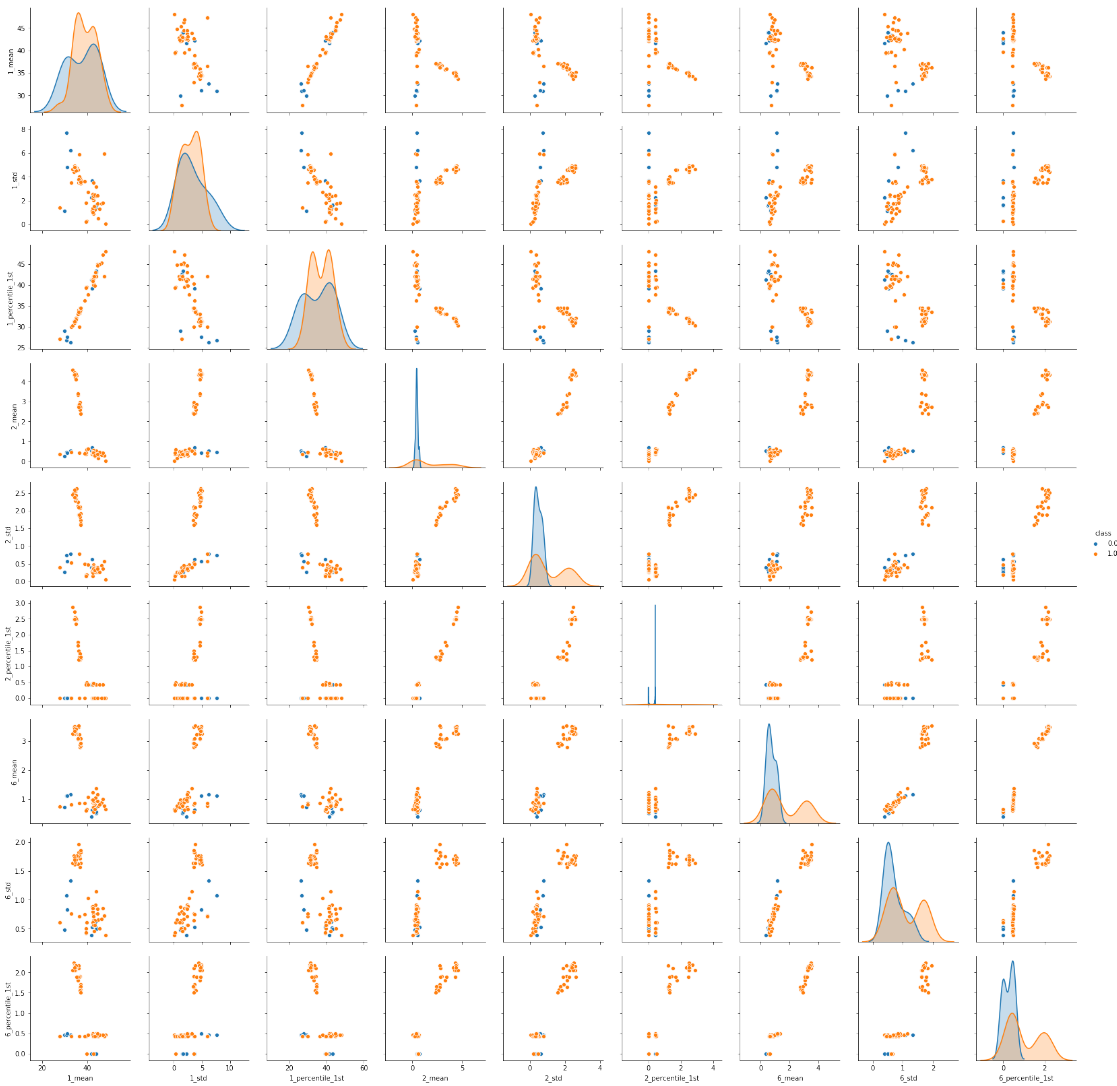
6	0.487318	0.0000	0.8300
7	0.692997	0.4300	0.8700
8	1.011287	0.4700	1.3000
9	1.331591	0.4700	1.3000
10	0.480909	0.4700	0.9400
11	1.079715	0.4700	1.3000
12	0.830614	0.5000	1.3000
13	1.850669	1.5000	3.9000
14	1.767359	1.4100	3.7700
15	1.813837	1.5000	4.1500
16	1.825864	2.1025	4.5500
17	1.627976	1.9125	4.0875
18	1.822633	1.7000	4.0375
19	1.568283	1.6400	3.6325
20	1.629680	1.6600	4.0300
21	1.635476	1.5800	3.7400
22	1.721298	1.5600	3.7700
23	1.767913	1.8850	4.2625
24	1.746503	1.7975	4.0600
25	1.624339	1.8900	4.0600
26	1.961639	2.1700	4.6175
27	1.850669	1.5000	3.9000
28	0.613049	0.4300	1.0000
29	0.675076	0.3225	0.9400
..	...	...	...
58	0.672907	0.4700	1.2500
59	0.665467	0.4700	1.2200
60	0.721413	0.4300	1.0900
61	0.708898	0.4700	1.1200
62	0.915702	0.4700	1.2200
63	0.755647	0.4700	1.2200
64	0.667727	0.4700	1.1200
65	0.854438	0.4700	1.2200
66	0.502483	0.4700	1.0000
67	0.891058	0.5000	1.5850
68	0.852390	0.4700	1.2200
69	0.460671	0.4300	0.9500
70	0.566859	0.4300	0.9400
71	0.566828	0.4300	0.8300
72	0.841210	0.5000	1.5000
73	1.790222	2.1050	4.4250
74	1.785497	2.0600	4.4400
75	1.629283	2.1200	4.2400
76	1.767339	1.8850	4.4400
77	1.645811	2.0500	4.3050
78	1.699145	2.1200	4.5000
79	1.615604	2.0500	4.3200
80	1.678418	2.1200	4.2600

```
81               1.759311       2.2400       4.5375
82               1.690614       2.1800       4.5575
83               1.638534       2.0500       4.3225
84               1.730921       2.1575       4.5650
85               1.655016       2.1600       4.3350
86               1.689198       2.1700       4.5000
87               1.697343       2.1200       4.3750

[88 rows x 42 columns]
```

```
In [11]: def d_1(train):
             data_list = []
             column_index = []
             for i in [1, 2, 6]:
                 mean_1 = []
                 std_1 = []
                 percentile_1st_1 = []
                 class_1 = []
                 for j in range(9):
                     mean = np.mean(train[j][:, i])
                     std = np.std(train[j][:, i])
                     percentile_1st = np.percentile(train[j][:, i], 25)
                     mean_1.append(mean)
                     std_1.append(std)
                     percentile_1st_1.append(percentile_1st)
                     class_1.append(0)
                 for j in range(9, 69, 1):
                     mean = np.mean(train[j][:, i])
                     std = np.std(train[j][:, i])
                     percentile_1st = np.percentile(train[j][:, i], 25)
                     mean_1.append(mean)
                     std_1.append(std)
                     percentile_1st_1.append(percentile_1st)
                     class_1.append(1)
                 data_list.append(mean_1)
                 data_list.append(std_1)
                 data_list.append(percentile_1st_1)
                 column_index.append(str(i) + '_mean')
                 column_index.append(str(i) + '_std')
                 column_index.append(str(i) + '_percentile_1st')
             data_list.append(class_1)
             column_index.append('class')
             data_result = np.array(data_list)
             print(data_result.shape)
             data_df = DataFrame(data_result.T, columns=column_index)
             sns.pairplot(data_df, hue='class',
                          vars=['1_mean', '1_std', '1_percentile_1st', '2_mean', '2_std', '2_p
```

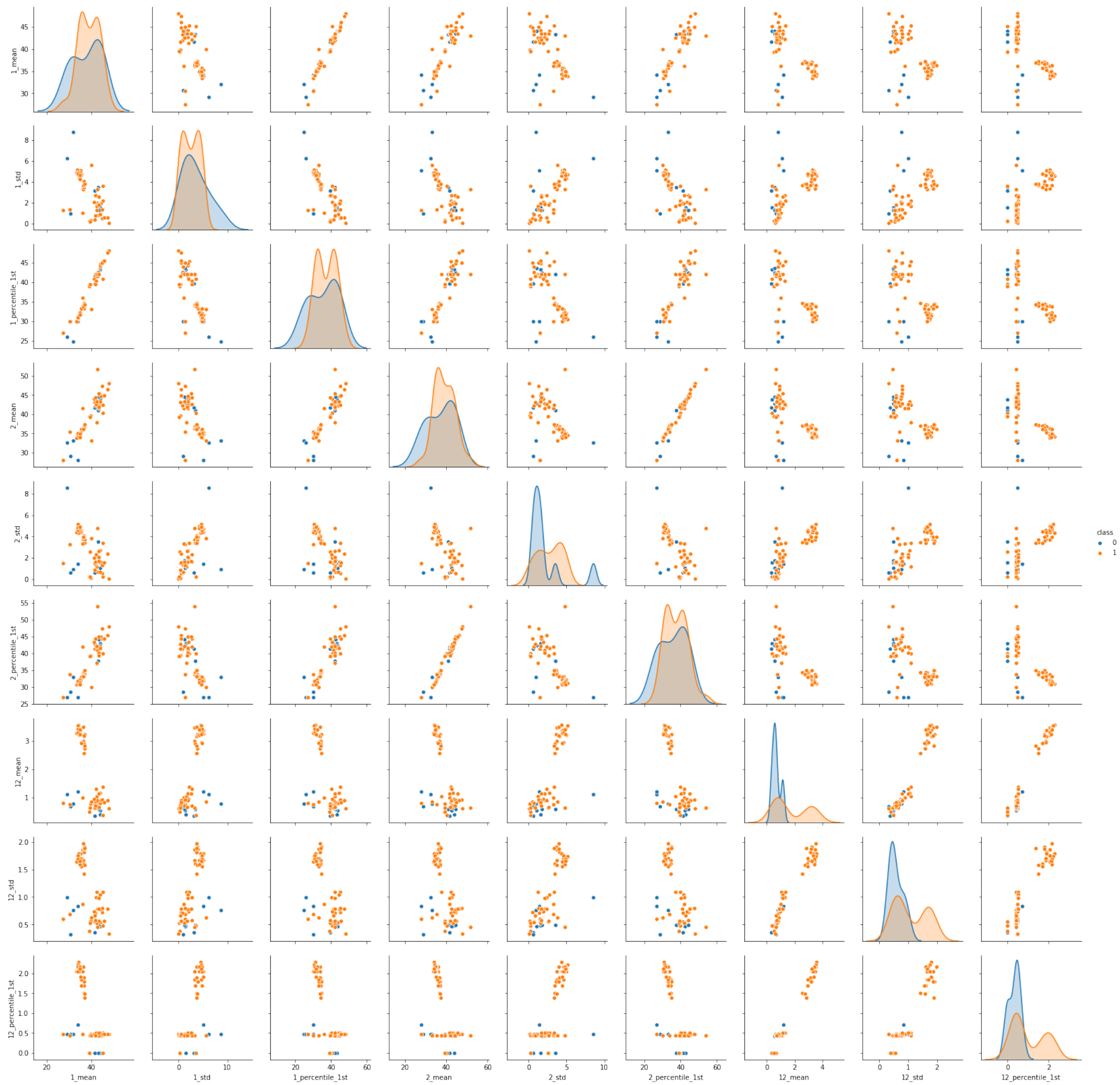
```
                          '6_percentile_1st'])

        d_1(train_data)
```

```
(10, 69)
```




```
In [12]: def d_2_1(train):
             piece = 240
             data_list = []
             column_index = []
             result = {}
             for i in [1, 6]:
                 for k in range(2):
```

```
            mean_1 = []
            std_1 = []
            percentile_1st_1 = []
            class_1 = []

            for j in range(0, 69, 1):
                if j >= 9:
                    flag = 1
                else:
                    flag = 0
                if (((i == 6) & (k == 1)) == True):
                    mean = np.mean(train[j][k * piece:(k + 1) * piece, i])
                    std = np.std(train[j][k * piece:(k + 1) * piece, i])
                    percentile_1st = np.percentile(train[j][k * piece:(k + 1) * piece
                    mean_1.append(mean)
                    std_1.append(std)
                    percentile_1st_1.append(percentile_1st)
                    class_1.append(flag)
                elif i == 1:
                    mean = np.mean(train[j][k * piece:(k + 1) * piece, i])
                    std = np.std(train[j][k * piece:(k + 1) * piece, i])
                    percentile_1st = np.percentile(train[j][k * piece:(k + 1) * piece
                    mean_1.append(mean)
                    std_1.append(std)
                    percentile_1st_1.append(percentile_1st)
                    class_1.append(flag)
                else:
                    pass
            if (((i == 6) & (k == 1)) == True) | (i == 1):
                data_list.append(mean_1)
                data_list.append(std_1)
                data_list.append(percentile_1st_1)
            else:
                pass

    data_list.append(class_1)
    for i in [1, 2, 12]:
        column_index.append(str(i) + '_mean')
        column_index.append(str(i) + '_std')
        column_index.append(str(i) + '_percentile_1st')
    column_index.append('class')
    for i in range(10):
        result[column_index[i]] = data_list[i]
    data_df = DataFrame(result)
    sns.pairplot(data_df, hue='class',
                 vars=['1_mean', '1_std', '1_percentile_1st', '2_mean', '2_std', '2_p
                       '12_std', '12_percentile_1st'])
d_2_1(train_data)
```



d_2: Acutally, I think the second diagram shows more sparse data, especially in the margin.

```
In [16]: def d_3_1(train, num, sep):
             piece = round(480 / num)
             results = {}
             if sep == 0:
                 sep_1 = 9
             else:
                 sep_1 = 4
             for i in [1, 2, 3, 4, 5, 6]:
                 for k in range(num):
                     mean_1 = []
                     std_1 = []
                     percentile_1st_1 = []
                     class_1 = []
```

```
            for j in range(len(train)):  # 69 or...
                if j >= sep_1:
                    flag = 0
                else:
                    flag = 1
                if k == (num - 1):
                    num_1 = len(train[j][:, 1])
                    mean = np.mean(train[j][k * piece:num_1, i])
                    std = np.std(train[j][k * piece:num_1, i])
                    percentile_1st = np.max(train[j][k * piece:num_1, i])

                    # percentile_1st = np.percentile(train[j][k * piece:num_1, i], 75
                    mean_1.append(mean)
                    std_1.append(std)
                    percentile_1st_1.append(percentile_1st)
                    class_1.append(flag)
                else:
                    mean = np.mean(train[j][k * piece:(k + 1) * piece, i])
                    std = np.std(train[j][k * piece:(k + 1) * piece, i])
                    percentile_1st = np.max(train[j][k * piece:(k + 1) * piece, i])

                    # percentile_1st = np.percentile(train[j][k * piece:(k + 1) * pie
                    mean_1.append(mean)
                    std_1.append(std)
                    percentile_1st_1.append(percentile_1st)
                    class_1.append(flag)
            results[str(i) + '_' + str(k + 1) + '_mean'] = mean_1
            results[str(i) + '_' + str(k + 1) + '_std'] = std_1
            results[str(i) + '_' + str(k + 1) + '_percentile_3st'] = percentile_1st_1
    results['class'] = class_1
    return results


def test_LogisticRegression(X_train, X_test, y_train, y_test):
    cls = LogisticRegression()
    results = cls.fit(X_train, y_train)
    scores = cls.score(X_test, y_test)
    return scores


def features_backward_selection(x_train, y_train, x_test, y_test):
    selected_features = []
    scores_result = []
    estimator = LogisticRegression()
    for i in range(1, len(x_train[:][0])):
        scores = []
        delete_column_1 = []
        selector = RFE(estimator, n_features_to_select=i)
```

```
        selector = selector.fit(x_train, y_train)
        delete_column = (selector.support_)
        for j in range(len(delete_column)):
            if delete_column[j] == True:
                delete_column_1.append(j)
            else:
                pass
        x_train_new = np.delete(x_train, delete_column_1, axis=1)
        x_test_new = np.delete(x_test, delete_column_1, axis=1)
        sfolder = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)
        for train, test in sfolder.split(x_train_new, y_train):
            # x_train, y_train, x_test, y_test
            x_train_1 = np.delete(x_train_new, test, axis=0)
            x_test_1 = np.delete(x_train_new, train, axis=0)
            y_train_1 = np.delete(y_train, test, axis=0)
            y_test_1 = np.delete(y_train, train, axis=0)
            scores.append(test_LogisticRegression(x_train_1, x_test_1, y_train_1, y_te
        scores_1 = np.mean(np.array(scores))
        scores_result.append(scores_1)
#     print(scores_result)
    bbb = np.argmax(np.array(scores_result))
    aaa = (np.array(scores_result))
    result_1 = len(x_train[:][0]) - (bbb + 1)
    result_2 = np.max(aaa)
    print('number of features selected is',result_1)
    print('best score is ',result_2)
    return result_1, result_2, x_train_new, x_test_new


def find_best():
    accuracy = []
    features = []
    x_train_new_1 = []
    x_test_new_1 = []
    l = range(1, 21, 1)
    test_data, train_data, whole_data = create_samples(path)
    for i in range(1, 21, 1):
        train = d_3_1(train_data, i, 0)
        train_2 = (DataFrame(train)).values
        x_train = np.delete(train_2, train_2.shape[1] - 1, axis=1)
        y_train = train_2[:, train_2.shape[1] - 1]

        test = d_3_1(test_data, i, 1)
        test_2 = (DataFrame(test)).values
        x_test = np.delete(test_2, test_2.shape[1] - 1, axis=1)
        y_test = test_2[:, test_2.shape[1] - 1]
        print('train shape is ',train_2.shape)
```

```
            result_1, result_2, x_train_new, x_test_new = features_backward_selection(x_t
            accuracy.append(result_2)
            features.append(result_1)
            x_train_new_1.append(x_train_new)
            x_test_new_1.append(x_test_new)
        accuracy_index = (int)(np.argmax(np.array(accuracy)))
        best = (l[accuracy_index], features[accuracy_index])
        print(best)
        print('The best test score is %f' % (
            test_LogisticRegression(x_train_new_1[accuracy_index], x_test_new_1[accuracy_
        print('the wrong way is that just assuming the number of predictors we should use
              'and then use cross validation to select the best l in this case.\n'
              'the right way is that using cross validation in both step 1 and 2, that is
              'using CV to find the best number of predicators, and then usc CV to find \n
              'the best number of l')
        return l[accuracy_index], features[accuracy_index]

    l,num = find_best()
```

```
train shape is  (69, 19)
number of features selected is 17
best score is  0.9285714285714286
train shape is  (69, 37)
number of features selected is 34
best score is  0.9571428571428571
train shape is  (69, 55)
number of features selected is 53
best score is  0.9417582417582417
train shape is  (69, 73)
number of features selected is 70
best score is  0.9714285714285715
train shape is  (69, 91)
number of features selected is 89
best score is  0.9714285714285715
train shape is  (69, 109)
number of features selected is 107
best score is  0.9428571428571428
train shape is  (69, 127)
number of features selected is 125
best score is  0.9714285714285715
train shape is  (69, 145)
number of features selected is 143
best score is  0.9571428571428571
train shape is  (69, 163)
number of features selected is 161
best score is  0.9571428571428571
train shape is  (69, 181)
number of features selected is 179
```

```
best score is  0.9571428571428571
train shape is  (69, 199)
number of features selected is 197
best score is  0.9571428571428571
train shape is  (69, 217)
number of features selected is 215
best score is  0.9571428571428571
train shape is  (69, 235)
number of features selected is 233
best score is  0.9571428571428571
train shape is  (69, 253)
number of features selected is 251
best score is  0.9571428571428571
train shape is  (69, 271)
number of features selected is 269
best score is  0.9571428571428571
train shape is  (69, 289)
number of features selected is 287
best score is  0.9571428571428571
train shape is  (69, 307)
number of features selected is 305
best score is  0.9571428571428571
train shape is  (69, 325)
number of features selected is 323
best score is  0.9571428571428571
train shape is  (69, 343)
number of features selected is 341
best score is  0.9571428571428571
train shape is  (69, 361)
number of features selected is 359
best score is  0.9571428571428571
(4, 70)
The best test score is 0.789474
the wrong way is that just assuming the number of predictors we should use,
and then use cross validation to select the best l in this case.
the right way is that using cross validation in both step 1 and 2, that is
using CV to find the best number of predicators, and then usc CV to find
the best number of l
```

```
In [19]: def dram_diagram(l):
             test_data, train_data, whole_data = create_samples(path)
             train = d_3_1(train_data, l, 0)
             train_2 = (DataFrame(train)).values
             x_train = np.delete(train_2, train_2.shape[1] - 1, axis=1)
             y_train = train_2[:, train_2.shape[1] - 1]

             test = d_3_1(test_data, l, 1)
```

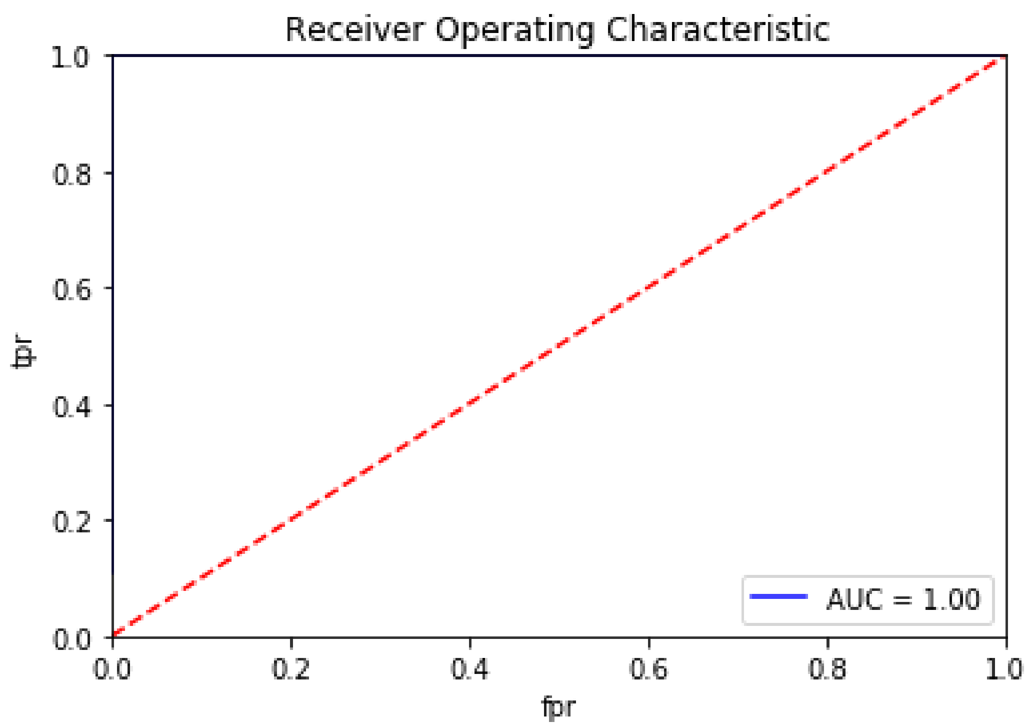
```
            test_2 = (DataFrame(test)).values
            x_test = np.delete(test_2, test_2.shape[1] - 1, axis=1)
            y_test = test_2[:, test_2.shape[1] - 1]
            print(train_2.shape)
            result_1, result_2, x_train_new, x_test_new = features_backward_selection(x_train
            print(result_1, result_2)

            classifier = LogisticRegression()
            classifier.fit(x_train, y_train)
            predic_train = classifier.predict(x_train)
            predic_test = classifier.predict(x_test)
            vc_matrix1 = confusion_matrix(y_train, predic_train)
            vc_matrix2 = confusion_matrix(y_test, predic_test)

            predictions = classifier.predict_proba(x_train)  # .auc
            print(predictions)
            false_positive_rate, recall, thresholds = roc_curve(y_train, predictions[:, 1])
            roc_auc = auc(false_positive_rate, recall)
            plt.title('Receiver Operating Characteristic')
            plt.plot(false_positive_rate, recall, 'b', label='AUC = %0.2f' % roc_auc)
            plt.legend(loc='lower right')
            plt.plot([0, 1], [0, 1], 'r--')
            plt.xlim([0.0, 1.0])
            plt.ylim([0.0, 1.0])
            plt.ylabel('tpr')
            plt.xlabel('fpr')
            scores, pvalues = chi2(x_train, y_train)
            print('confusion matrix for train',vc_matrix1)
            print('confusion matrix for test',vc_matrix2)
            print('coefficient is ',classifier.coef_)
            print('interception is ',classifier.intercept_)
            print('p values is ', pvalues)
            test_score = classifier.score(x_test, y_test)
            print('The test score is',test_score)
            plt.show()
        dram_diagram(l)
```

```
(69, 73)
number of features selected is 70
best score is  0.9714285714285715
70 0.9714285714285715
[[7.44760624e-07 9.99999255e-01]
 [2.79879908e-06 9.99997201e-01]
 [1.39466596e-05 9.99986053e-01]
 [1.67392865e-02 9.83260714e-01]
 [3.76231412e-02 9.62376859e-01]
 [1.49449668e-03 9.98505503e-01]
 [4.56539797e-03 9.95434602e-01]
```

```
[1.18298578e-02 9.88170142e-01]
[3.28001482e-03 9.96719985e-01]
[9.99968354e-01 3.16462400e-05]
[9.99980173e-01 1.98274548e-05]
[9.99966565e-01 3.34347282e-05]
[9.99989151e-01 1.08494545e-05]
[9.99948159e-01 5.18409100e-05]
[9.99935660e-01 6.43396284e-05]
[9.99831138e-01 1.68861679e-04]
[9.99763066e-01 2.36933676e-04]
[9.99934161e-01 6.58394335e-05]
[9.99714372e-01 2.85628057e-04]
[9.95991266e-01 4.00873447e-03]
[9.98749234e-01 1.25076631e-03]
[9.99999998e-01 2.32438010e-09]
[9.99999967e-01 3.30587647e-08]
[9.99982535e-01 1.74649941e-05]
[9.99985649e-01 1.43512550e-05]
[9.90069008e-01 9.93099217e-03]
[9.99999051e-01 9.48748831e-07]
[9.98542429e-01 1.45757105e-03]
[1.00000000e+00 3.77122191e-10]
[9.99999967e-01 3.30587647e-08]
[9.99985649e-01 1.43512550e-05]
[9.99999051e-01 9.48748831e-07]
[9.98541858e-01 1.45814161e-03]
[9.98198578e-01 1.80142242e-03]
[9.87493719e-01 1.25062809e-02]
[9.68184966e-01 3.18150338e-02]
[9.93508021e-01 6.49197861e-03]
[9.97734878e-01 2.26512242e-03]
[9.99357343e-01 6.42656743e-04]
[9.98432927e-01 1.56707349e-03]
[9.99581237e-01 4.18762637e-04]
[9.99982484e-01 1.75156594e-05]
[9.97271075e-01 2.72892519e-03]
[9.99745344e-01 2.54655595e-04]
[9.98000879e-01 1.99912138e-03]
[9.99998874e-01 1.12601813e-06]
[9.99985079e-01 1.49206073e-05]
[9.99996817e-01 3.18266873e-06]
[9.99999732e-01 2.68162227e-07]
[9.99996222e-01 3.77780270e-06]
[9.99999560e-01 4.40016843e-07]
[9.99983656e-01 1.63444351e-05]
[9.99966816e-01 3.31843254e-05]
[9.99618797e-01 3.81202908e-04]
[9.99924371e-01 7.56290328e-05]
```

```
 [9.97414859e-01 2.58514060e-03]
 [9.99953844e-01 4.61556838e-05]
 [9.99988958e-01 1.10420554e-05]
 [9.99925231e-01 7.47690449e-05]
 [9.99970563e-01 2.94373927e-05]
 [9.99977937e-01 2.20630272e-05]
 [9.99969140e-01 3.08600105e-05]
 [9.99976644e-01 2.33556504e-05]
 [9.99986529e-01 1.34714151e-05]
 [9.99857895e-01 1.42105239e-04]
 [9.99984412e-01 1.55876916e-05]
 [9.99961808e-01 3.81916605e-05]
 [9.99974562e-01 2.54379071e-05]
 [9.99983918e-01 1.60824221e-05]]
confusion matrix for train [[60  0]
 [ 0  9]]
confusion matrix for test [[15  0]
 [ 0  4]]
coefficient is  [[-9.06346564e-02 -1.18833175e-01 -2.01884851e-02  1.84768254e-02
  -1.48130124e-01  2.44892836e-02 -2.06616438e-01 -9.66633852e-02
  -8.66684197e-02 -1.13604829e-02  4.39995684e-02  3.64335222e-02
  -1.13248808e-02 -1.58957377e-01 -1.96734144e-02 -1.52222606e-02
  -7.35988436e-02 -1.22102885e-02 -4.05682156e-02 -6.84654808e-02
  -1.89427256e-02 -1.84604205e-02 -3.01037970e-02 -1.55132665e-02
  -1.56363649e-01  1.52976519e-01  1.07165209e-01 -7.72903648e-02
   1.64137958e-02  4.47754740e-04  1.15175963e-01 -4.22134500e-02
  -5.65816746e-02 -7.93222355e-02 -8.40224767e-03 -3.88357616e-02
   2.01173656e-04  1.51726104e-01  1.79834599e-02 -1.93332336e-02
  -1.44953338e-01 -1.76874781e-02 -3.31182803e-02 -5.97028143e-02
  -1.29265140e-02 -2.11796536e-02  1.28886335e-02 -4.45563178e-03
   2.52629274e-01  4.11818520e-01  9.72554631e-02  9.94568000e-02
   1.03972940e-01 -3.89655828e-02  6.60078302e-02 -8.25034148e-03
  -1.06155331e-01  1.95055098e-01  1.02749004e-01 -7.30074111e-02
  -2.31112630e-02 -1.15541129e-02 -3.67363624e-03 -3.57953247e-02
  -1.54573753e-01 -2.55182341e-02 -2.89288000e-02 -1.52694298e-01
  -2.33634365e-02 -3.17110798e-02 -1.04792557e-01 -1.70852698e-02]]
interception is  [-0.0095948]
p values is  [5.29861542e-01 3.68305945e-01 3.90809400e-01 7.91639315e-01
 6.35189925e-01 3.00640363e-01 2.61624700e-01 2.09274058e-01
 2.11266284e-01 5.59859763e-01 2.72615917e-01 3.36876736e-01
 6.17960561e-03 1.75989568e-04 7.10204612e-02 6.50807497e-03
 2.77992618e-03 1.41864675e-01 4.61191214e-03 1.19123025e-05
 6.18599357e-02 5.80969269e-03 1.48828054e-05 6.13160620e-02
 5.62134182e-02 9.60767045e-03 8.52216215e-02 3.39298288e-02
 3.40277344e-03 5.60385720e-02 2.48477006e-02 9.52650251e-02
 7.80065167e-01 2.56792740e-01 7.41927166e-02 6.99771336e-01
 6.63097219e-02 6.76822845e-01 6.50277663e-01 7.41023405e-02
 2.55285512e-01 5.82781838e-01 3.74342641e-02 8.04863962e-02
```

```
 3.84282042e-01 3.64780131e-02 1.89465164e-01 4.14344112e-01
 1.10673780e-17 1.14303593e-14 5.49848144e-01 4.31583655e-11
 7.32698490e-09 8.29526845e-01 1.23117987e-08 7.51308970e-05
 2.73720348e-01 1.92780635e-09 1.42105778e-05 3.48598320e-01
 3.40417259e-02 2.36545870e-01 4.22990692e-01 4.03173174e-02
 6.41128112e-03 2.99411830e-01 2.27024413e-02 6.59279327e-04
 1.88694179e-01 1.32986085e-02 5.02792857e-04 1.19897416e-01]
The test score is 1.0
```



d_v: for the cross validation accuracy, the score is 0.9714285714285715, and the test score is 1.0

d_vi: That is true, my class have some kind of well-separated problem which causes instability. However, I can use some penalty to penalize them and I can have a good result.

d_vii: There is no imbalanced classes, since the definition of imbalanced classes is that ratio of one number of one class to the other classes is going to zero. But in this question, the lowest ratio is 0.2, not zero.

```
In [20]: def e_1():
             C_1 = []
             train_score_1 = []
             test_score_1 = []
             test_data, train_data, whole_data = create_samples(path)
             l_1 = range(1, 21, 1)
             for l in range(1, 21, 1):
                 train = d_3_1(train_data, l, 0)
```

```python
            train_2 = (DataFrame(train)).values
            x_train = np.delete(train_2, train_2.shape[1] - 1, axis=1)
            y_train = train_2[:, train_2.shape[1] - 1]

            test = d_3_1(test_data, l, 1)
            test_2 = (DataFrame(test)).values
            x_test = np.delete(test_2, test_2.shape[1] - 1, axis=1)
            y_test = test_2[:, test_2.shape[1] - 1]
            print(train_2.shape)

            train_set_x = normalize(x_train, axis=1)
            test_set_x = normalize(x_test, axis=1)
            cv = StratifiedKFold(n_splits=5)  # stratified method, 5 folds
            classifier_new = LogisticRegressionCV(scoring='accuracy', penalty='l1', solve
            classifier_new.fit(train_set_x, y_train)
            train_set_predic = classifier_new.predict(train_set_x)
            train_score = classifier_new.score(train_set_x, y_train)
            test_score = classifier_new.score(test_set_x, y_test)
            C = classifier_new.C_
            C_1.append(C)
            train_score_1.append(train_score)
            print(train_score)
            print(test_score)
            test_score_1.append(test_score)

        num = (int)(np.argmax(np.array(test_score_1)))
        print('The best C is ', C_1[num])
        print('The bset test accuracy is ', test_score_1[num])
        print('The best l is ', l_1[num])

    e_1()
```

```
(69, 19)
0.9855072463768116
1.0
(69, 37)
1.0
0.9473684210526315
(69, 55)
1.0
0.9473684210526315
(69, 73)
0.9855072463768116
0.9473684210526315
(69, 91)
1.0
0.9473684210526315
(69, 109)
```

```
1.0
0.9473684210526315
(69, 127)
1.0
0.9473684210526315
(69, 145)
1.0
1.0
(69, 163)
1.0
0.9473684210526315
(69, 181)
1.0
1.0
(69, 199)
1.0
0.9473684210526315
(69, 217)
1.0
0.9473684210526315
(69, 235)
0.9855072463768116
0.9473684210526315
(69, 253)
0.9855072463768116
0.9473684210526315
(69, 271)
1.0
0.9473684210526315
(69, 289)
1.0
0.9473684210526315
(69, 307)
1.0
0.9473684210526315
(69, 325)
0.9855072463768116
0.9473684210526315
(69, 343)
0.9855072463768116
0.9473684210526315
(69, 361)
1.0
1.0
The best C is  [21.5443469]
The bset test accuracy is  1.0
The best l is  1
```

e_ii: The method in L1 penalized logistic regresstion is better, since it is very quick to see the results, and the test score is up to 1, it is very high actually.

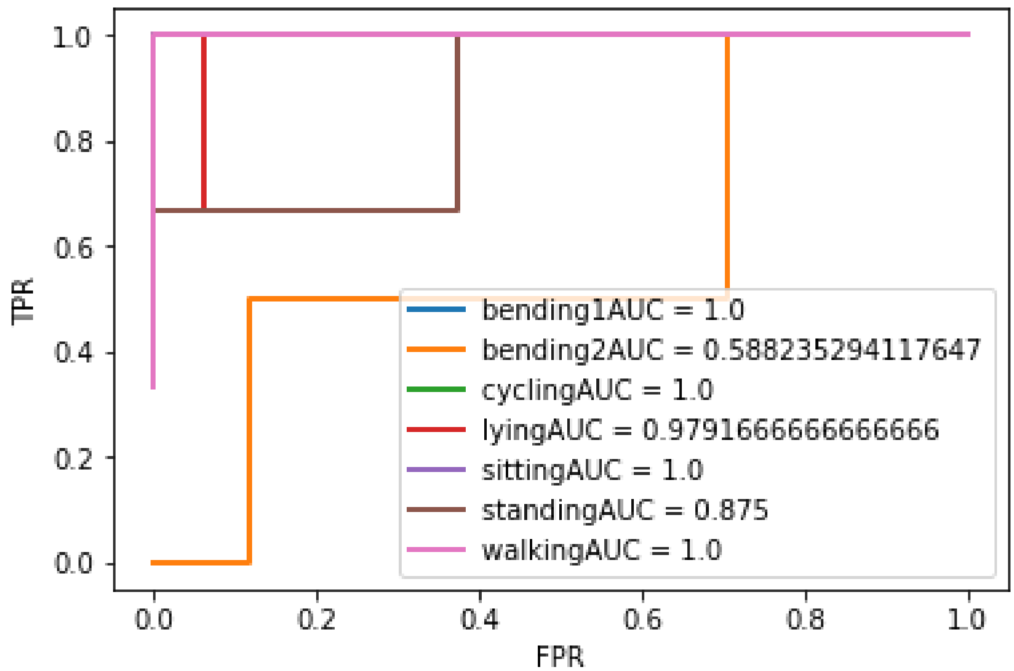
```
In [33]: def f_1(classifier_name='LogisticRegression'):
             C_1 = []
             train_score_1 = []
             test_score_1 = []
             test_sample = []
             train_sample = []
             l_1 = range(1, 21, 1)
             test_data, train_data, whole_data, y_train, y_test = create_samples_multi(path)
             activity = ['bending1', 'bending2', 'cycling', 'lying', 'sitting', 'standing', 'wa
             for l in range(1, 21, 1):
                 train = d_3_1(train_data, l, 0)
                 train_2 = (DataFrame(train)).values
                 x_train_1 = np.delete(train_2, train_2.shape[1] - 1, axis=1)

                 test = d_3_1(test_data, l, 1)
                 test_2 = (DataFrame(test)).values
                 x_test_1 = np.delete(test_2, test_2.shape[1] - 1, axis=1)
                 cv = StratifiedKFold(n_splits=5)  # stratified method, 5 folds
                 test_sample.append(x_test_1)
                 train_sample.append(x_train_1)

                 if classifier_name == 'LogisticRegression':
                     classifier = LogisticRegressionCV(solver='liblinear', penalty='l1', multi_
                     x_train_1 = normalize(x_train_1)
                     x_test_1 = normalize(x_test_1)
                 elif classifier_name == 'GaussianNB':
                     classifier = GaussianNB()
                 elif classifier_name == 'MultinomialNB':
                     classifier = MultinomialNB()
                 else:
                     pass

                 classifier.fit(x_train_1, y_train)
                 train_set_predic = classifier.predict(x_train_1)
                 test_set_predic = classifier.predict(x_test_1)
                 vc_matrix_test = confusion_matrix(y_test, test_set_predic)
                 vc_matrix_train = confusion_matrix(y_test, test_set_predic)

                 test_score = classifier.score(x_test_1, y_test)

                 test_error = 1 - classifier.score(x_test_1, y_test)
                 print('Test error = ' + str(test_error))
                 test_score_1.append(test_error)
```

```
            Num = (int)(np.argmin(test_score_1))
            plt.figure()
            xdata = dict()
            ydata = dict()
            x_train_1 = train_sample[Num]
            x_test_1 = test_sample[Num]
            print('The lowest test error is ', test_score_1[Num])
            print('The best l is ', l_1[Num])

            if classifier_name == 'LogisticRegression':
                classifier = LogisticRegressionCV(solver='liblinear', penalty='l1', multi_cla
                x_train_1 = normalize(x_train_1)
                x_test_1 = normalize(x_test_1)
            elif classifier_name == 'GaussianNB':
                classifier = GaussianNB()
            elif classifier_name == 'MultinomialNB':
                classifier = MultinomialNB()
            else:
                pass
            classifier.fit(x_train_1, y_train)
            test_score = classifier.predict_proba(x_test_1)
            train_set_predic = classifier.predict(x_train_1)

            test_set_predic = classifier.predict(x_test_1)
            vc_matrix_test = confusion_matrix(y_train , train_set_predic)
            vc_matrix_train = confusion_matrix(y_test, test_set_predic)

            print('Following is the confusion matrix of test: ')
            print(vc_matrix_test)
            print('Following is the confusion matrix of train: ')
            print(vc_matrix_train)

            auc_var = dict()
            test_set_y = label_binarize(y_test, classes=activity)
            for i in range(0, 7):
                res = roc_curve(test_set_y[:, i], test_score[:, i])
                xdata[i] = res[0]
                ydata[i] = res[1]
                auc_var[i] = auc(res[0], res[1])
                plt.plot(res[0], res[1], lw=2, label=activity[i] + 'AUC = ' + str(auc_var[i])
            plt.legend(loc='lower right')
            plt.xlabel('FPR')
            plt.ylabel('TPR')
```

```
In [34]: f_1(classifier_name='LogisticRegression')
         plt.show()
```

```
D:\G\Anaconda3_python3.6\lib\site-packages\sklearn\model_selection\_split.py:605: Warning: The
```

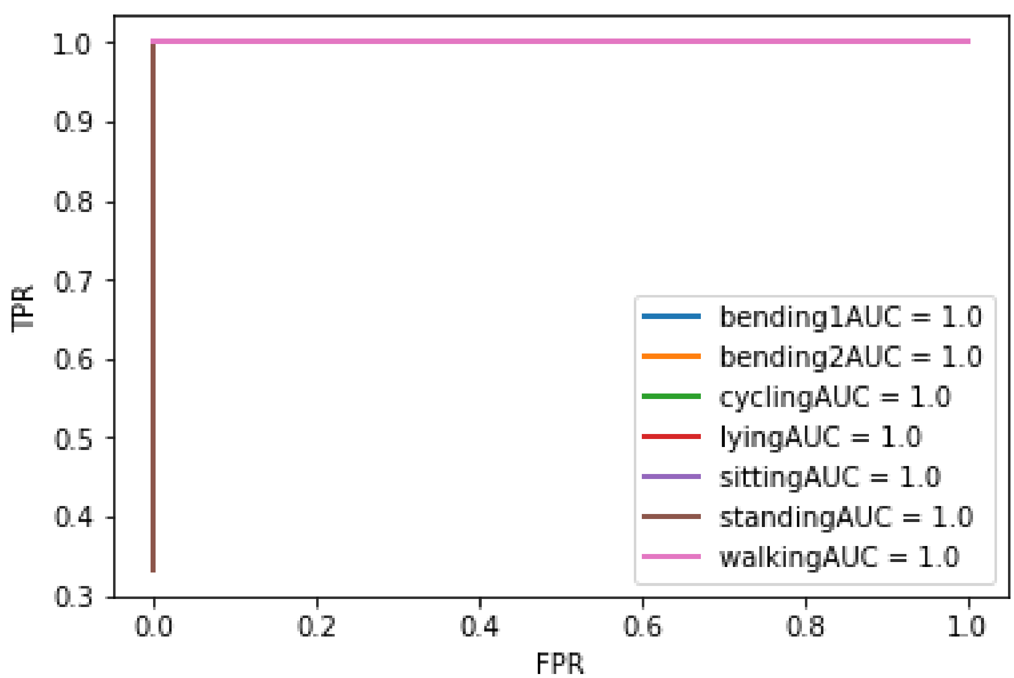
```
  % (min_groups, self.n_splits)), Warning)


Test error = 0.368421052631579
Test error = 0.368421052631579
Test error = 0.3157894736842105
Test error = 0.21052631578947367
Test error = 0.368421052631579
Test error = 0.26315789473684215
Test error = 0.3157894736842105
Test error = 0.1578947368421053
Test error = 0.26315789473684215
Test error = 0.3157894736842105
Test error = 0.3157894736842105
Test error = 0.3157894736842105
Test error = 0.368421052631579
Test error = 0.368421052631579
Test error = 0.3157894736842105
Test error = 0.5263157894736843
Test error = 0.4736842105263158
Test error = 0.368421052631579
Test error = 0.42105263157894735
Test error = 0.42105263157894735
The lowest test error is  0.1578947368421053
The best l is  8
Following is the confusion matrix of test:
[[ 5  0  0  0  0  0  0]
 [ 0  4  0  0  0  0  0]
 [ 0  0 12  0  0  0  0]
 [ 0  0  0 12  0  0  0]
 [ 0  0  0  0 12  0  0]
 [ 0  0  0  0  0 12  0]
 [ 0  0  0  0  0  0 12]]
Following is the confusion matrix of train:
[[2 0 0 0 0 0 0]
 [1 0 1 0 0 0 0]
 [0 0 3 0 0 0 0]
 [0 0 0 3 0 0 0]
 [0 0 0 0 3 0 0]
 [0 0 0 1 0 2 0]
 [0 0 0 0 0 0 3]]
```




```
In [35]: f_1(classifier_name='GaussianNB')
         plt.show()
```

```
Test error = 0.10526315789473684
Test error = 0.052631578947368474
Test error = 0.10526315789473684
Test error = 0.10526315789473684
Test error = 0.1578947368421053
Test error = 0.1578947368421053
Test error = 0.21052631578947367
Test error = 0.1578947368421053
Test error = 0.1578947368421053
Test error = 0.1578947368421053
Test error = 0.21052631578947367
Test error = 0.26315789473684215
Test error = 0.3157894736842105
Test error = 0.1578947368421053
Test error = 0.26315789473684215
Test error = 0.21052631578947367
Test error = 0.21052631578947367
Test error = 0.26315789473684215
Test error = 0.21052631578947367
Test error = 0.26315789473684215
The lowest test error is  0.052631578947368474
The best l is  2
```

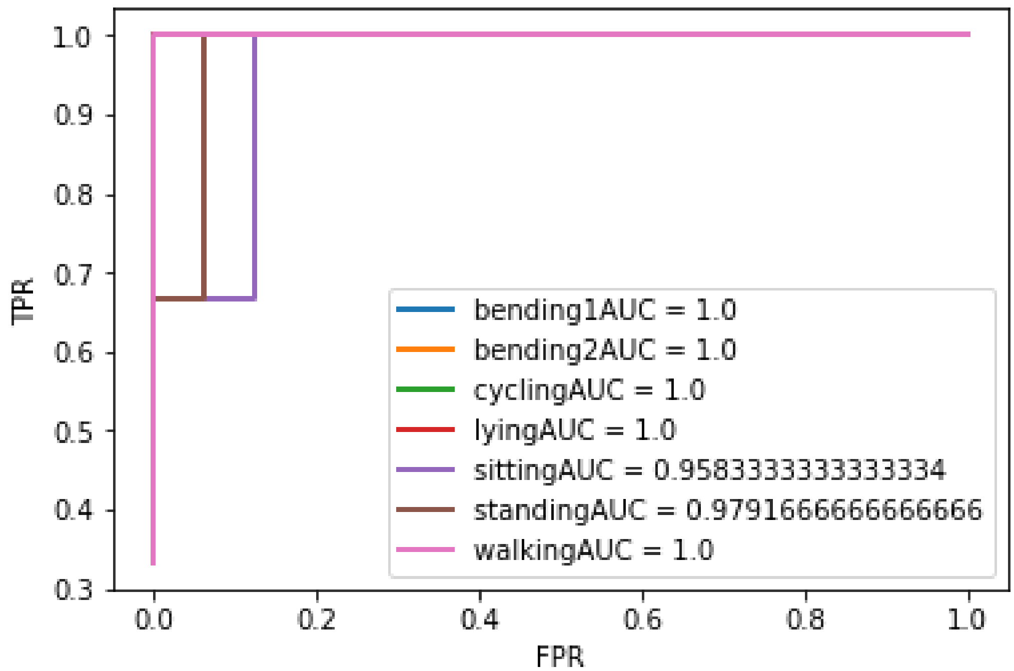
```
Following is the confusion matrix of test:
[[ 5  0  0  0  0  0  0]
 [ 0  3  0  1  0  0  0]
 [ 0  0 12  0  0  0  0]
 [ 0  0  0 12  0  0  0]
 [ 0  0  0  0 12  0  0]
 [ 0  0  0  0  0 12  0]
 [ 0  0  0  0  0  0 12]]
Following is the confusion matrix of train:
[[2 0 0 0 0 0 0]
 [0 2 0 0 0 0 0]
 [0 0 3 0 0 0 0]
 [0 0 0 3 0 0 0]
 [0 0 0 0 3 0 0]
 [0 0 0 0 1 2 0]
 [0 0 0 0 0 0 3]]
```




```
In [36]: f_1(classifier_name='MultinomialNB')
         plt.show()
```

```
Test error = 0.10526315789473684
Test error = 0.21052631578947367
Test error = 0.21052631578947367
Test error = 0.1578947368421053
```

```
Test error = 0.1578947368421053
Test error = 0.1578947368421053
Test error = 0.1578947368421053
Test error = 0.1578947368421053
Test error = 0.21052631578947367
Test error = 0.21052631578947367
Test error = 0.052631578947368474
Test error = 0.1578947368421053
Test error = 0.1578947368421053
Test error = 0.21052631578947367
Test error = 0.1578947368421053
Test error = 0.1578947368421053
Test error = 0.10526315789473684
Test error = 0.10526315789473684
Test error = 0.10526315789473684
Test error = 0.10526315789473684
The lowest test error is  0.052631578947368474
The best l is  11
Following is the confusion matrix of test:
[[ 5  0  0  0  0  0  0]
 [ 1  3  0  0  0  0  0]
 [ 0  0 12  0  0  0  0]
 [ 0  0  0 12  0  0  0]
 [ 0  0  0  0 12  0  0]
 [ 0  0  0  0  1 11  0]
 [ 0  0  0  0  0  0 12]]
Following is the confusion matrix of train:
[[2 0 0 0 0 0 0]
 [0 2 0 0 0 0 0]
 [0 0 3 0 0 0 0]
 [0 0 0 3 0 0 0]
 [0 0 0 0 2 1 0]
 [0 0 0 0 0 3 0]
 [0 0 0 0 0 0 3]]
```



f_iii: The best two is Multinomial priors and Gaussian, since its test error is same.