report_executed

August 5, 2021

1 Statistics

In the statistics module we analyze data for different responses and at different spectral peak locations. We use Python package scipy in this module.

1.1 T-Test

T-test checks for difference in the mean between two sample from different responses. We assume the data is independent and follows the normality assumption. Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ be the two samples and we test whether the means are equal. The null hypothesis states means μ_1 and μ_2 are equal and the alternative hypothesis states they are not equal. If the p-value is lower than the chosen significance level, we can reject the null hypothesis, i.e. the samples do not have the same means.

```
[1]:        import modules.adapml_data as adapml_data
            import modules.adapml_classification as adapml_classification
            import modules.adapml_clustering as adapml_clustering
            import modules.adapml_chemometrics as adapml_chemometrics
            import modules.adapml_statistics as adapml_statistics
            import modules.adapml_regression as adapml_regression
            import numpy as np
            import modules.loadTestData as load_data
            import sklearn.preprocessing as pre
            from sklearn.cross_decomposition import PLSRegression as PLS
            from matplotlib import pyplot as plt
            from sklearn import cluster as clst
            from scipy.cluster.hierarchy import dendrogram

            import os

            reldir = os.getcwd()
            path_to_data = os.path.join(reldir, '..', 'data',␣
     ↪'SCLC_study_output_filtered_2.csv')

            data = adapml_data.DataImport(path_to_data)

            response1D = data.resp
            #response1D = adapml_data.DataImport.getResponse(path_to_data)
```
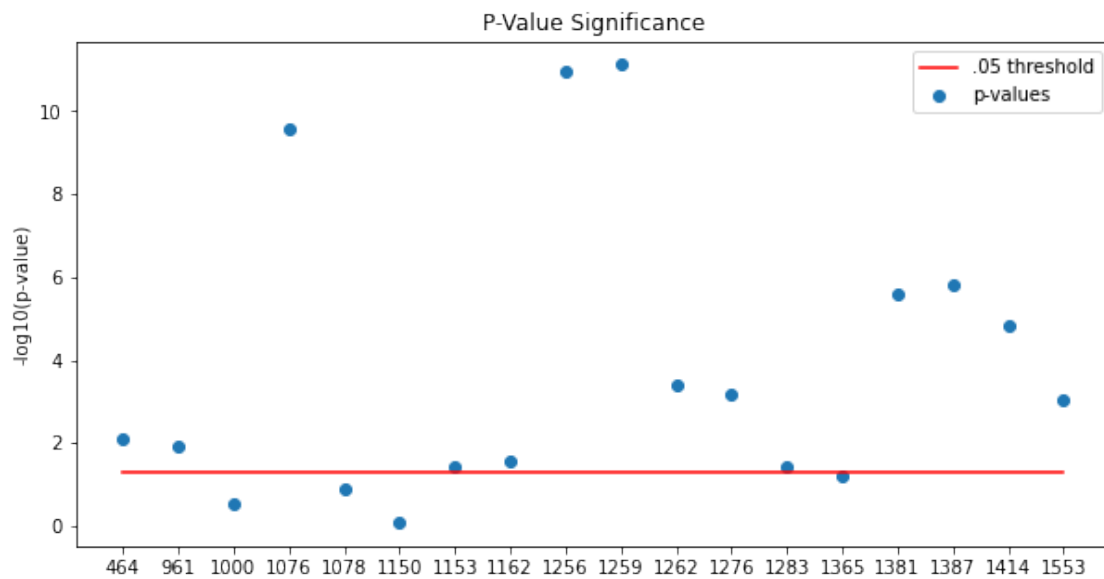
```
        response2D = adapml_data.DataImport.getDummyResponse(response1D)

        variables = data.getVariableNames()
        samples = data.getSampleNames()

        t_test = adapml_statistics.Statistics(data.data, 'anova', response1D)
        t_test.plot_logp_values(variables)
```
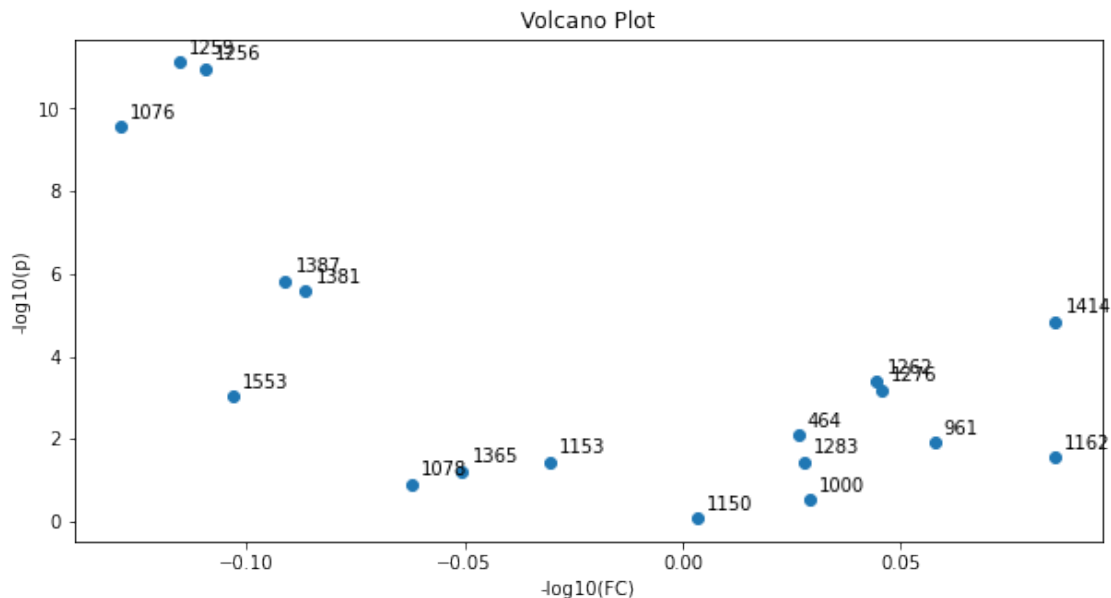

1.2 Volcano Plot

Volcano plot is a scatter plot which demonstrates magnitude between the responses and t-test significance of the data. We can choose a significance level and fold change limit to specify the rectangle of interest.

```
[2]: t_test.plot_volcano_t(variables)
```

Volcano Plot

2   Dimension-Reduction

Dimension-reduction methods are used to condense high dimensional data down to dimensions which provide the most information. We have implemented the principal component analysis (PCA). It performs a change of basis and the new basis is chosen, such that the i-th principal component is orthogonal to the first i-1 principal components and the direction maximizes the variance of the projected data. We use the Python library sklearn.

2.1   Principal Component Analysis

The principal component analysis (PCA) is one of the methods for dimension-reduction. It performs a change of basis and the new basis is chosen, such that the i-th principal component is orthogonal to the first i-1 principal components and the direction maximizes the variance of the projected data. Instead of considering all the dimensions, we pick the necessary number of principal components.

```
[3]: data.normalizeData("autoscale")

pca = adapml_chemometrics.Chemometrics(data.data, "pca", response1D)

print("PCA Projections");pca.plotProjectionScatterMultiClass(2,␣
 ↪labels=["Healthy", "Not Healthy"])
```
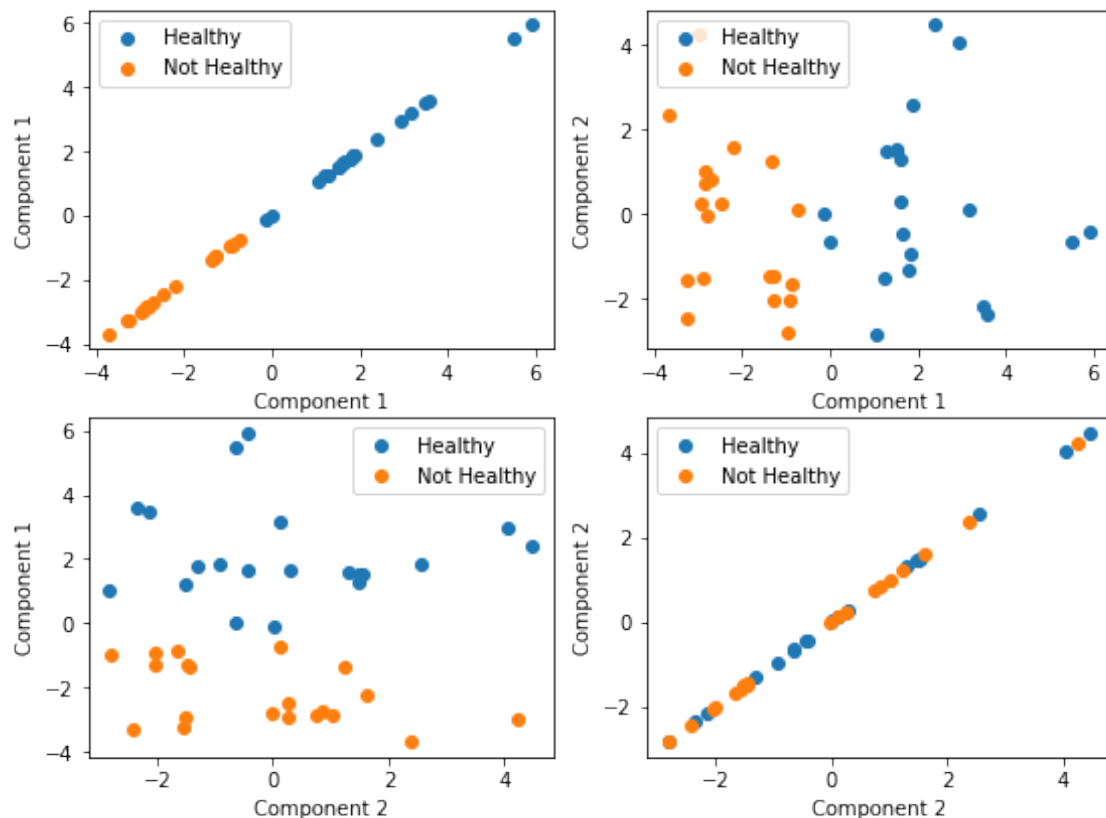
```
PCA Projections
Projections of data into latent space.
Data is colored by response
```

2.2 Linear Discriminant Analysis

Linear discriminant analysis is a classifier with a linear decision boundary. We assume normality and fit conditional densities $p(x \mid y = 0)$ and $p(x \mid y = 1)$ with mean and covariance parameters (μ_0, σ_0) and (μ_1, σ_1), where x, μ_0 and μ_1 are vectors. Dimensionality-reduction is done by projecting the input to the most discriminative directions.
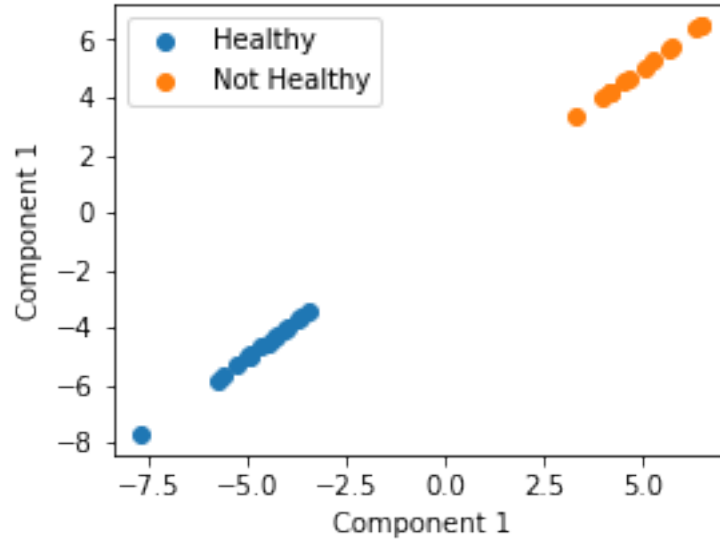
```
[4]: lda = adapml_chemometrics.Chemometrics(data.data, "lda", response1D) # Also
     ↪Predicts

     print("LDA Projections");lda.plotProjectionScatterMultiClass(1,
     ↪labels=["Healthy", "Not Healthy"])
```

```
LDA Projections
Projections of data into latent space.
Data is colored by response
```

3 Clustering

In this module we use various different clustering methods on spectra. Clustering is done with scipy and sklearn libraries.

3.1 K-Means Clustering

K-means clustering aims to partition the data into k sets and to minimize the within-cluster sum of squares (WCSS)

$$WCSS = \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$$

where $x_1, \ldots, x_n$ is the data and μ_i is the centroid of C_i cluster. It is solved by either Lloyd's or Elkan's algorithm and we use sklearn module in Python.

```
[5]: kmeans_cluster = adapml_clustering.Clustering(data.data, 'kmeans', 3)
     kmeans_cluster.getClusterResults(samples)
```

	Cluster 1	Cluster 2	Cluster 3
0	NSCLC_H358_2	NSCLC_A549_1	SCLC_86M1_2
1	NSCLC_H522_1	NSCLC_H1703_2	SCLC_86M1_1
2	NSCLC_H522_2	NSCLC_H1703_1	SCLC_16HV_1
3	NSCLC_H358_1	NSCLC_A549_2	SCLC_16HV_2
4	NSCLC_PC9_1	NSCLC_H1437_1	SCLC_DMS79_1
5	NSCLC_PC9_2	NSCLC_H2228_1	SCLC_DMS79_2
6	NaN	NSCLC_H2228_2	SCLC_H187_2
7	NaN	NSCLC_H1437_2	SCLC_H187_1
8	NaN	NSCLC_H3122_1	SCLC_H209_1
9	NaN	NSCLC_H322_2	SCLC_H524_1

```
10        NaN    NSCLC_H322_1     SCLC_H209_2
11        NaN    NSCLC_H3122_2    SCLC_H524_2
12        NaN    NSCLC_HCC4006_1   SCLC_H69_1
13        NaN    NSCLC_HCC4006_2   SCLC_H82_1
14        NaN              NaN     SCLC_H82_2
15        NaN              NaN     SCLC_H69_2
16        NaN              NaN    SCLC_N417_2
17        NaN              NaN    SCLC_N417_1
18        NaN              NaN  SCLC_SW210-5_1
19        NaN              NaN  SCLC_SW210_5_2
```

3.2   Hierarchical Clustering

```
[6]: hierarchical_cluster = adapml_clustering.Clustering(data.data, 'hierarchical',␣
     ↪3)
     hierarchical_cluster.getClusterResults(samples)
     hierarchical_cluster.plot_dendrogram(samples)
```

```
           Cluster 1          Cluster 2       Cluster 3
0        SCLC_86M1_2      NSCLC_A549_1    NSCLC_H358_2
1        SCLC_86M1_1     NSCLC_H1703_2    NSCLC_H522_1
2        SCLC_16HV_1     NSCLC_H1703_1    NSCLC_H522_2
3        SCLC_16HV_2      NSCLC_A549_2    NSCLC_H358_1
4       SCLC_DMS79_1     NSCLC_H1437_1     NSCLC_PC9_1
5       SCLC_DMS79_2     NSCLC_H2228_1     NSCLC_PC9_2
6        SCLC_H187_2     NSCLC_H2228_2             NaN
7        SCLC_H187_1     NSCLC_H1437_2             NaN
8        SCLC_H209_1     NSCLC_H3122_1             NaN
9        SCLC_H524_1      NSCLC_H322_2             NaN
10       SCLC_H209_2      NSCLC_H322_1             NaN
11       SCLC_H524_2     NSCLC_H3122_2             NaN
12        SCLC_H69_1   NSCLC_HCC4006_1             NaN
13        SCLC_H82_1   NSCLC_HCC4006_2             NaN
14        SCLC_H82_2              NaN             NaN
15        SCLC_H69_2              NaN             NaN
16       SCLC_N417_2              NaN             NaN
17       SCLC_N417_1              NaN             NaN
18     SCLC_SW210-5_1            NaN             NaN
19     SCLC_SW210_5_2            NaN             NaN
```
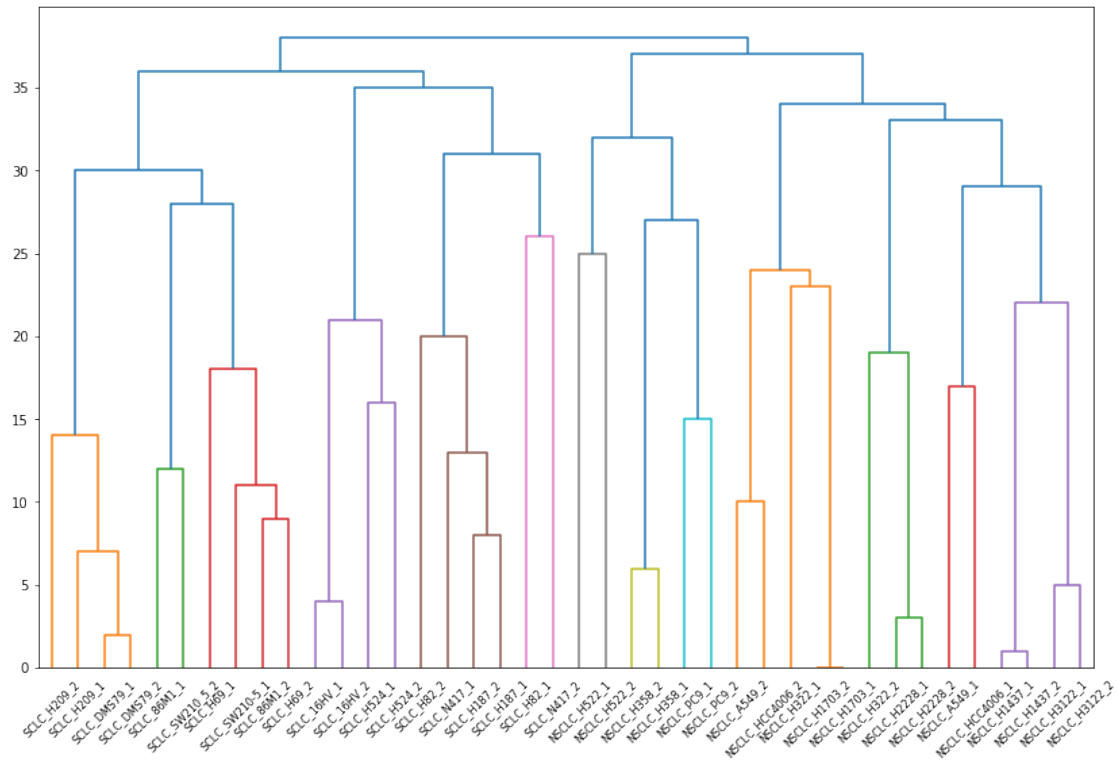
4 Classification

4.1 Partial Least Squares-Discriminant Analysis

```
[7]: def plotProjectionScatterMultiClass(pc, resp, num_var):
         plt.figure(figsize=(24, 18))

         for i in range(num_var):
             for j in range(num_var):
                 plt.subplot(5,5,5*(i) + j + 1)
                 for c in range(resp.shape[1]):
                     inx = np.where(resp[:,c] == 1)[0]
                     tmp = pc[inx,:]
                     pc1 = tmp[:,i]
                     pc2 = tmp[:,j]
                     plt.scatter(pc1, pc2)
                 plt.xlabel("PLS Component "+str(i+1))
                 plt.ylabel("PLS Component "+str(j+1))

         plt.show()

     data = load_data.loadDataPandas(path_to_data)
```

```
d = data.to_numpy()
var_index = data.columns.values.tolist()

resp = load_data.getResponseMatrix2D()

norm_trans = pre.StandardScaler().fit(d)
data_norm = norm_trans.transform(d)
#data_norm, norm_trans = pre.mean_center(d)
#In-built preprocessing method - TBD

pls = PLS().fit(data_norm, resp)
pls_trans = pls.transform(data_norm)

plotProjectionScatterMultiClass(pls_trans, resp, 2)
```
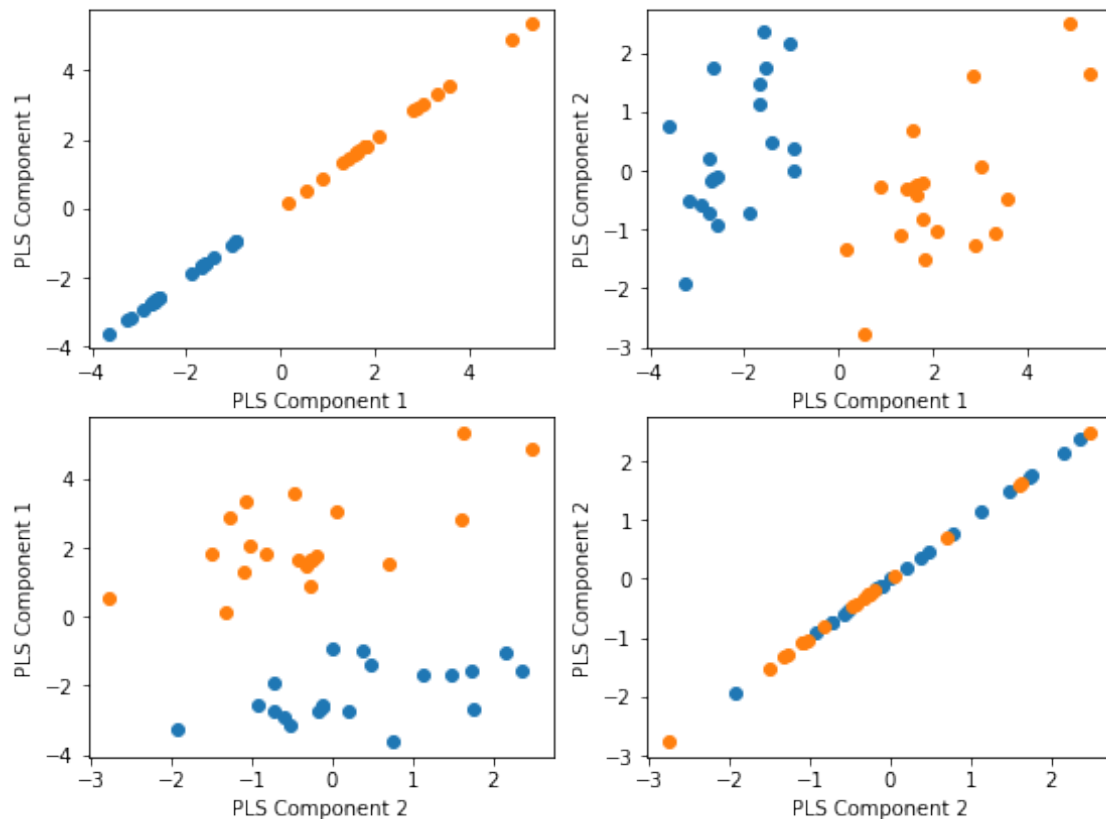

4.2 Support Vector Machines

```
[8]: data = adapml_data.DataImport(path_to_data)
svm = adapml_classification.Classification(data.data, response1D, 'svm', .75,
 ↪kfolds=3)
```

```
adapml_classification.print_model_stats(svm, "SVM")
```

```
SVM Validated Parameters:  {'kernel': 'linear', 'shrinking': True}
SVM: R^2=1.0 Q^2=1.0
```

4.3 Random Forest

```
[9]: data = adapml_data.DataImport(path_to_data)
     rnf = adapml_classification.Classification(data.data, response1D,␣
      ↪'randomforest', .75, kfolds=3)

     adapml_classification.print_model_stats(rnf, "RF")
```
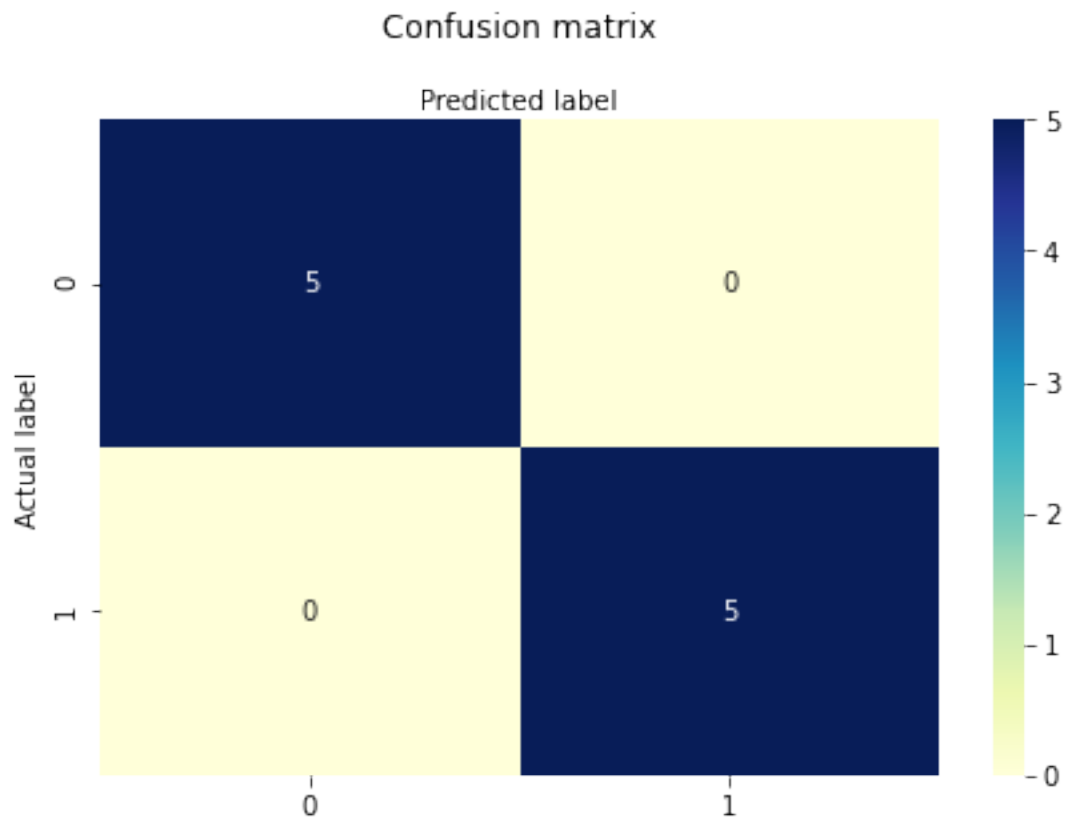
```
Random Forest Validated Parameters:  {'criterion': 'gini', 'n_estimators': 10}
RF: R^2=1.0 Q^2=1.0
```

4.4 Logistic Regression

```
[10]: data = adapml_data.DataImport(path_to_data)

      logistic = adapml_classification.Classification(data.data, response1D,␣
       ↪'logistic', .25)
      print(logistic)
```

```
Accuracy: 1.0
<modules.adapml_classification.Classification object at 0x11eaf24c0>
```

Confusion matrix

5 Regression

5.1 Linear Regression

```
[11]: reg = adapml_regression.Regression(data.data, "linear")
      reg.linear
```