Exercise 9.1 Creating a decision tree by hand

We will here create a decision tree based on table 1 which lists the cases for which our friend Hermann is playing tennis. We see here that we have two possible classes for playing

Outlook	Wind	Humidity	Playing tennis
overcast	strong	normal	yes
sunny	strong	normal	yes
rain	weak	high	yes
sunny	strong	high	no
sunny	weak	high	no
sunny	weak	normal	yes
rain	strong	high	no

Table 1: *Conditions for which Hermann is playing tennis.*

tennis, "yes" or "no", and this is what we want to predict, given the values for the attributes "outlook", "wind" and "humidity". For instance, we see that if the outlook is rain, the wind is weak and the humidity is high, Hermann will play tennis.

In order to create such a decision tree, we will have to rank the different attributes according to which ones gives us a better separation into the two classes. We can here see in table 1 that if the humidity is normal, Hermann will always play tennis. If it is high, most often he will not play tennis, but sometimes he will (depending on the outlook and wind). Thus, this attributes seems like a good one to use for predicting if tennis is to be played. However, can we quantitatively say if it is better than using, for instance, the outlook? The answer to this is yes, and we do this by introducing the entropy, E, and the information gain G. We can then calculate these properties for each of the attributes, and rank them according to the value of G. The attribute giving the largest G is the attribute that best separate the data into the classes.

The entropy, E, is defined as

$$E = -\sum_{i=1}^{K} p_i \log_2 p_i,$$

where K is the number of classes (2 in this present case) and p_i is given by,

$$p_i = \frac{\text{number of objects in class } i}{\text{total number of objexts}}.$$

Further, we will use the information gain, G, which is calculated for a given attribute, A, as,

$$G(S, A) = E(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} E(S_v).$$

Here, S is the collection of objects we currently have, $|S|$ is the number of objects and $V(A)$ is the set of all possible values the attribute A can have. Further, S_v is the subset of objects for a particular value v of the attribute A, and $|S_v|$ is the number of such objects.

Given our original table (see table 1), we see that we have two classes, "yes" and "no", and three attributes, "outlook", "wind" and "humidity". For these three attributes, we see that:

- The outlook can be "overcast", "sunny" or "rain". That is,

$$V(\text{outlook}) = \{\text{overcast}, \text{sunny}, \text{rain}\}.$$

- The wind can be "strong" or "weak". That is,

$$V(\text{wind}) = \{\text{strong}, \text{weak}\}.$$

- The humidity can be "normal" or "high". That is,

$$V(\text{humidity}) = \{\text{normal}, \text{high}\}.$$

Selecting the first attribute

We begin by calculating the initial entropy for the full set,

$$E_0 = -p_{\text{no}} \log_2 p_{\text{no}} - p_{\text{yes}} \log_2 p_{\text{yes}} = -\left(\tfrac{3}{7} \log_2 \tfrac{3}{7} + \tfrac{4}{7} \log_2 \tfrac{4}{7}\right) = 0.985,$$

and we check the information gain for each of the 3 attributes:

1. Considering only the outlook, we have three possible values as shown in table 2. From

Outlook	**Playing tennis**
overcast	yes
sunny	yes
sunny	no
sunny	no
sunny	yes
rain	yes
rain	no

Table 2: *Tennis-playing conditions for the attribute outlook.*

this table, we see that $|S_{\text{overcast}}| = 1$, $|S_{\text{sunny}}| = 4$ and $|S_{\text{rain}}| = 2$, and the corresponding entropies are,

$$\begin{aligned} E(S_{\text{overcast}}) &= -\tfrac{1}{1}\log_2 1 = 0, \\ E(S_{\text{sunny}}) &= -\tfrac{2}{4}\log_2 \tfrac{2}{4} - \tfrac{2}{4}\log_2 \tfrac{2}{4} = 1, \\ E(S_{\text{rain}}) &= -\tfrac{1}{2}\log_2 \tfrac{1}{2} - \tfrac{1}{2}\log_2 \tfrac{1}{2} = 1. \end{aligned}$$

The gain is thus,

$$G(S_0, \text{outlook}) = E_0 - (\tfrac{1}{7} \times 0 + \tfrac{2}{7} \times 1 + \tfrac{4}{7} \times 1) = 0.128.$$

2. For the wind, we have two possible values as shown in table 3. From this table, we see

Wind	**Playing tennis**
strong	yes
strong	yes
strong	no
strong	no
weak	yes
weak	yes
weak	no

Table 3: *Tennis-playing conditions for the attribute wind.*

that $|S_{\text{strong}}| = 4$ and $|S_{\text{weak}}| = 3$, and the corresponding entropies are,

$$\begin{aligned} E(S_{\text{strong}}) &= -\tfrac{2}{4}\log_2 \tfrac{2}{4} - \tfrac{2}{4}\log_2 \tfrac{2}{4} = 1, \\ E(S_{\text{weak}}) &= -\tfrac{2}{3}\log_2 \tfrac{2}{3} - \tfrac{1}{3}\log_2 \tfrac{1}{3} = 0.918. \end{aligned}$$

The gain is thus,

$$G(S_0, \text{wind}) = E_0 - (\tfrac{4}{7} \times 1 + \tfrac{3}{7} \times 0.918) = 0.020.$$

3. For the humidity we have two possible values as shown in table 4. From this table, we see that $|S_{\text{normal}}| = 3$ and $|S_{\text{high}}| = 4$, and the corresponding entropies are,

$$\begin{aligned} E(S_{\text{normal}}) &= -\tfrac{3}{3}\log_2 \tfrac{3}{3} = 0, \\ E(S_{\text{high}}) &= -\tfrac{1}{4}\log_2 \tfrac{1}{4} - \tfrac{3}{4}\log_2 \tfrac{3}{4} = 0.811. \end{aligned}$$

The gain is thus,

$$G(S_0, \text{humidity}) = E_0 - (\tfrac{3}{7} \times 0 + \tfrac{4}{7} \times 0.811) = 0.522.$$

Humidity	Playing tennis
normal	yes
normal	yes
normal	yes
high	yes
high	no
high	no
high	no

Table 4: *Tennis-playing conditions for the attribute humidity.*

Comparing these tree gains, we see that it is largest for the humidity. This will therefore be the first attribute we check. If the humidity is normal, we are always playing tennis and we do not need more information. If the humidity is high, we do need more information and we need to check the other attributes. We can now reduce our original problem and take out the humidity and the cases where the humidity is normal. This leads to a new data table which we will use to select the second attribute.

Figure 1 shows a representation of what our decision tree looks like after this first step.

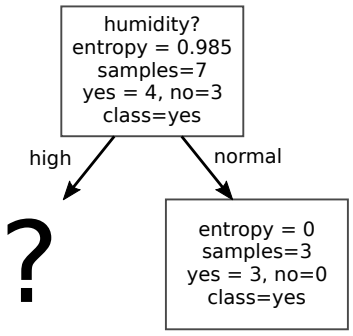

Figure 1: *Decision tree after selecting the first attribute.*

Selecting the second attributes

Having used the humidity attribute, we remove this column and the rows where this attribute is "normal" (since "playing tennis" is "yes" in all these cases). The reduced data table is shown in table 5. We now calculate the entropy for the current situation:

$$E_1 = -p_{\text{no}} \log_2 p_{\text{no}} - p_{\text{yes}} \log_2 p_{\text{yes}} = -\left(\tfrac{3}{4} \log_2 \tfrac{3}{4} + \tfrac{1}{4} \log_2 \tfrac{1}{4}\right) = 0.811,$$

Outlook	Wind	Playing tennis
rain	weak	yes
sunny	strong	no
sunny	weak	no
rain	strong	no

Table 5: *Conditions for which Hermann is playing tennis.*

and we check the information gain for the two attributes:

1. Considering only the outlook, we have two possible values as shown in table 6. From

Outlook	Playing tennis
rain	yes
rain	no
sunny	no
sunny	no

Table 6: *Tennis-playing conditions for the attribute outlook.*

this table, we see that $|S_{\text{rain}}| = 2$ and $|S_{\text{sunny}}| = 2$, and the corresponding entropies are,

$$E(S_{\text{rain}}) = -\tfrac{1}{2}\log_2\tfrac{1}{2} - \tfrac{1}{2}\log_2\tfrac{1}{2} = 1,$$
$$E(S_{\text{sunny}}) = -\tfrac{1}{1}\log_2\tfrac{1}{1} = 0.$$

The gain is thus,

$$G(S_1, \text{outlook}) = E_1 - (\tfrac{2}{4}\times 1 + \tfrac{2}{4}\times 0) = 0.311.$$

2. For the wind, we have also two possible values as shown in table 7. From this table,

Wind	Playing tennis
weak	yes
weak	no
strong	no
strong	no

Table 7: *Tennis-playing conditions for the attribute wind.*

we see that $|S_{\text{weak}}| = 2$ and $|S_{\text{strong}}| = 2$, and the corresponding entropies are,

$$E(S_{\text{weak}}) = -\tfrac{1}{2}\log_2\tfrac{1}{2} - \tfrac{1}{2}\log_2\tfrac{1}{2} = 1,$$
$$E(S_{\text{strong}}) = -\tfrac{1}{1}\log_2\tfrac{1}{1} = 0.$$

The gain is thus,

$$G(S_1, \text{wind}) = E_1 - (\tfrac{2}{4}\times 1 + \tfrac{2}{4}\times 0) = 0.311.$$

We here have a case where the two gains are equal and they are equally good. It does not matter which one we choose here, so we just (randomly) select the outlook as our second attribute.

Figure 2 shows a representation of what our decision tree looks like after this second step.

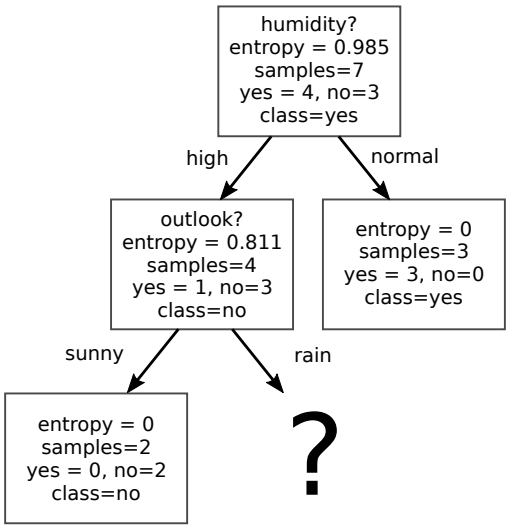

Figure 2: *Decision tree after selecting the second attribute.*

Selecting the third attribute

We have now also used the outlook attribute. This means that we have only one attribute left, and we do not need to do any more calculations. Reducing table 6 further by removing the outlook column and the sunny rows (as "playing tennis" is always "no" in these cases) we are left with the data table shown in table 8. This is enough information to finalize our decision tree which is shown in Fig. 3.

Wind	Playing tennis
weak	yes
strong	no

Table 8: *Conditions for which Hermann is playing tennis.*

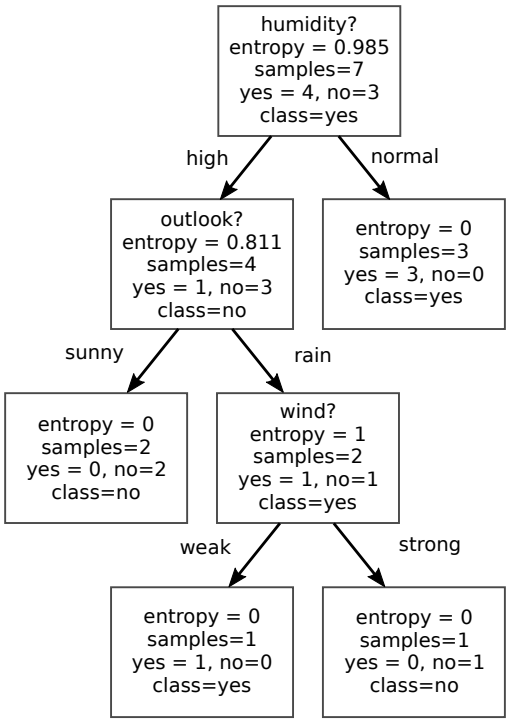

Figure 3: *Final decision tree.*

Exercise 9.2 Predicting concentrations from near-infrared spectra

The Python code for this exercise can be found in listing 1.

(a) See the answer below in point (b).

(b) The RMSECV and RMSEP are shown as a function of the number of principal components in Fig. 4. We see that there is a large drop in both values when going from 1 to around 5 components. A comparison of PLSR models with 1, 5 and 8 PLSR components, respectively, can be found in Figs. 5–7. We see here that we improve the PLSR model when including more components, and this is as expected. However, we prefer a model with a relatively small number of components (to avoid over-fitting), and we are satisfied with using just 5 components. This is motivated by the large drops in RMSECV and RMSEP observed up to 5 PLS components (see Fig. 4) and the fact

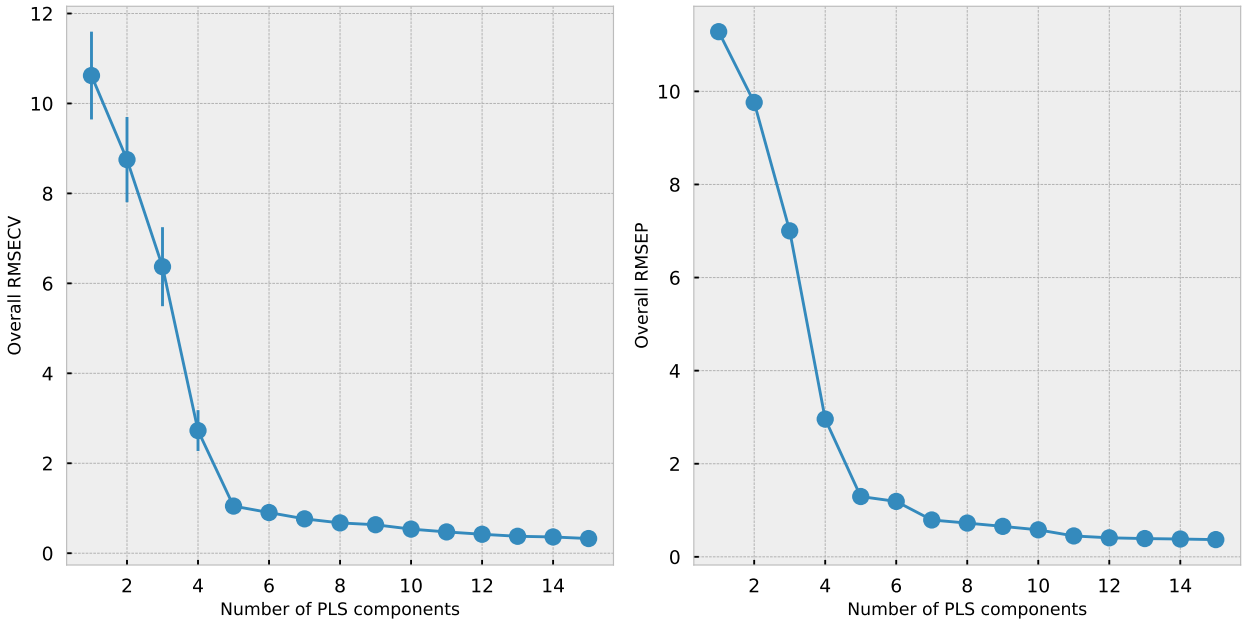

Figure 4: *RMSECV and RMSEP as a function of PLS components.*

that the model does not improve substantially when including more components (see Fig. 6 and Fig. 7).

(c) The regression coefficients can be found in Fig. 8. Interestingly, we see here that the regression coefficients show some resemblance to spectra and that they pick out different parts as being important for the different chemical species present.

(d) Here, we interpret the RMSEP values as representing the expected errors in our predictions. These are around 1 concentration units for methylene chloride, 2-butanol and methanol and around 2 concentration units for dichloropropane and acetone.

(e) Here, we have a situation where we are fitting a data set with 140 samples using 700 variables. We thus expect the least-squares model to be over-fitted and model the raw data almost perfectly. The results for the least-squares model, when applied to the test set, shows this (see Fig. 9), and the least-squared model is actually performing better than the PLSR model with 5 components.

The regression coefficients are shown in Fig. 10. The interpretation of the coefficients in Fig. 10 is less clear than in the corresponding figure for the PLSR model (see Fig. 8).

Exercise 9.3 Gene expressions in ovarian cancer

The Python code for this exercise can be found in listing 2.

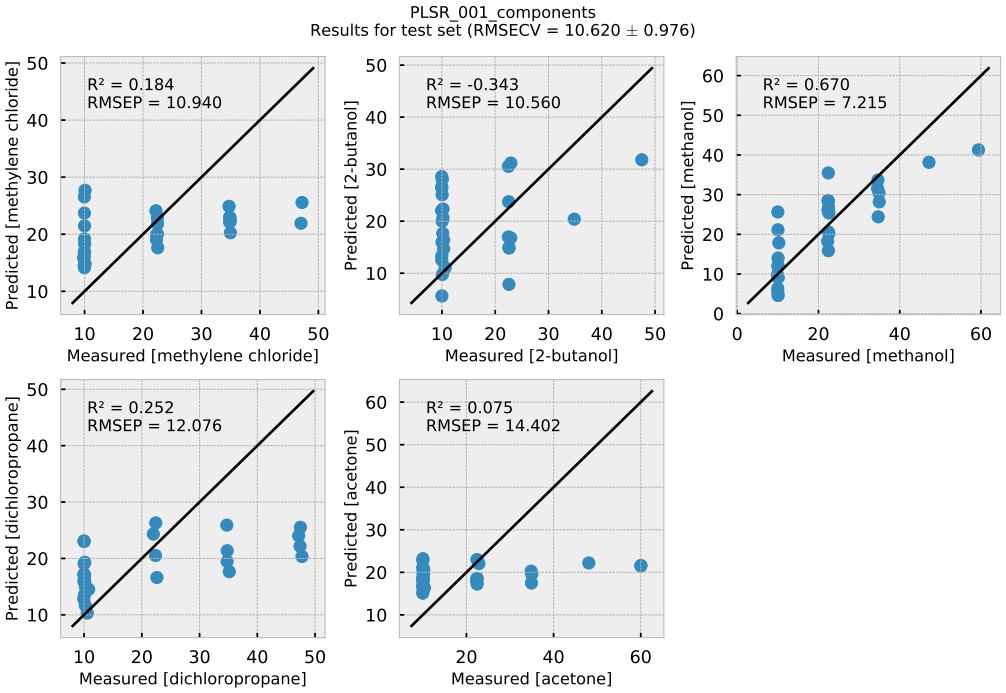

Figure 5: *Results for the test set when using 1 component for the PLSR model.*

(a) The explained variance as a function of the number of principal components can be found in Fig. 11, and the cumulative explained variance for 1, 2, 5, and 10 components are given in table 9. We see from this table (and from Fig. 11) that the explained variance is small, even with a relatively large number of components.

Number of principal components	**Explained variance**
1	0.13
2	0.21
5	0.36
10	0.49

Table 9: *Explained variance as a function of the number of principal components.*

(b) The scores and loadings for principal components 1 and 2 are shown in Fig. 12.

(i) We see some clustering into two groups (along PC1) with cancer and normal samples. We note that there is some overlap between the two clusters, in particular for samples 16, 21, 22, and 23.

(ii) The samples 34, 44 and 43 seem to be separated from the other points. These could potentially be outliers. To investigate this further, we plot the diagonal elements

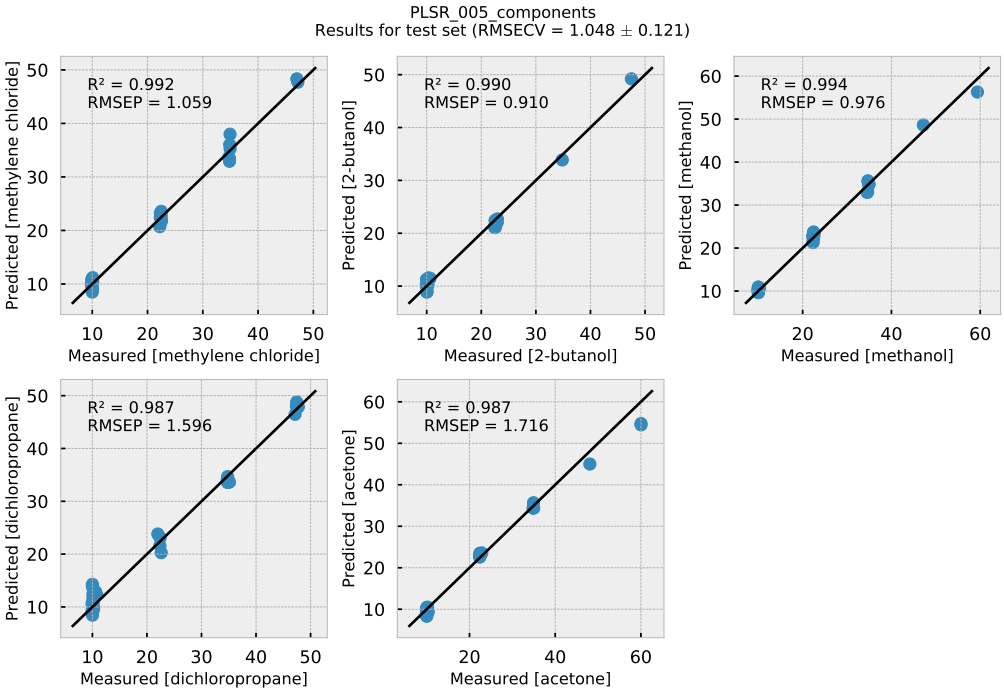

Figure 6: *Results for the test set when using 5 components for the PLSR model.*

of the leverage matrix ($\mathbf{H}$) for the individual points. The leverage matrix is found from the scores $\mathbf{T}$,

$$\mathbf{H} = \mathbf{T}(\mathbf{T}^\top\mathbf{T})^{-1}\mathbf{T}^\top.$$

The leverages for our samples are shown in Fig. 13. In this plot, the leverage is relatively high for the samples 34 and 43 and this strengthens our suspicion that these points are outliers. Presently, we do not have enough information to know if they are truly outliers or really interesting samples that are important for the model. We will thus keep all points in the following, but in a real-life situation we would investigate these two points further.

(iii) From the plot of the loadings, we find that the 5 genes with the largest expression for the cancer samples (along PC1) are 364, 291, 522, 510, and 509.

(iv) From the plot of the loadings, we find that the 5 genes with the lowest expression for the cancer samples (along PC1) are 1490, 538, 1116, 92, and 692.

(c) In the previous point (see (b)) we found that the genes 364, 291, 522, 510, and 509 were overexpressed in cancer samples and that 1490, 538, 1116, 92, and 692 were underexpressed. We expect that these genes can be used to distinguish between the two types of samples. In Fig. 14 we have investigated this further by plotting distributions of the samples for these 10 genes. Motivated by this figure, we select some pairs of genes which seem to separate the classes best, (1116, 509), (1490, 509), (92, 403), and (92,

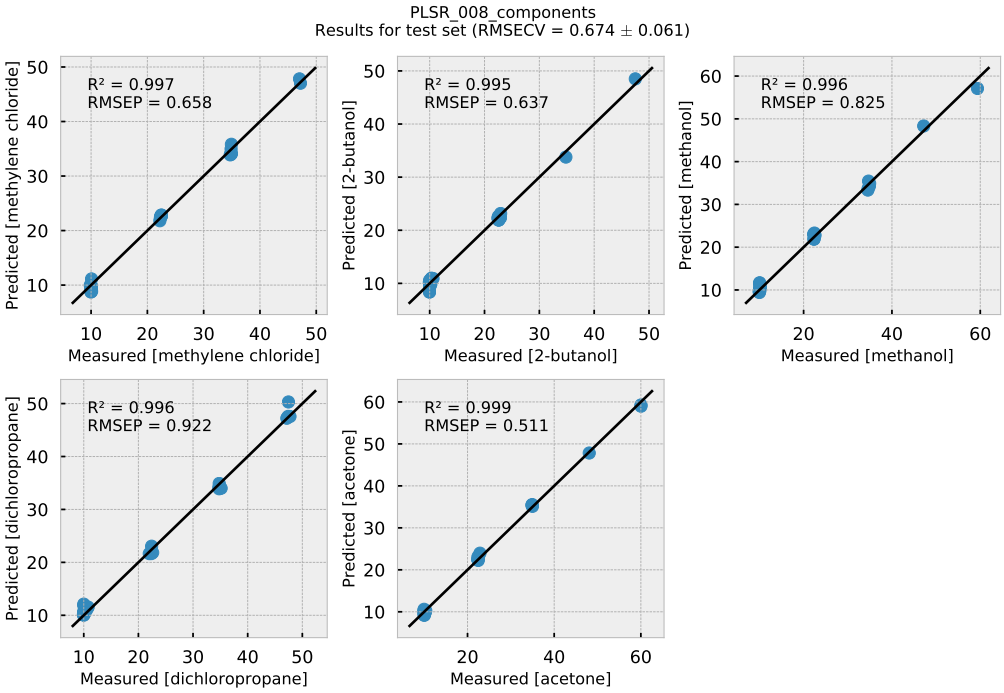

Figure 7: *Results for the test set when using 8 components for the PLSR model.*

509), and plot the gene expressions for the samples in Figs. 15–18. We see from these figures that we can largely separate between the two classes using the gene expressions for just two genes, for instance, 92 and 509.

(d) The trained classifier has an accuracy of 0.82, precision of 1 and recall of 0.67, for the test set. The confusion matrix for the training set is shown in Fig. 19. The corresponding decision tree is shown in Fig. 20. In this case, the decision tree picks out two of the genes we identified in point (c) above, namely 92 and 403. Note that many different combinations will give an equally good classification, and the one you find will depend on the random numbers used, for instance when splitting the data into training and test sets.

The decision boundaries are shown in Fig. 21. We can here identify the two false negatives (sample 16 and sample 26) that the classifier is making. In the next point, we will see if the model improves when we try a random forest.

(e) The trained random forest classifier has an accuracy of 0.91, precision of 1 and recall of 0.83, for the test set. The confusion matrix for the training set is shown in Fig. 22. We see that this classifier performs slightly better than the decision tree. But we note that we have a relatively small set of samples here. Still, we conclude that we to a large extent can distinguish between normal and cancer samples based on the measured gene expressions.

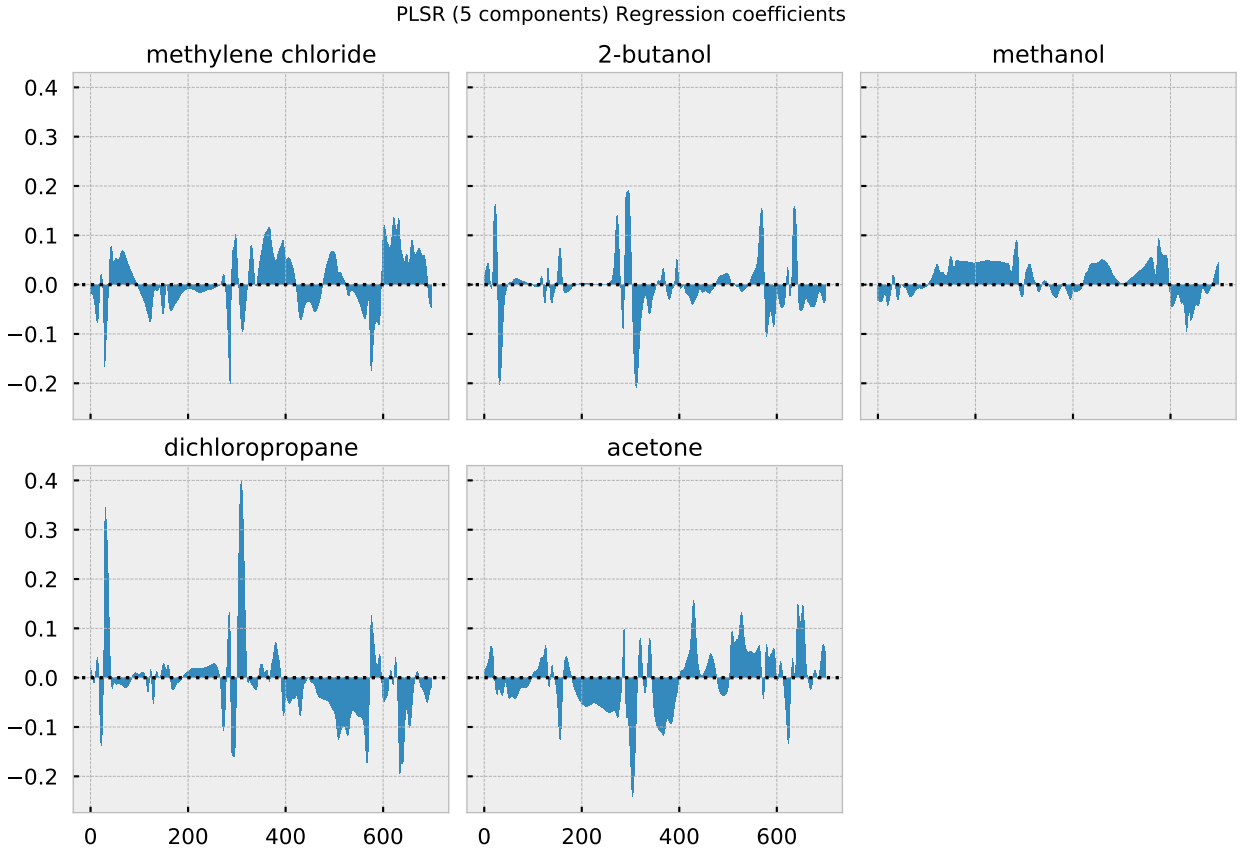

Figure 8: *Regression coefficients for the PLSR model with* 5 *components.*

The false-negative sample is in this case sample 16. It could be worthwhile to investigate this sample in more detail to see if there is anything special about it.

The variable importance is shown in Fig. 23. From this figure, we see that gene 92 is apparently the most important gene for the classification. Further, we see some other genes labeled as important that we did not single out when performing the PCA. We note that their importance is relatively similar, so when we constrain the trees to a maximum depth of 2, many possibilities will give classifications of comparable quality.

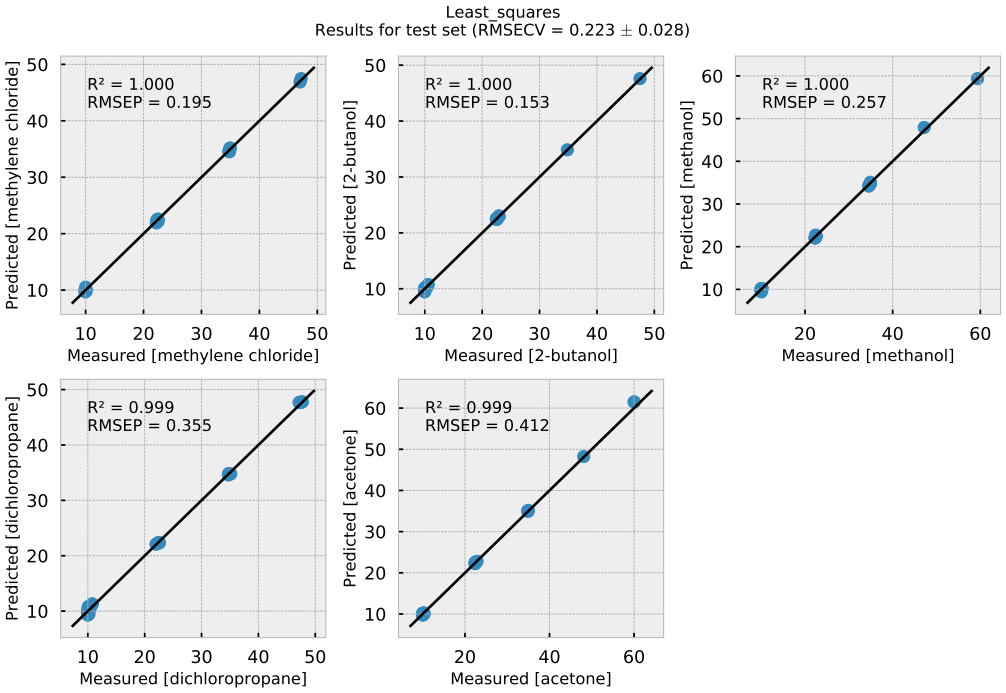

Figure 9: *Results for the test set when using a least-squares model.*

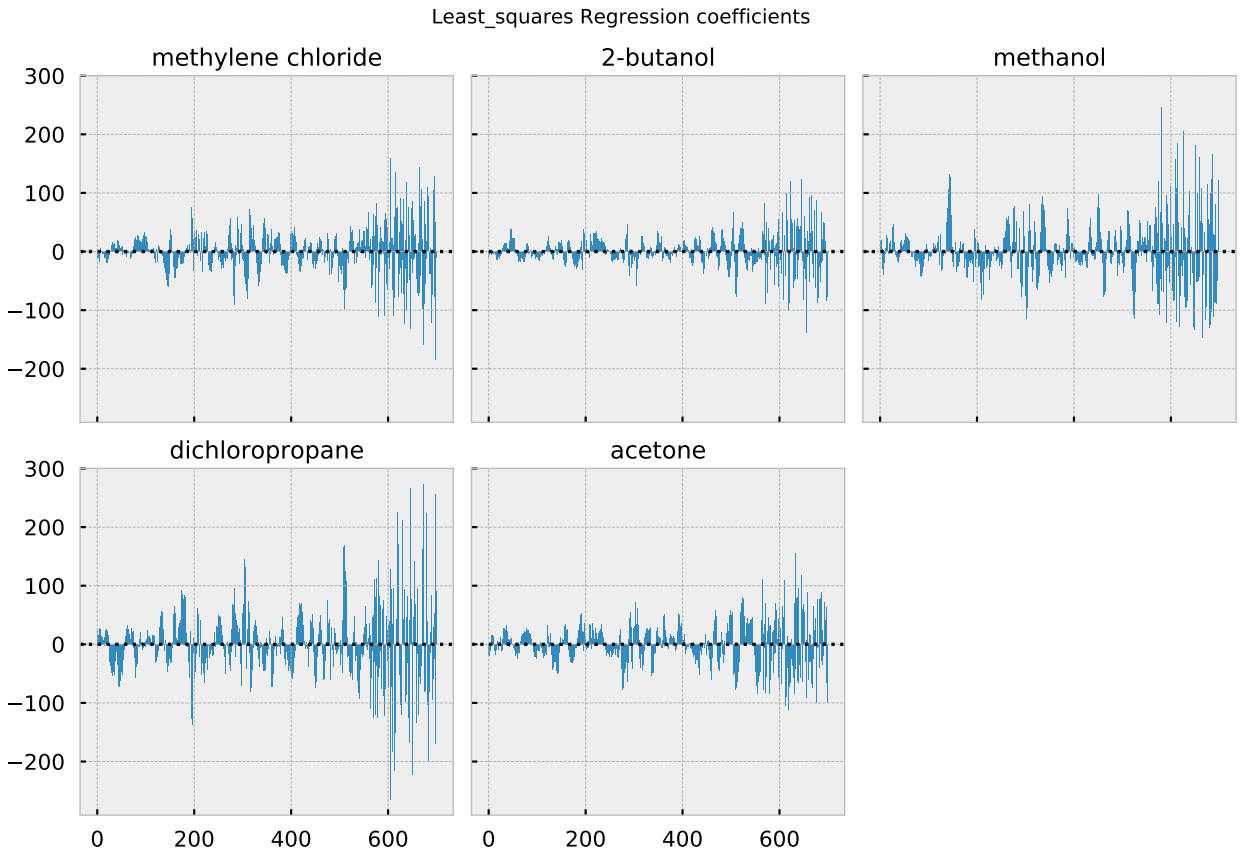

Figure 10: *Regression coefficients for the least-squares model.*

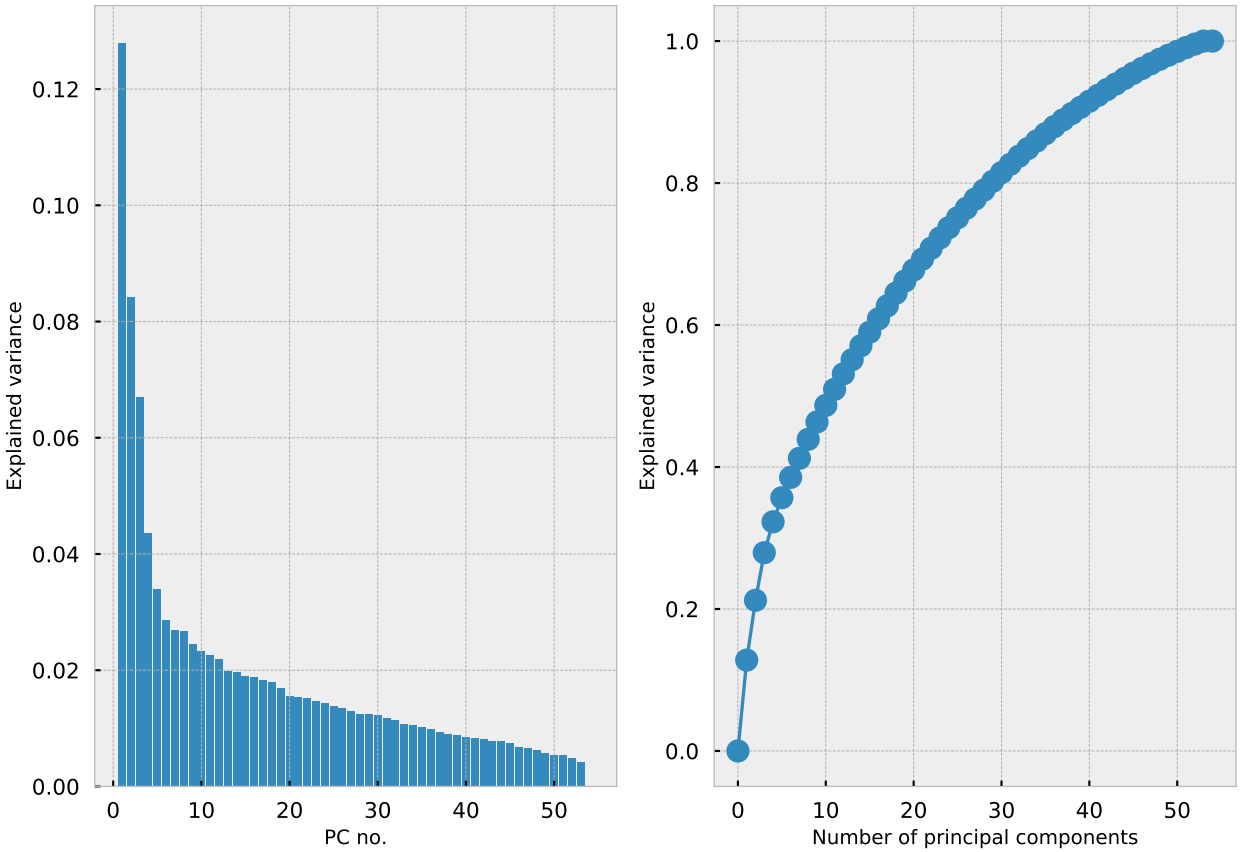

Figure 11: *Explained variance as a function of the number of principal components.*

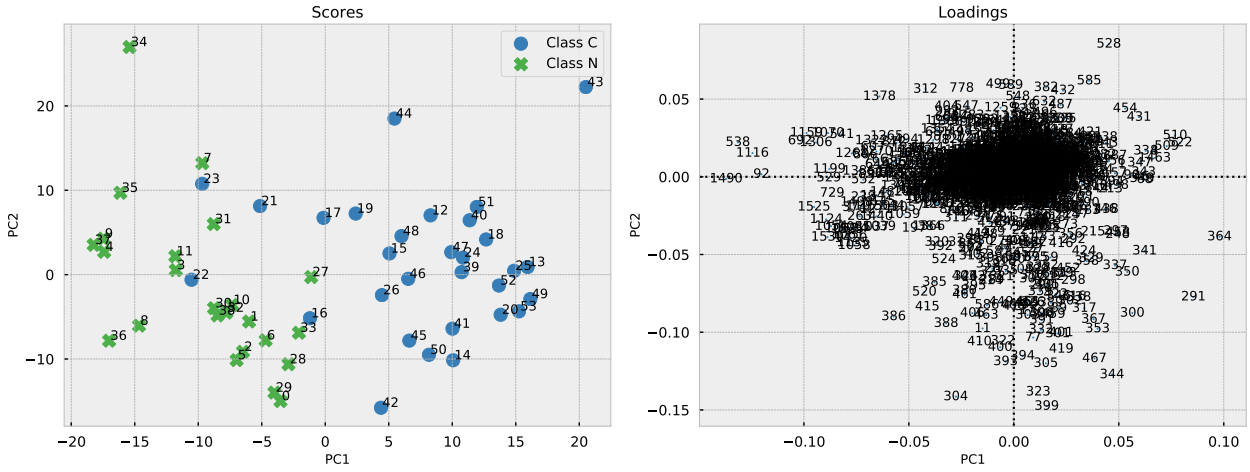

Figure 12: *Scores and loadings for principal component 1 and 2. The samples have been colored according to their label as normal ("N") samples or cancer ("C") samples.*

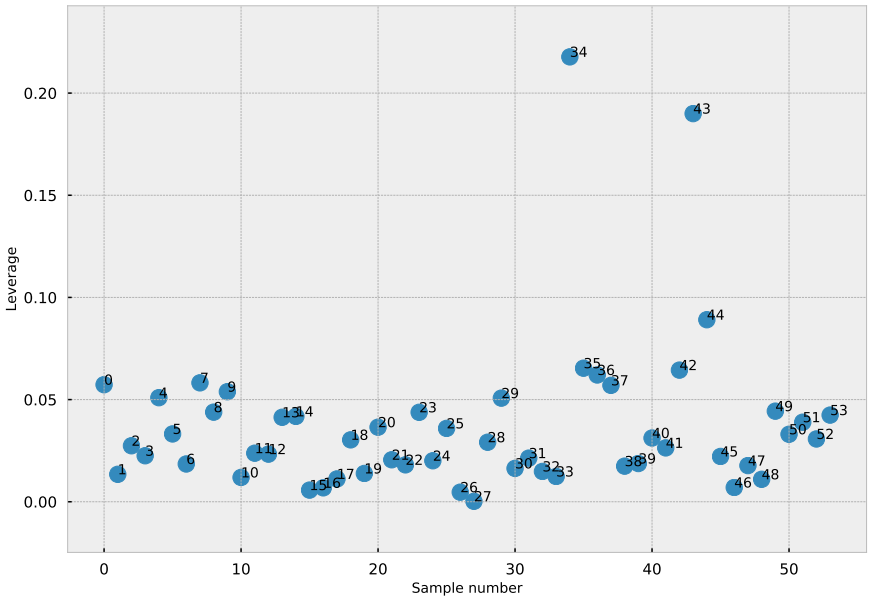

Figure 13: *Leverage for the individual samples in a PCA model with two components.*

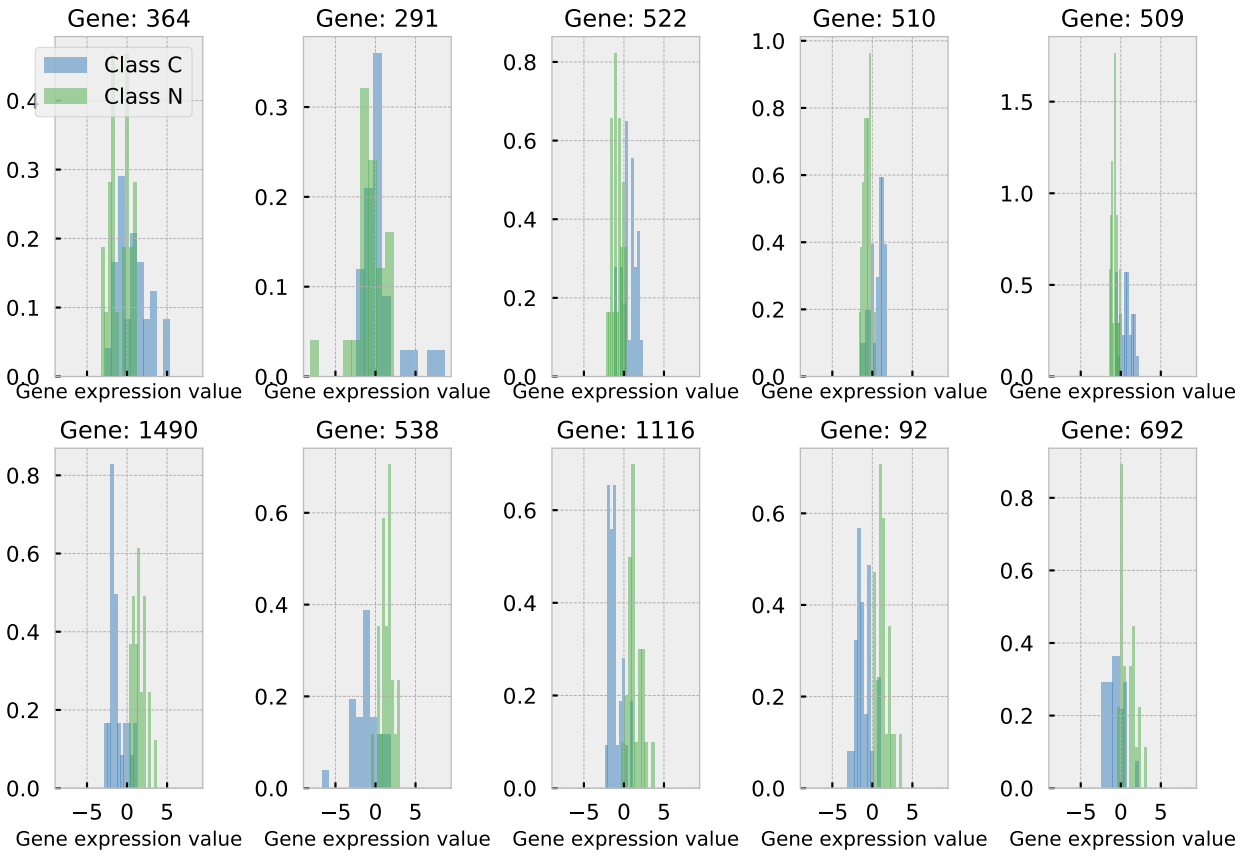

Figure 14: *Distribution of samples, labeled as normal ("N") or cancer ("C") samples for a selection of genes.*

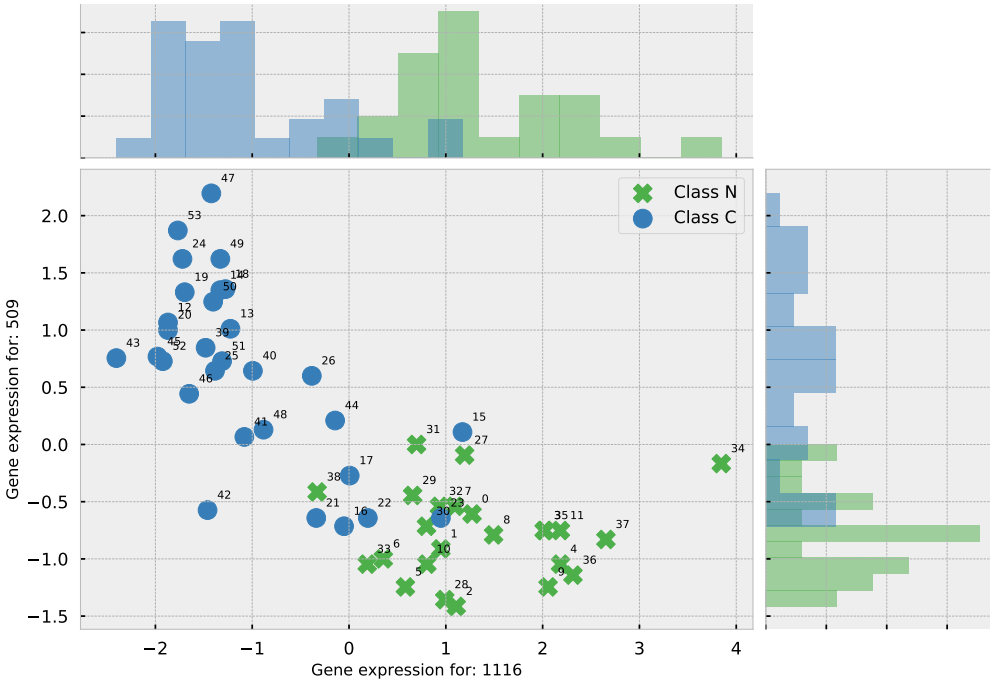

Figure 15: *Distribution of samples, labeled as normal ("N") or cancer ("C") samples for the genes 1116 and 509.*

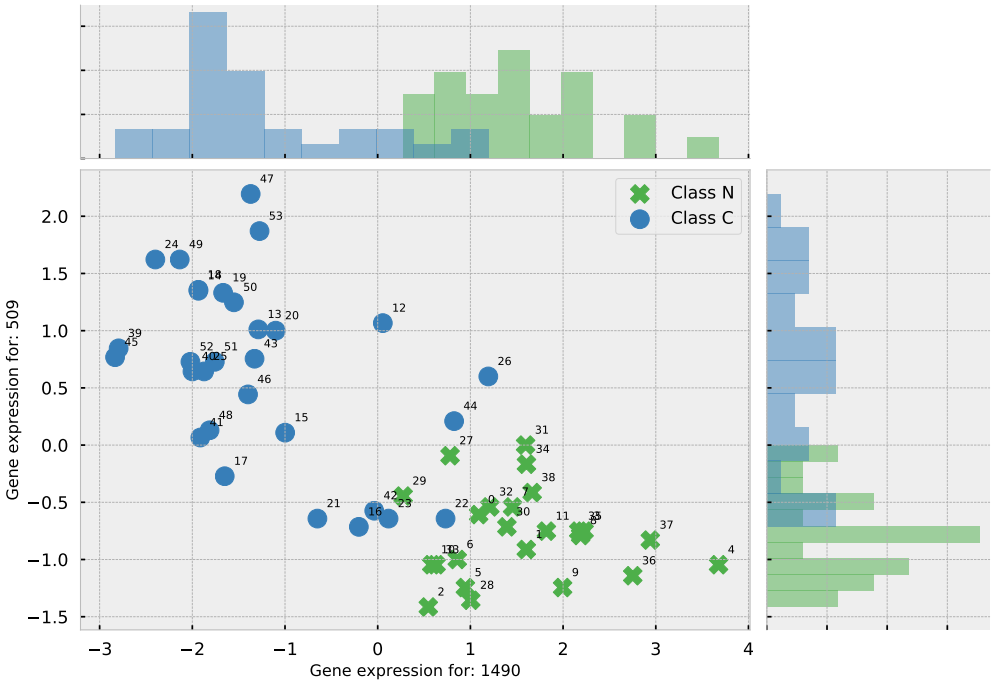

Figure 16: *Distribution of samples, labeled as normal ("N") or cancer ("C") samples for the genes 1490 and 509.*

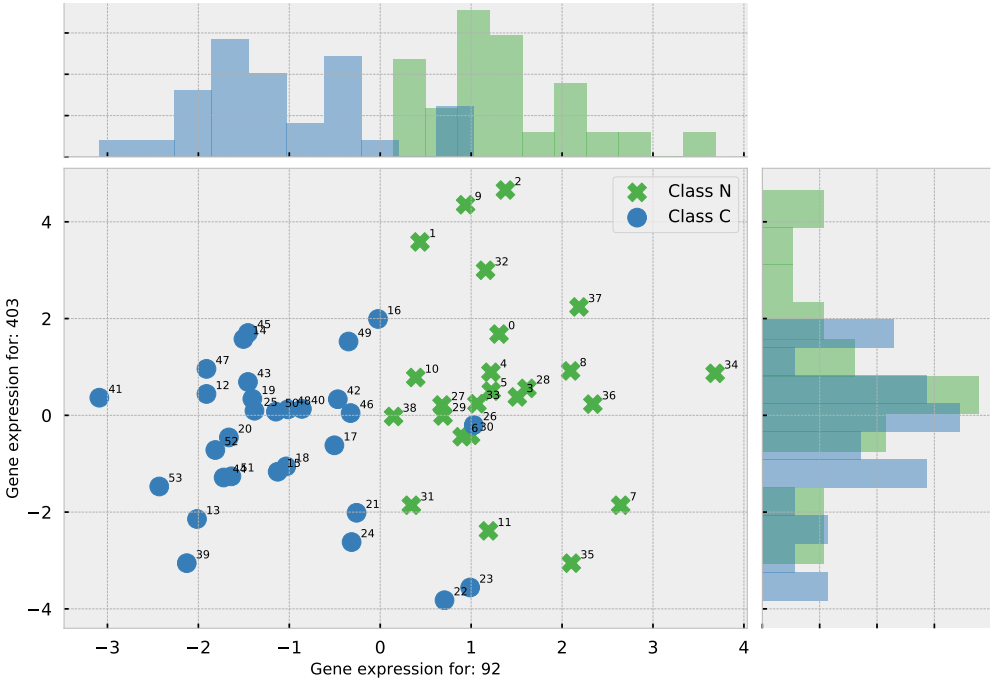

Figure 17: *Distribution of samples, labeled as normal ("N") or cancer ("C") samples for the genes 92 and 403.*

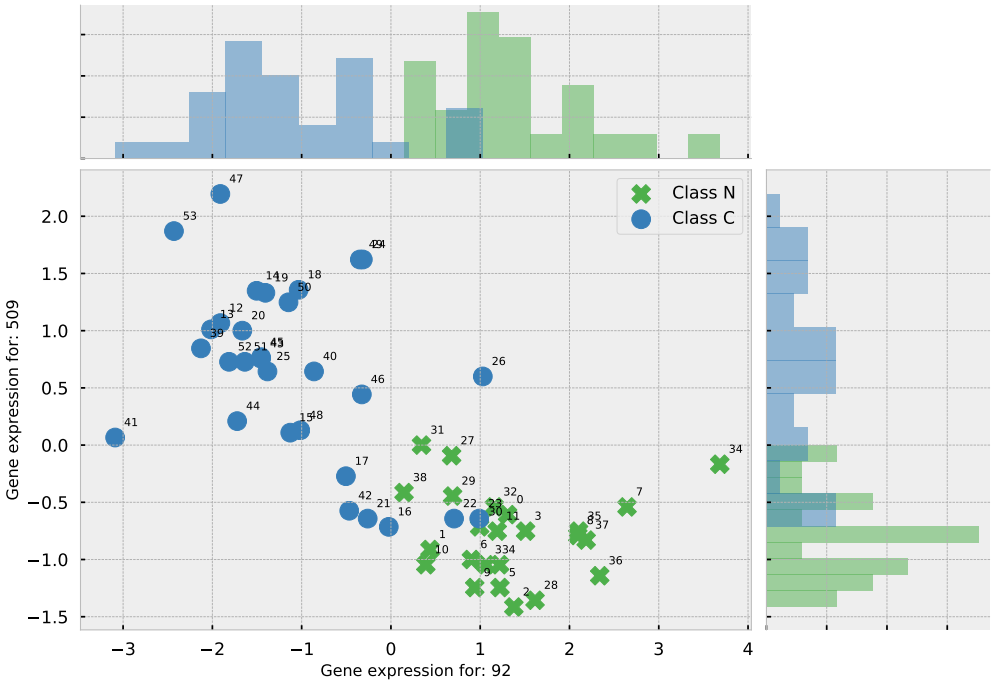

Figure 18: *Distribution of samples, labeled as normal ("N") or cancer ("C") samples for the genes 92 and 509.*

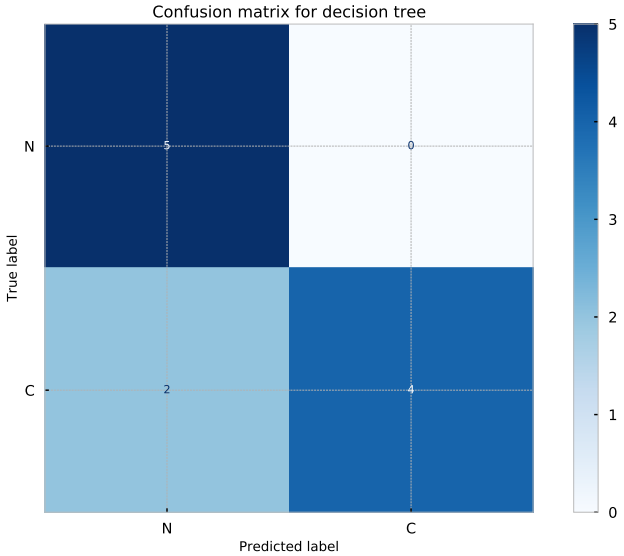

Figure 19: *Confusion matrix for the decision tree classifier, when applied to the test set.*

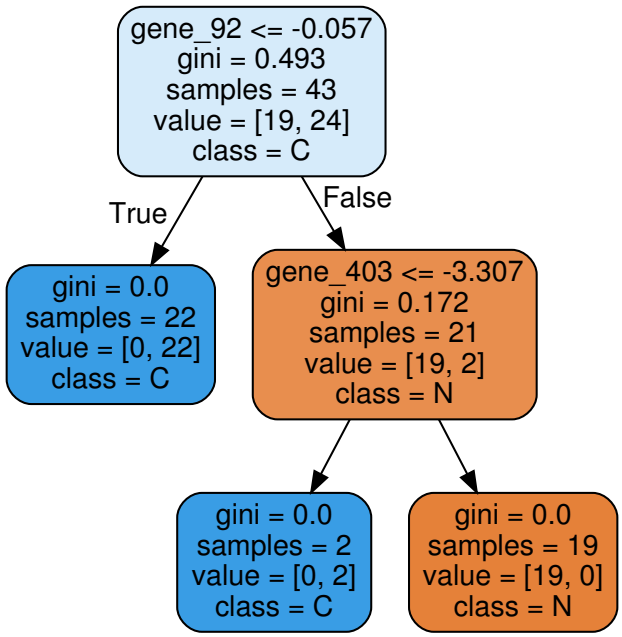

Figure 20: *Decision tree obtained from training a decision tree classifier.*

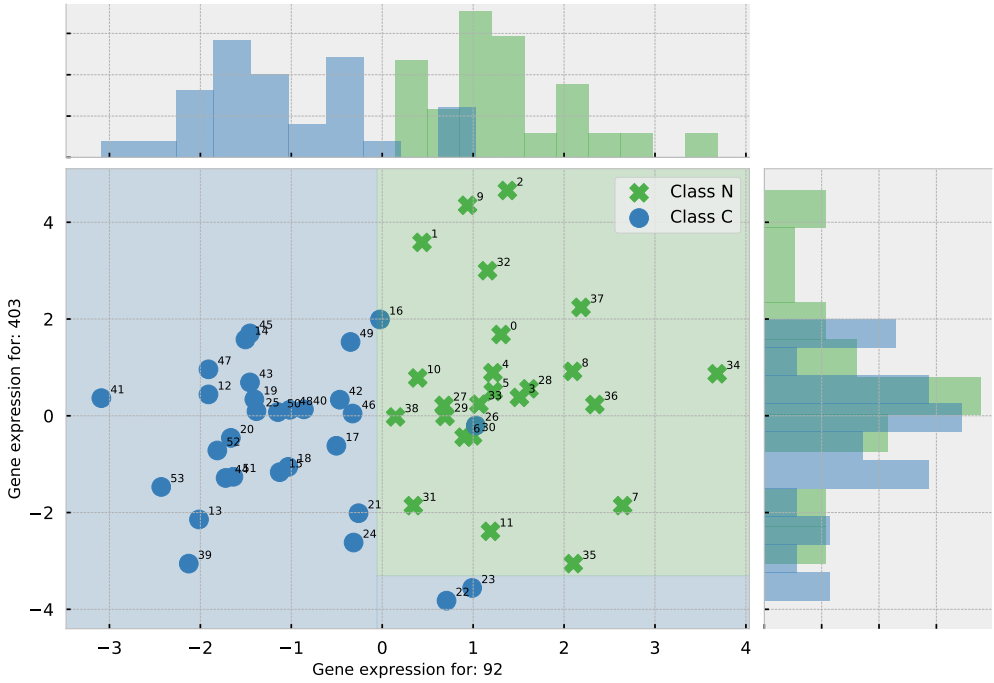

Figure 21: *Decision boundaries for the decision tree classifier.*

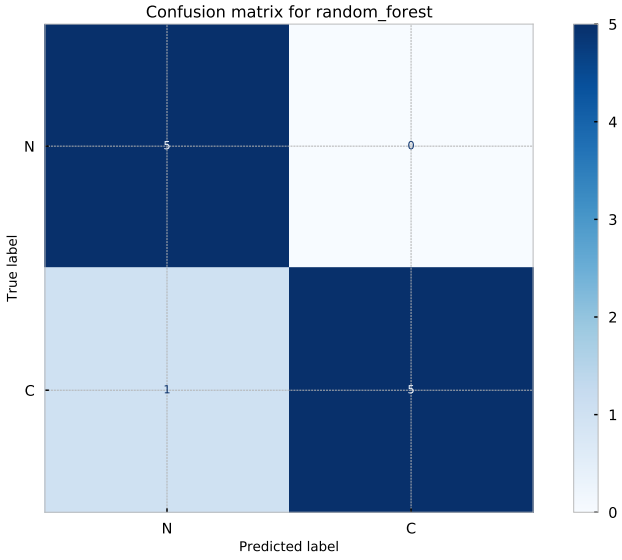

Figure 22: *Confusion matrix for the random forest classifier, when applied to the test set.*

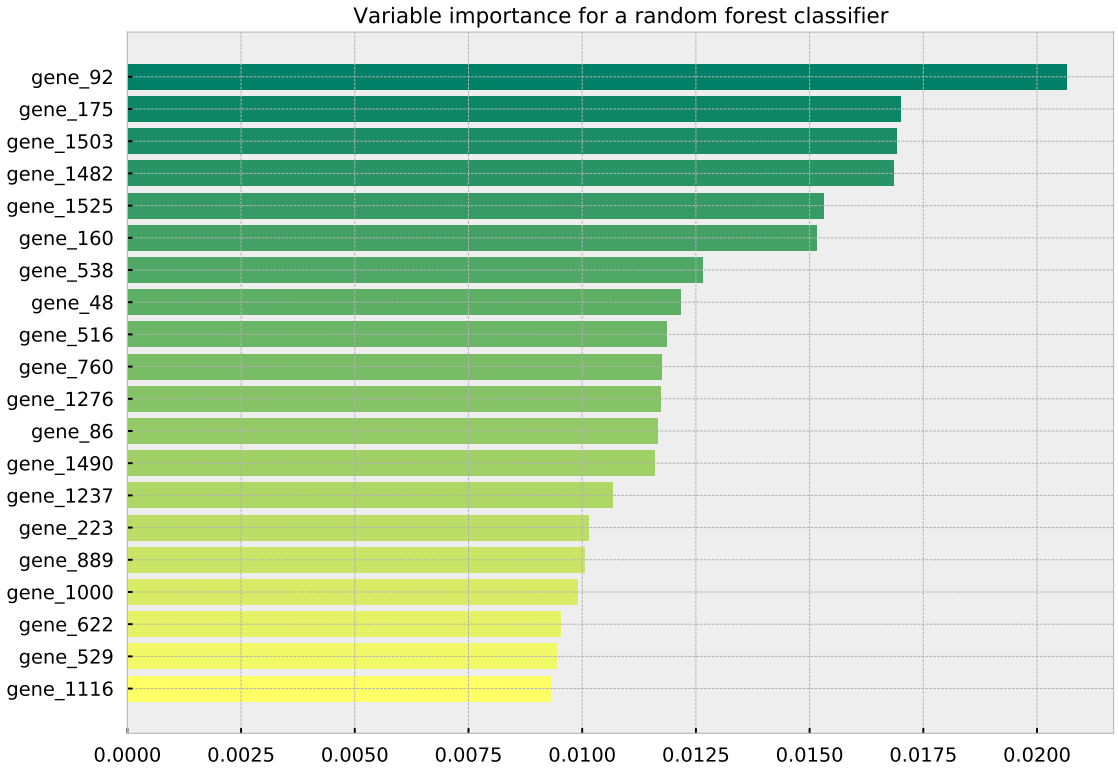

Figure 23: *Variable importance for the 20 most important variables in the random forest classifier.*

Python solutions

```
"""PLSR solution to the Windig data set."""
import pandas as pd
from matplotlib import pyplot as plt
import matplotlib as mpl
from sklearn.cross_decomposition import PLSRegression
from sklearn.model_selection import (
    train_test_split,
    cross_val_score,
)
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.linear_model import LinearRegression
import numpy as np
from slugify import slugify  # For making safe file names.
plt.style.use(['seaborn-talk', 'bmh'])
mpl.rcParams["savefig.format"] = 'pdf'


SOLVENTS = ['methylene chloride', '2-butanol', 'methanol',
            'dichloropropane', 'acetone']


def add_xy_line(axi, **kwargs):
    """Add a y=x line to the given axes.

    Parameters
    ----------
    axi : object like :class:`matplotlib.axes.Axes`
        The axis to add the y=x line to.
    **kwargs : dict, optional
        Additional arguments passed to the plotting method.

    Returns
    -------
    line : object like :class:`matplotlib.lines.Line2D`
        The created y=x line.

    """
    lim_min = np.min([axi.get_xlim(), axi.get_ylim()])
    lim_max = np.max([axi.get_xlim(), axi.get_ylim()])
    line, = axi.plot([lim_min, lim_max], [lim_min, lim_max], **kwargs)
    return line


def score_prediction(Y_true, Y_hat):
    """Obtain some metrics for prediction."""
    n_solvents = Y_true.shape[1]
    metrics = {
        'r2_solvent': [],
```

```
        'rmse_solvent': []
    }
    metrics['r2_overall'] = r2_score(Y_true, Y_hat)
    metrics['rmse_overall'] = np.sqrt(
        mean_squared_error(Y_true, Y_hat)
    )
    for i in range(n_solvents):
        rscore = r2_score(Y_true[:, i], Y_hat[:, i])
        rmse = np.sqrt(mean_squared_error(Y_true[:, i], Y_hat[:, i]))
        metrics['r2_solvent'].append(rscore)
        metrics['rmse_solvent'].append(rmse)
    return metrics


def plot_coefficients(model, model_name='PLS', transpose=False):
    """Plot coefficients for a model in a bar plot."""
    B = model.coef_
    if transpose:
        B = B.T
    fig1, axes1 = plt.subplots(constrained_layout=True, ncols=3, nrows=2,
                               sharex=True, sharey=True)
    fig1.suptitle('{} Regression coefficients'.format(model_name))
    axes1 = axes1.flatten()
    for i, axi in enumerate(axes1):
        try:
            coeffs = B[:, i]
            pos = range(len(coeffs))
            axi.bar(pos, coeffs, width=1)
            axi.axhline(y=0, ls=':', color='k')
            axi.set_title(SOLVENTS[i])
        except IndexError:
            axi.axis('off')
    fig1.savefig('nir_{}_regression_coefficients'.format(slugify(model_name)),
                 bbox_inches='tight')


def plot_compare_results(components, results):
    """Plot a comparison of the PLSR results."""
    fig1, (ax1, ax2) = plt.subplots(constrained_layout=True, ncols=2,
                                    sharex=True, figsize=(12, 6))
    rmsecv = [result[3] for result in results]
    rmsecv_std = [result[4] for result in results]
    ax1.errorbar(components, rmsecv, yerr=rmsecv_std,
                 marker='o', markersize=12)
    ax1.set_xlabel('Number of PLS components')
    ax1.set_ylabel('Overall RMSECV')
    rmsep = [result[1]['rmse_overall'] for result in results]
    ax2.plot(components, rmsep, marker='o', markersize=12)
    ax2.set_xlabel('Number of PLS components')
    ax2.set_ylabel('Overall RMSEP')
```

```
    fig1.savefig('nir_compare_plsr_rmsecv', bbox_inches='tight')
    plt.close(fig1)


def train_model(model, X_train, X_test, Y_train, Y_test, model_name='PLSR'):
    """Run PLSR and obtain results."""

    model.fit(X_train, Y_train)
    Y_train_hat = model.predict(X_train)
    Y_hat = model.predict(X_test)

    cvscore = cross_val_score(model, X_train, Y_train,
                              scoring='neg_mean_squared_error',
                              cv=5)
    cvscore = np.sqrt(-cvscore)
    rmsecv = cvscore.mean()
    rmsecv_std = cvscore.std()

    metrics_calibration = score_prediction(Y_train, Y_train_hat)
    metrics_test = score_prediction(Y_test, Y_hat)

    title = (
        '{}\n'
        r'Results for {{}} (RMSECV = {:.3f} $\pm$ {:.3f})'
    ).format(model_name, rmsecv, rmsecv_std)

    fig1, axes1 = plt.subplots(constrained_layout=True, nrows=2, ncols=3)
    fig1.suptitle(title.format('training set'))
    fig2, axes2 = plt.subplots(constrained_layout=True, nrows=2, ncols=3)
    fig2.suptitle(title.format('test set'))
    axes1 = axes1.flatten()
    axes2 = axes2.flatten()

    for i, (axi, axj) in enumerate(zip(axes1, axes2)):
        try:
            axi.scatter(Y_train[:, i], Y_train_hat[:, i])
            axi.set_xlabel('Measured [{}]'.format(SOLVENTS[i]))
            axi.set_ylabel('Predicted [{}]'.format(SOLVENTS[i]))
            add_xy_line(axi, ls='-', color='k')
            text = 'R² = {:.3f}\nRMSEC = {:.3f}'.format(
                metrics_calibration['r2_solvent'][i],
                metrics_calibration['rmse_solvent'][i],
            )
            axi.text(0.1, 0.8, text, transform=axi.transAxes,
                     fontsize='large')

            axj.scatter(Y_test[:, i], Y_hat[:, i])
            axj.set_xlabel('Measured [{}]'.format(SOLVENTS[i]))
            axj.set_ylabel('Predicted [{}]'.format(SOLVENTS[i]))
            add_xy_line(axj, ls='-', color='k')
```

```
            text = 'R² = {:.3f}\nRMSEP = {:.3f}'.format(
                metrics_test['r2_solvent'][i],
                metrics_test['rmse_solvent'][i],
            )
            axj.text(0.1, 0.8, text, transform=axj.transAxes,
                     fontsize='large')
        except IndexError:
            axi.axis('off')
            axj.axis('off')
    fig1.savefig('nir_train_{}'.format(model_name),
                 bbox_inchec='tight')
    fig2.savefig('nir_test_{}'.format(model_name),
                 bbox_inchec='tight')
    plt.close(fig1)
    plt.close(fig2)
    return model, metrics_test, metrics_calibration, rmsecv, rmsecv_std


def main():
    """Load the data and run PLSR."""
    data = pd.read_csv('Data/windig.csv')
    X = data.filter(like='data', axis=1).values
    Y = data.filter(like='concentrations', axis=1).values

    X_train, X_test, Y_train, Y_test = train_test_split(
        X, Y, test_size=0.2, random_state=10, shuffle=True,
    )

    results = []

    components = range(1, 16)
    for i in components:
        print('Running PLSR for {} component(s)'.format(i))
        plsr = PLSRegression(n_components=i)
        result = train_model(plsr, X_train, X_test, Y_train, Y_test,
                             model_name='PLSR_{:03d}_components'.format(i))
        results.append(result)

    plot_compare_results(components, results)

    plsr = results[4][0]
    plot_coefficients(
        plsr,
        model_name='PLSR ({} components)'.format(plsr.n_components)
    )

    lsq = LinearRegression()
    train_model(lsq, X_train, X_test, Y_train, Y_test,
                model_name='Least_squares')
    plot_coefficients(lsq, model_name='Least_squares', transpose=True)
```

```
if __name__ == '__main__':
    main()
```

Listing 1: *Python code for exercise 9.*

```
"""Solution for the ovo data set."""
import random
import pandas as pd
from matplotlib import pyplot as plt
import matplotlib as mpl
from matplotlib.cm import get_cmap
from cycler import cycler
from matplotlib.patches import Rectangle
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from sklearn.metrics import (
    accuracy_score,
    f1_score,
    confusion_matrix,
    precision_score,
    recall_score,
    plot_confusion_matrix,
)
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import (
    train_test_split,
)
import numpy as np
from slugify import slugify  # For making safe file names.
plt.style.use(['seaborn-talk', 'bmh'])
mpl.rcParams["savefig.format"] = 'pdf'


COLORS = {
    'C': '#377eb8',
    'N': '#4daf4a',
}


MARKERS = {
    'C': 'o',
    'N': 'X',
}


def plot_explained_variance(pca):
    """Plot the explained variance."""
    fig1, (ax1, ax2) = plt.subplots(constrained_layout=True, ncols=2)
```

```
    variance = [0] + list(np.cumsum(pca.explained_variance_ratio_))
    pos = list(range(0, len(variance)))
    ax2.plot(pos, variance, marker='o', markersize=14)
    ax2.set(
        xlabel='Number of principal components',
        ylabel='Explained variance'
    )
    pos = range(1, len(pca.explained_variance_ratio_) + 1)
    ax1.bar(pos, pca.explained_variance_ratio_)
    ax1.set(
        xlabel='PC no.',
        ylabel='Explained variance'
    )
    return fig1, ax1, ax2


def plot_2d_scores(scores, component1, component2, class_data=None,
                   loadings=None):
    """Plot scores for two PC's"""
    if loadings is None:
        fig1, ax1 = plt.subplots(constrained_layout=True)
        ax2 = None
    else:
        fig1, (ax1, ax2) = plt.subplots(constrained_layout=True, ncols=2,
                                        figsize=(16, 6))
        ax1.set_title('Scores')
        ax2.set_title('Loadings')
    axes = (ax1, ax2)
    if class_data is None:
        ax1.scatter(scores[:, component1], scores[:, component2], s=150)
    else:
        for class_id in set(class_data):
            idx = np.where(class_data == class_id)[0]
            ax1.scatter(
                scores[idx, component1],
                scores[idx, component2],
                s=150,
                color=COLORS[class_id],
                marker=MARKERS[class_id],
                label='Class {}'.format(class_id)
            )
        ax1.legend()
    for i, score in enumerate(scores):
        ax1.annotate(i, (score[component1] + 0.1, score[component2] + 0.1),
                     fontsize='large')
    ax1.set(
        xlabel='PC{}'.format(component1 + 1),
        ylabel='PC{}'.format(component2 + 1)
    )
    if loadings is not None:
```

```
        ax2.set(
            xlabel='PC{}'.format(component1 + 1),
            ylabel='PC{}'.format(component2 + 1)
        )
        ax2.axhline(y=0, ls=':', color='k')
        ax2.axvline(x=0, ls=':', color='k')
        load1 = loadings[component1, :]
        load2 = loadings[component2, :]
        idx = range(len(load1))
        ax2.scatter(load1, load2, s=1)
        for i in idx:
            ax2.annotate(i, (load1[i], load2[i]), fontsize='large',
                         ha='center', va='center')
        # Print out the five largest loadings coefficients along each axis:
        for load, comp in zip((load1, load2), (component1, component2)):
            idx1 = np.argsort(load)
            print('5 largest (positive) components for PC{}'.format(comp + 1))
            print('\t Gene: Loading')
            for i in idx1[-5:][::-1]:
                print('\t{:>5d}: {:>7.3f}'.format(i, load[i]))
            print('5 largest (negative) components for PC{}'.format(comp + 1))
            print('\t Gene: Loading')
            for i in idx1[:5]:
                print('\t{:>5d}: {:>7.3f}'.format(i, load[i]))
    return fig1, axes


def plot_1d_scores(scores, components, class_data=None):
    """Plot scores for two PC's"""
    fig1, axes = plt.subplots(constrained_layout=True, nrows=len(components))
    axes = axes.flatten()
    legend = True
    for componenti, axi in zip(components, axes):
        if class_data is None:
            xval = scores[:, componenti]
            axi.scatter(xval, np.zeros_like(xval), s=150)
        else:
            for class_id in set(class_data):
                idx = np.where(class_data == class_id)[0]
                xval = scores[idx, componenti]
                axi.scatter(
                    xval,
                    np.zeros_like(xval),
                    s=150,
                    color=COLORS[class_id],
                    marker=MARKERS[class_id],
                    label='Class {}'.format(class_id)
                )
            if legend:
                axi.legend()
```

```
                legend = False
        axi.set_ylim(-0.05, 0.05)
        axi.axhline(y=0, color='k')
        axi.set_xlabel('PC{}'.format(componenti + 1))
        for loc in ('left', 'right', 'top'):
            axi.spines[loc].set_visible(False)
        axi.get_yaxis().set_visible(False)
        axi.spines['bottom'].set_position('zero')
    return fig1, axes


def create_pca(n_components, xdata, class_data):
    """Do PCA for the given number of components."""
    pca = PCA(n_components=n_components)
    scores = pca.fit_transform(xdata)
    loadings = pca.components_
    fig1, _, _ = plot_explained_variance(pca)
    fig1.savefig('ovo_pca_explained_variance', bbox_inches='tight')
    plt.close(fig1)
    fig2, _ = plot_2d_scores(scores, 0, 1, class_data=class_data,
                             loadings=loadings)
    fig2.savefig('ovo_pca_2d_components_1_2', bbox_inches='tight')
    plt.close(fig2)
    fig3, _ = plot_1d_scores(scores, [0, 1, 2, 3, 4], class_data=class_data)
    fig3.savefig('ovo_pca_1d_components', bbox_inches='tight')
    plt.close(fig3)


def _histogram2d_style(axi, *spines, xlim=None, ylim=None):
    """Style the given axis for 1D histograms.

    Parameters
    ----------
    axi : object like :class:`matplotlib.axes.Axes`
        The axis we will style.
    spines : list of strings
        The spines we will hide.
    xlim : tuple of floats, optional
        The limits for the x-axis.
    ylim : tuple of floats, optional
        The limits for the y-axis.

    """
    axi.yaxis.set_ticklabels([])
    axi.xaxis.set_ticklabels([])
    for spine in spines:
        axi.spines[spine].set_visible(False)
    axi.set(xlabel=None, ylabel=None)
    if xlim is not None:
        axi.set_xlim(xlim)
```

```
    if ylim is not None:
        axi.set_ylim(ylim)


def make_pair_histograms(xdata, classes, gene1, gene2, close=True,
                         annotate=True):
    """Plot histograms for pairs of genes."""
    fig = plt.figure(constrained_layout=True)
    grid = fig.add_gridspec(4, 4)
    ax_l = fig.add_subplot(grid[0, -1])
    ax_l.axis('off')
    ax_x = fig.add_subplot(grid[0, :-1])
    ax_y = fig.add_subplot(grid[1:, -1])
    axi = fig.add_subplot(grid[1:, :-1])

    for class_id in set(classes):
        idx = np.where(classes == class_id)[0]
        ax_x.hist(xdata[idx, gene1], color=COLORS[class_id],
                  alpha=0.5, density=True)
        ax_y.hist(xdata[idx, gene2], color=COLORS[class_id],
                  alpha=0.5, density=True, orientation='horizontal')
        axi.scatter(xdata[idx, gene1], xdata[idx, gene2],
                    color=COLORS[class_id],
                    s=200, marker=MARKERS[class_id],
                    label='Class {}'.format(class_id))
    if annotate:
        for i, xdatai in enumerate(xdata):
            axi.annotate(i, (xdatai[gene1] + 0.1, xdatai[gene2] + 0.1),
                         fontsize='small')
    axi.legend()
    axi.set_xlabel('Gene expression for: {}'.format(gene1))
    axi.set_ylabel('Gene expression for: {}'.format(gene2))

    _histogram2d_style(ax_x, 'top', 'right', xlim=axi.get_xlim())
    _histogram2d_style(ax_y, 'bottom', 'right', ylim=axi.get_ylim())
    fig.savefig('ovo_histogram_pair_{}_{}'.format(gene1, gene2),
                bbox_inches='tight')
    if close:
        plt.close(fig)
        return None, None
    else:
        return fig, [axi, ax_x, ax_y, ax_l]


def make_single_histograms(xdata, classes, genes):
    """Make histograms for selected genes."""
    if (ngenes := len(genes)) <= 3:
        nrows = 1
    else:
        nrows = 2
```

```
    ncols = - (-ngenes // nrows)
    ncols = max(ncols, 1)
    fig1, axes = plt.subplots(constrained_layout=True,
                              nrows=nrows, ncols=ncols, sharex=True)
    axes = axes.flatten()
    legend = True
    for i, axi in enumerate(axes):
        try:
            gene = genes[i]
            for class_id in set(classes):
                idx = np.where(classes == class_id)[0]
                axi.hist(xdata[idx, gene], color=COLORS[class_id],
                         label='Class {}'.format(class_id), alpha=0.5,
                         bins=10, density=True)
            if legend:
                axi.legend()
                legend = False
            axi.set_title('Gene: {}'.format(gene))
            axi.set_xlabel('Gene expression value')
        except IndexError:
            axi.axis('off')
    fig1.savefig('ovo_histogram', bbox_inches='tight')
    plt.close(fig1)


def score_clf(clf, X, y_true, name='Classifier', class_names=None):
    """Test a classifier."""
    y_hat = clf.predict(X)
    accuracy = accuracy_score(y_true, y_hat)
    precision = precision_score(y_true, y_hat)
    recall = recall_score(y_true, y_hat)
    f1 = f1_score(y_true, y_hat)
    conf = confusion_matrix(y_true, y_hat)
    print('\nScores for:', name)
    print('Accuracy:', accuracy)
    print('Precision:', precision)
    print('Recall:', recall)
    print('F1', f1)
    print('Confusion matrix:', conf)
    if class_names is not None:
        fig = plot_confusion_matrix(
            clf, X, y_true,
            display_labels=class_names,
            cmap=plt.cm.Blues,
            values_format='d',
        )
        fig.ax_.set_title('Confusion matrix for {}'.format(name))
        fig.figure_.tight_layout()
        fig.figure_.savefig('ovo_confmat{}'.format(slugify(name)),
                            bbox_inches='tight')
```

```
        plt.close(fig.figure_)
    result = {'accuracy': accuracy, 'precision': precision, 'recall': recall,
              'f1': f1, 'confusion': conf}
    return result


def add_boundaries(axi, boundary):
    """Add decision boundaries to a plot."""
    xlim = axi.get_xlim()
    ylim = axi.get_ylim()

    width1 = boundary[0] - min(xlim)
    height1 = max(ylim) - min(ylim)
    rect1 = Rectangle((min(xlim), min(ylim)), width1, height1, alpha=0.2,
                      color=COLORS['C'])
    axi.add_artist(rect1)

    width2 = max(xlim) - boundary[0]
    height2 = boundary[1] - min(ylim)
    rect2 = Rectangle((boundary[0], min(ylim)), width2, height2, alpha=0.2,
                      color=COLORS['C'])
    axi.add_artist(rect2)

    width3 = max(xlim) - boundary[0]
    height3 = max(ylim) - boundary[1]
    rect3 = Rectangle(boundary, width3, height3, alpha=0.2, color=COLORS['N'])
    axi.add_artist(rect3)


def plot_variable_importance(variables, clf, nitems=20, color_map='summer'):
    """Plot variable importance for a random forest."""
    figi, axi = plt.subplots(constrained_layout=True)
    cmap = get_cmap(name=color_map)
    if nitems is None:
        nitems = len(variables)
    color_cycler = cycler(color=cmap(np.linspace(0, 1, nitems)))
    axi.set_prop_cycle(color_cycler)
    importance = clf.feature_importances_
    idx = np.argsort(importance)
    y_pos = []
    y_label = []
    for i, idxi in enumerate(idx[-nitems:]):
        y_pos.append(i)
        y_label.append(variables[idxi])
        axi.barh(i, importance[idxi], align='center')
    axi.set_yticks(y_pos)
    axi.set_yticklabels(y_label)
    axi.set_title('Variable importance for a random forest classifier')
    figi.savefig('ovo_variableimportance', bbox_inches='tight')
    return figi, axi
```

```
def plot_pca_leverage(xdata, n_components=2):
    """Make a influence plot to detect outliers."""
    pca = PCA(n_components=n_components)
    scores = pca.fit_transform(xdata)
    components = pca.components_
    mat_inv = np.linalg.inv(np.dot(scores.T, scores))
    leverage = np.dot(np.dot(scores, mat_inv), scores.T)
    diag1 = np.diag(leverage)
    fig, ax1 = plt.subplots(constrained_layout=True)
    ax1.scatter(range(len(diag1)), diag1, s=200)
    for i, diag1i in enumerate(diag1):
        ax1.annotate(i, (i, diag1i), fontsize='large')
    ax1.set(xlabel='Sample number', ylabel='Leverage')
    fig.savefig('ovo_leverage_{}'.format(n_components), bbox_inches='tight')


def main():
    """Load the data and run PLSR."""
    # First, set global random states to get reproducible results:
    np.random.seed(123)
    random.seed(123)
    data = pd.read_csv('Data/ovo.csv')
    classes = data['objlabels']
    X = data.filter(like='X.', axis=1).values
    X = scale(X, with_std=False)

    create_pca(None, X, classes)

    plot_pca_leverage(X, n_components=2)

    positive = [364, 291, 522, 510, 509]
    negative = [1490, 538, 1116, 92, 692]
    selected = positive + negative

    make_single_histograms(X, classes, selected)
    combinations = [(92, 509), (1490, 509), (1116, 509)]
    for (gene1, gene2) in combinations:
        make_pair_histograms(X, classes, gene1, gene2)

    tree0 = DecisionTreeClassifier(max_depth=2)
    rnd_tree = RandomForestClassifier(max_depth=2, n_estimators=500)

    y = [1 if i == 'C' else 0 for i in classes]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=10, stratify=y
    )
    tree0.fit(X_train, y_train)
    rnd_tree.fit(X_train, y_train)
```

```
    feature_names = ['gene_{}'.format(i) for i in range(X.shape[1])]
    export_graphviz(tree0, out_file='tree.dot',
                    feature_names=feature_names,
                    class_names=['N', 'C'],
                    rounded=True,
                    filled=True)
    score_clf(tree0, X_test, y_test, name='decision tree',
              class_names=['N', 'C'])

    fig, axes = make_pair_histograms(X, classes, 92, 403, close=False)
    add_boundaries(axes[0], [-0.057, -3.307])
    fig.savefig('ovo_decision_tree_92_403', bbox_inches='tight')
    plt.close(fig)

    score_clf(rnd_tree, X_test, y_test, name='random_forest',
              class_names=['N', 'C'])
    plot_variable_importance(feature_names, rnd_tree, nitems=20,
                             color_map='summer_r')


if __name__ == '__main__':
    main()
```

Listing 2: *Python code for exercise 9.*