Week_09

November 23, 2021

Lau Møller Andersen
November 23 2021

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from os.path import join

path = '/home/lau/Skrivebord/class_subject'
data = np.load(join(path, 'megmag_data.npy'))

print('Shape: ' + str(data.shape))
print('n measurements: ' + str(np.prod(data.shape)))
print('n observations: ' + str(data.shape[0]))
print('n features: ' + str(np.prod(data.shape[1:])))
```

```
Shape: (682, 102, 251)
n measurements: 17460564
n observations: 682
n features: 25602
```

[2]:
```
## import wine data
import pandas as pd
url = 'https://archive.ics.uci.edu/ml/' + \
        'machine-learning-databases/wine/wine.data'
df_wine = pd.read_csv(url, header=None)
```

[3]:
```
## default parameters
import matplotlib as mpl
mpl.rcParams['axes.labelsize'] = 18
mpl.rcParams['axes.titlesize'] = 24
mpl.rcParams['xtick.labelsize'] = 16
mpl.rcParams['ytick.labelsize'] = 16
mpl.rcParams['font.weight'] = 'bold'
```

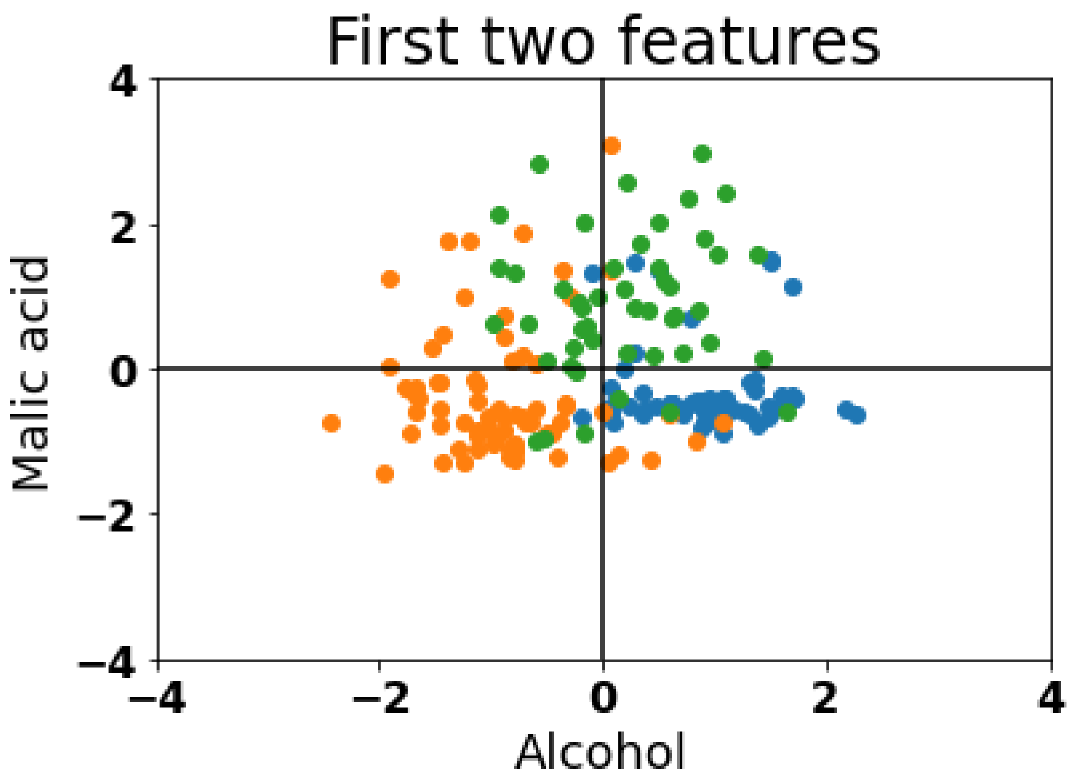
[4]:
```
# split into training and test data

from sklearn.model_selection import train_test_split
```

```python
from sklearn.preprocessing import StandardScaler

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values
print(X.shape)
print(np.unique(y))
```

```
(178, 13)
[1 2 3]
```

[5]:
```python
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.3, random_state=0)

sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.fit_transform(X_test)
X_std = sc.fit_transform(X)
```

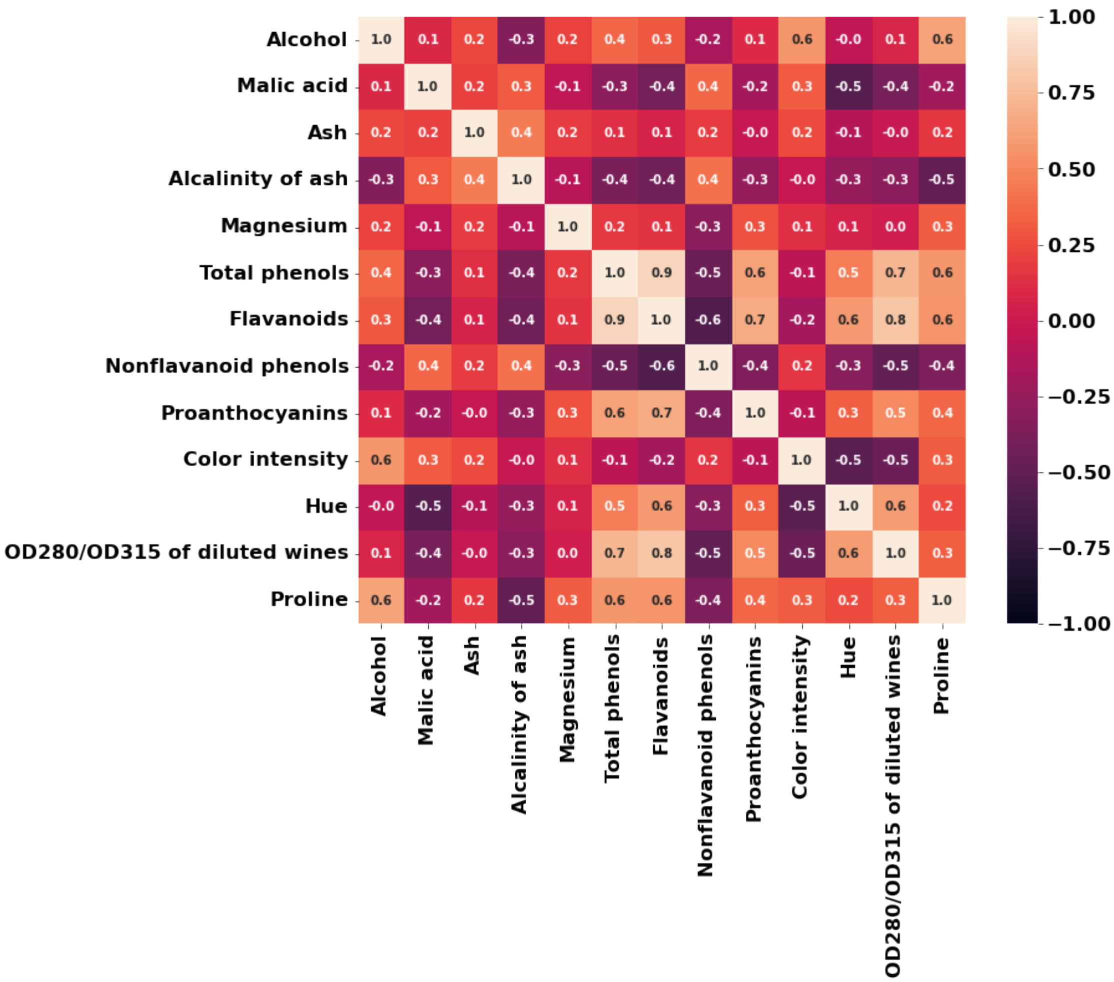
[6]:
```python
## plot data
import matplotlib.pyplot as plt
target_values = np.unique(y)
plt.figure()
for target_value in target_values:
    plt.plot(X_std[y == target_value, 0], X_std[y == target_value, 1], 'o')
plt.xlabel('Alcohol')
plt.ylabel('Malic acid')
plt.title('First two features')
plt.xlim(-4, 4)
plt.ylim(-4, 4)
plt.axvline(color='k')
plt.axhline(color='k')
plt.show()
```



[7]:
```
## covariance matrix
import numpy as np
import seaborn as sns
cov_mat = np.cov(X_train_std.T)
print(cov_mat.shape)
names = ['Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash',
         'Magnesium', 'Total phenols', 'Flavanoids',
         'Nonflavanoid phenols', 'Proanthocyanins',
         'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',
         'Proline']
plt.figure(figsize=(12, 9))
sns.heatmap(cov_mat, cbar=True, annot=True, square=True,
            yticklabels=names, xticklabels=names, fmt='.1f',
            vmin=-1.0, vmax=1.0)
```

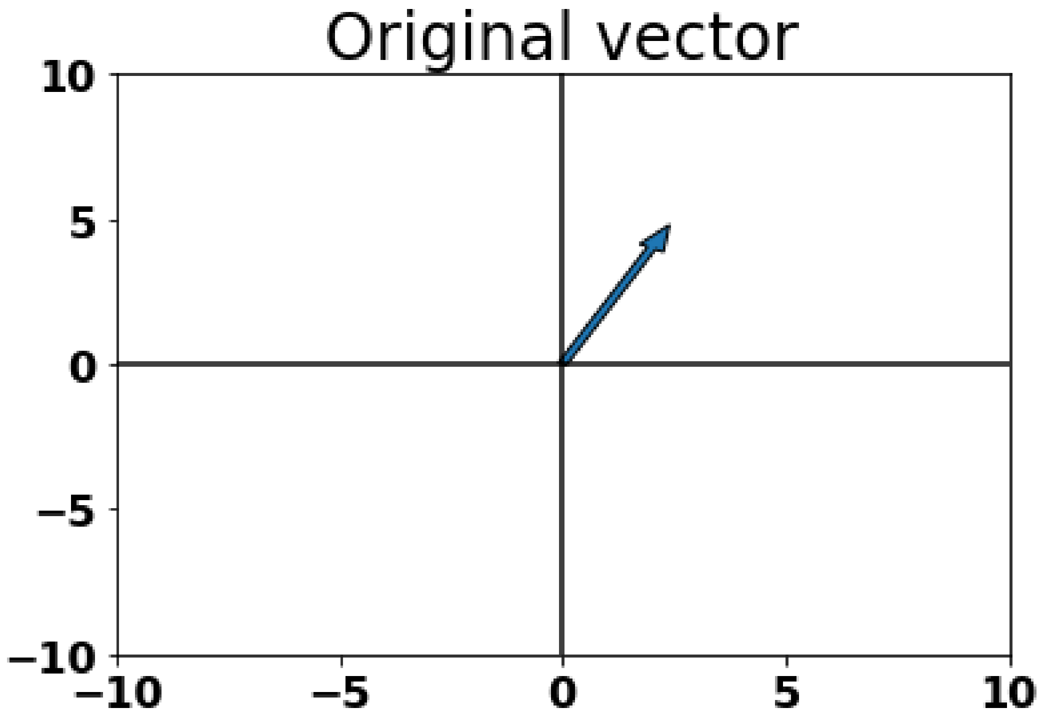
```
(13, 13)
```

[7]: <AxesSubplot:>



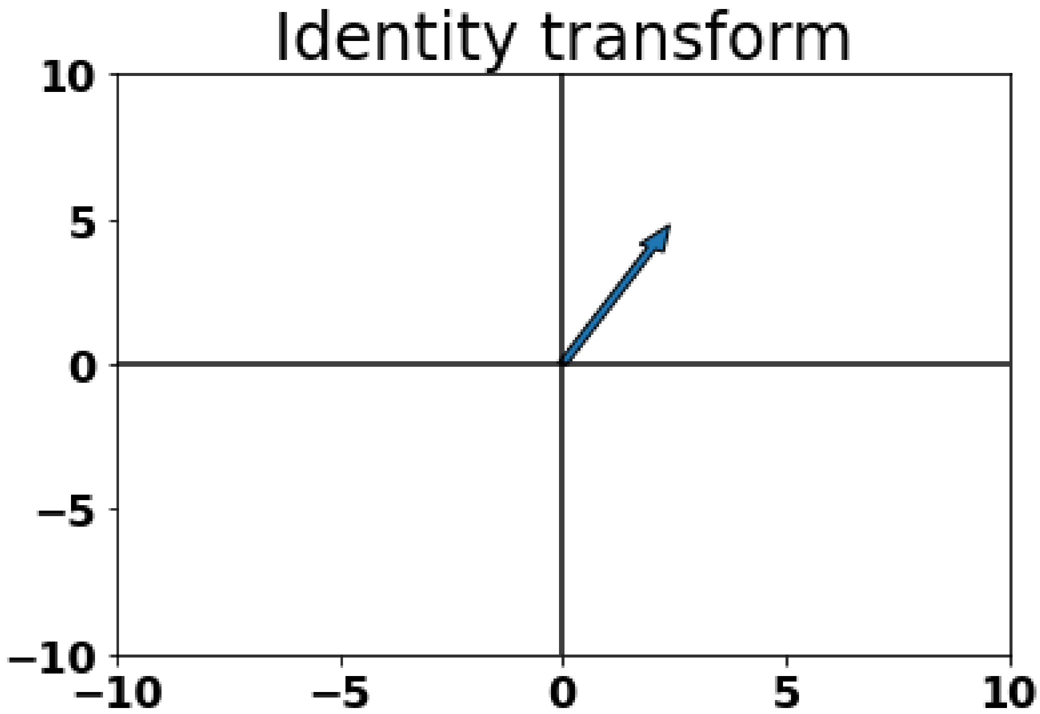
[8]:

```
## function for plotting vector

def plot_vector(v, title='', xmin=-10, xmax=10, ymin=-10, ymax=10):
    plt.figure()
    plt.arrow(0, 0, v[0], v[1], width=0.2)
    plt.xlim(xmin, xmax)
    plt.ylim(ymin, ymax)
    plt.axvline(color='k')
    plt.axhline(color='k')
    plt.title(title)
    plt.show()

## plot vector (2, 4)
v = (2, 4)
plot_vector(v, 'Original vector')
```



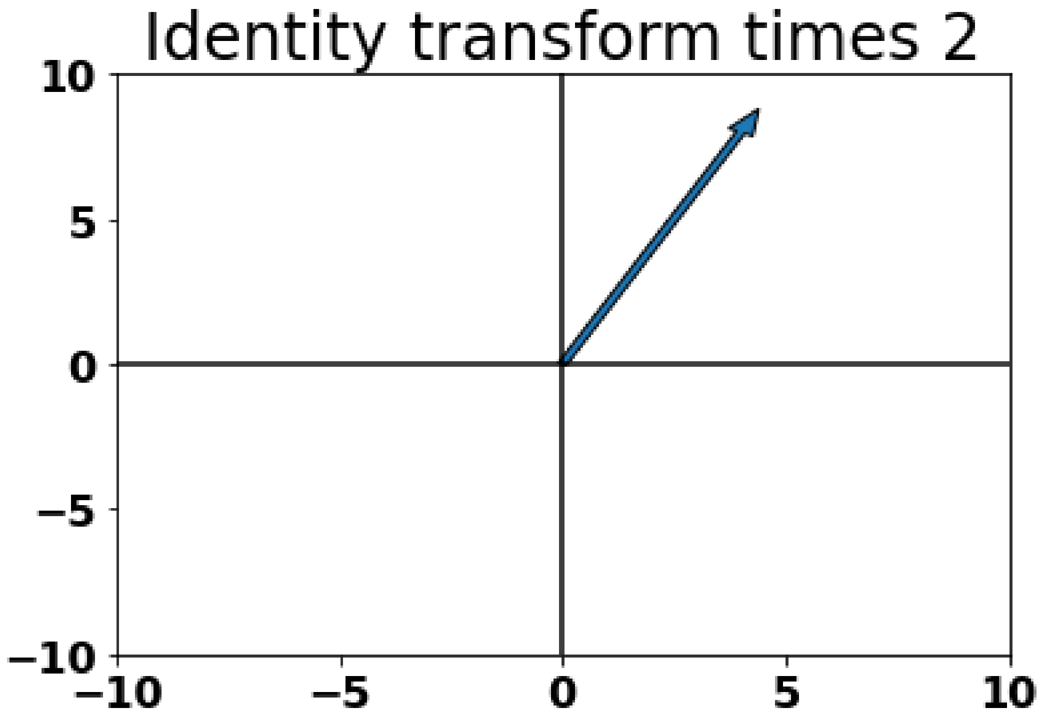
[9]:
```
## transform with identity matrix
A = np.identity(2)
print('A = \n' + str(A))
plot_vector(A @ v, 'Identity transform')
```

```
A =
[[1. 0.]
 [0. 1.]]
```



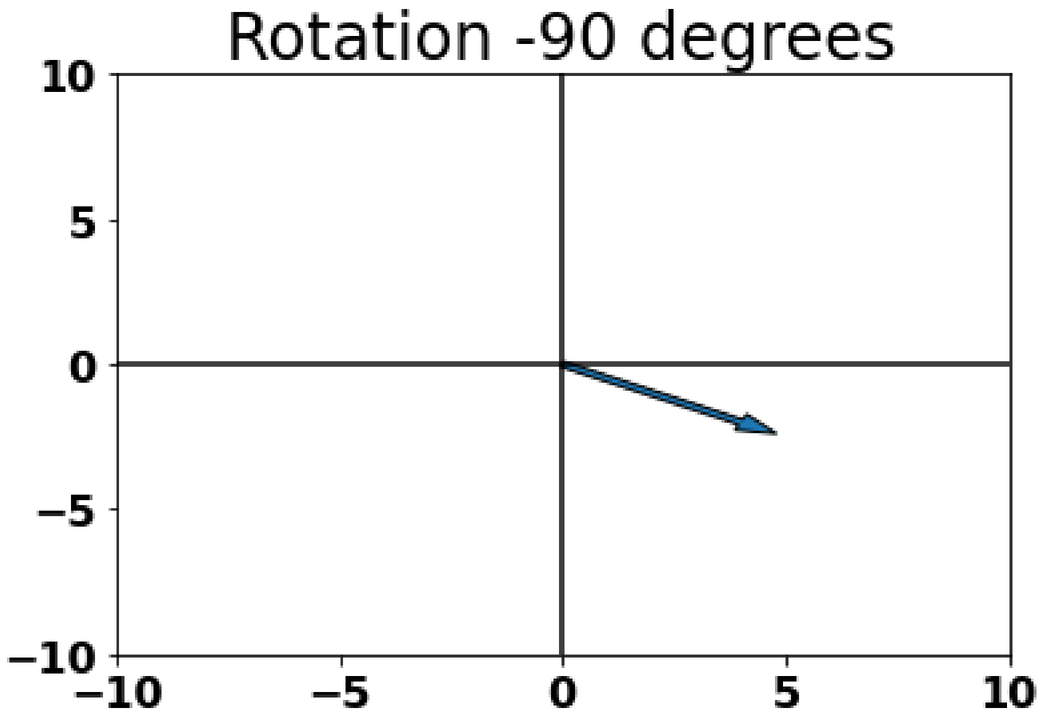
[10]:
```
## transform with identity matrix times 2
A = 2 * np.identity(2)
print('A = \n' + str(A))
plot_vector(A @ v, 'Identity transform times 2')
```

```
A =
[[2. 0.]
 [0. 2.]]
```



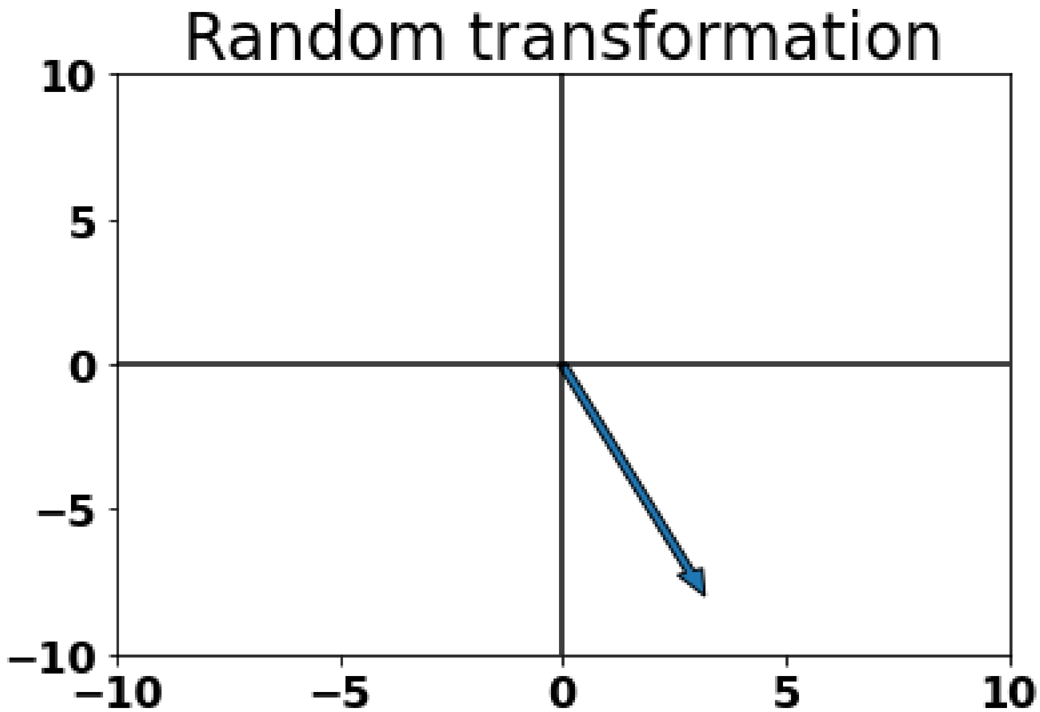
[11]:
```
A = np.array([[0, 1], [-1, 0]])
print('A = \n' + str(A))
plot_vector(A @ v, 'Rotation -90 degrees')
```

```
A =
[[ 0  1]
 [-1  0]]
```



[12]:
```
np.random.seed(1)
A = np.reshape(np.random.uniform(low=-2, high=2, size=4), newshape=(2, 2))
print('A = \n' + str(A))
plot_vector(A @ v, 'Random transformation')
```

```
A =
[[-0.33191198  0.88129797]
 [-1.9995425  -0.79066971]]
```



[13]:
```
## eigenvalues and eigenvectors
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
print(eigen_vals.shape)
print(eigen_vecs.shape)
```

```
(13,)
(13, 13)
```

[14]:
```
# checking whether the equation holds
evec_0 = eigen_vecs[:, 0]
eval_0 = eigen_vals[0]

mat_trans = cov_mat @ evec_0 ## A times x
scalar_trans = eval_0 * evec_0 # lambda times x
print(mat_trans)
print(scalar_trans)

print(np.isclose(mat_trans, scalar_trans)) ## instead of using "==", due to␣
↪rounding error
```

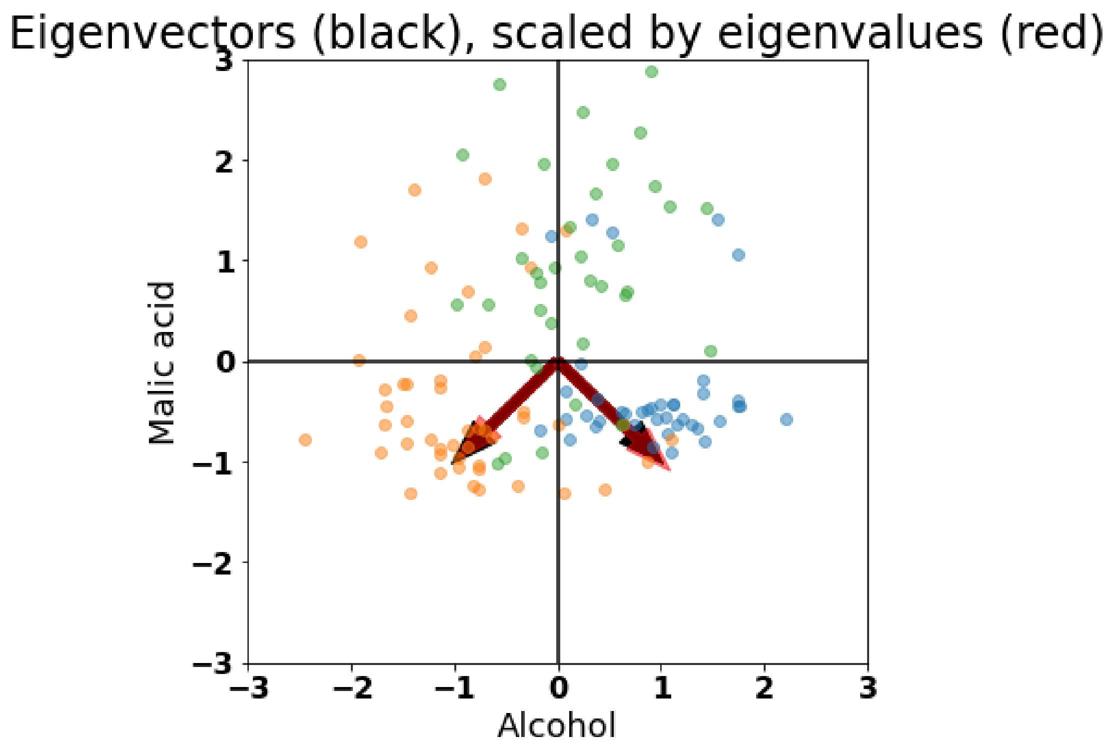
```
[ 0.7176924  -1.18513985 -0.14644842 -1.24846827  0.5909797   1.90479358
  2.07074217 -1.49875647  1.49568723 -0.48283127  1.46928422  1.80140436
  1.43147537]
[ 0.7176924  -1.18513985 -0.14644842 -1.24846827  0.5909797   1.90479358
```

```
 2.07074217 -1.49875647  1.49568723 -0.48283127  1.46928422  1.80140436
 1.43147537]
[ True  True  True  True  True  True  True  True  True  True  True  True
 True]
```

[15]:

```
## now let's do it for the first three features (such that we can make a plot)
X_reduced_std_mat = np.cov(X_train_std[:, 0:2].T)
reduced_eigen_vals, reduced_eigen_vecs = np.linalg.eig(X_reduced_std_mat)

plt.figure(figsize=(6, 6))

for target_value in target_values:
    plt.plot(X_train_std[y_train == target_value, 0], X_train_std[y_train ==␣
↪target_value, 1], 'o', alpha=0.5)
plt.xlabel('Alcohol')
plt.ylabel('Malic acid')

plt.arrow(0, 0, reduced_eigen_vecs[0, 0], reduced_eigen_vecs[0, 1], width=0.1,␣
↪color='k')
plt.arrow(0, 0, reduced_eigen_vecs[1, 0], reduced_eigen_vecs[1, 1], width=0.1,␣
↪color='k')

print(reduced_eigen_vals)
plt.arrow(0, 0, reduced_eigen_vals [0] * reduced_eigen_vecs[0, 0],
                reduced_eigen_vals[0]  * reduced_eigen_vecs[0, 1], width=0.1,␣
↪color='r', alpha=0.5)
plt.arrow(0, 0, reduced_eigen_vals [1] * reduced_eigen_vecs[1, 0],
                reduced_eigen_vals[1]  * reduced_eigen_vecs[1, 1], width=0.1,␣
↪color='r', alpha=0.5)
plt.title('Eigenvectors (black), scaled by eigenvalues (red)')
plt.axvline(color='k')
plt.axhline(color='k')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.show()
print(np.var(X_std[:, :2], axis=0))
```

```
[0.92015307 1.09610709]
```




```
[1. 1.]
```

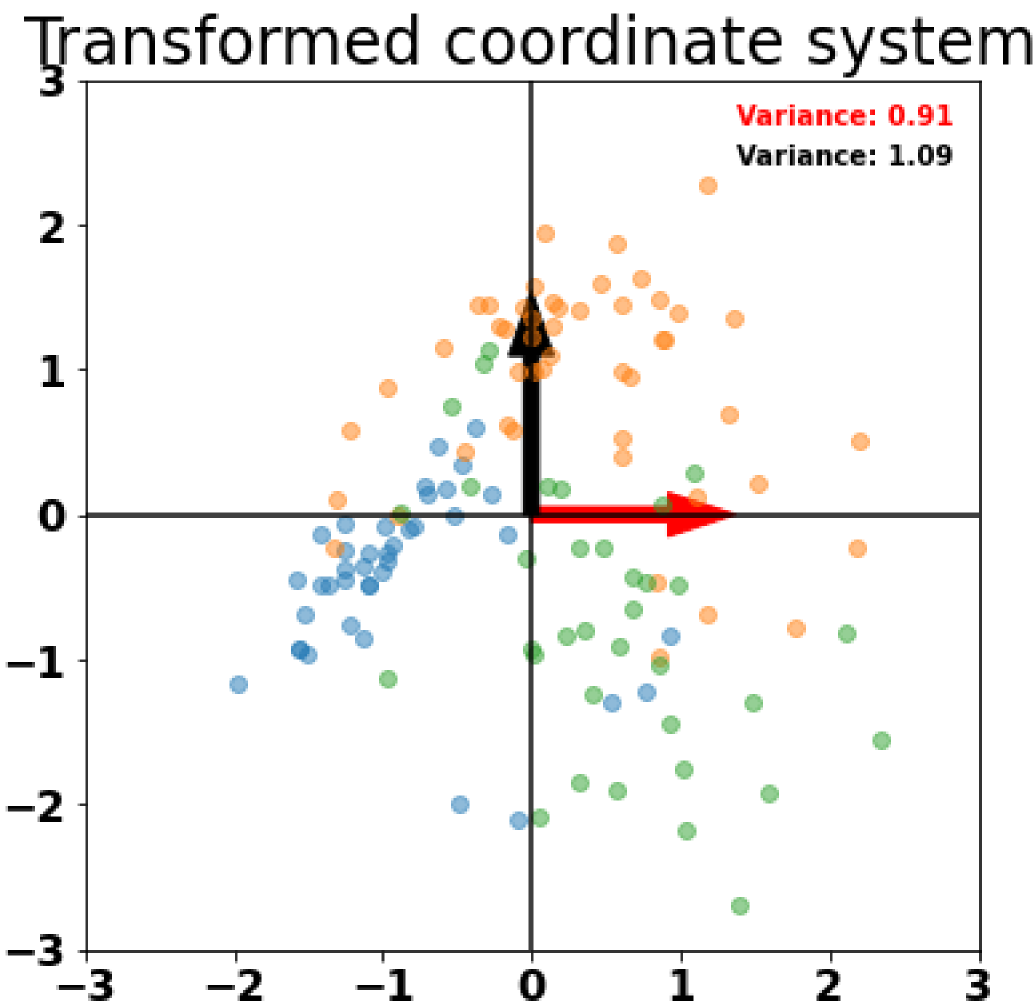
[16]:
```
print('Eigenvalues:\n' + str(reduced_eigen_vals) + '\n')
print('Eigenvectors:\n '  +  str(reduced_eigen_vecs)+ '\n')
print('Covariance matrix:\n' + str(X_reduced_std_mat))
```

```
Eigenvalues:
[0.92015307 1.09610709]

Eigenvectors:
 [[-0.70710678 -0.70710678]
 [ 0.70710678 -0.70710678]]

Covariance matrix:
[[1.00813008 0.08797701]
 [0.08797701 1.00813008]]
```

[17]:
```
radian = np.arctan(reduced_eigen_vecs[1, 1] / reduced_eigen_vecs[1, 0]) - np.pi


A = np.array([
                [np.cos(radian), np.sin(radian)],
                [-np.sin(radian), np.cos(radian)]
```

```
             ])

trans = (A @ X_train_std[:, :2].T).T
plt.figure(figsize=(6, 6))

for target_value in target_values:
    plt.plot(trans[y_train == target_value, 0], trans[y_train == target_value,␣
 ↪1], 'o', alpha=0.5)

trans_vector = (A @ reduced_eigen_vecs)
plt.arrow(0, 0, reduced_eigen_vals[0] * trans_vector[0, 0],
                reduced_eigen_vals[0] * trans_vector[0, 1], width=0.1,␣
 ↪color='r')
plt.arrow(0, 0, reduced_eigen_vals[1] * trans_vector[1, 0],
                reduced_eigen_vals[1] * trans_vector[1, 1], width=0.1,␣
 ↪color='k')
plt.axvline(color='k')
plt.axhline(color='k')
plt.xlim(-3, 3)
plt.ylim(-3, 3)
plt.title('Transformed coordinate system')
var = np.var(trans, axis=0)
plt.legend(['Variance: ' + str(round(var[0], 2)),
            'Variance: ' + str(round(var[1], 2))], labelcolor=['r', 'k'],␣
 ↪markerscale=0,
          frameon=False)
```

[17]: <matplotlib.legend.Legend at 0x7fe3806806a0>



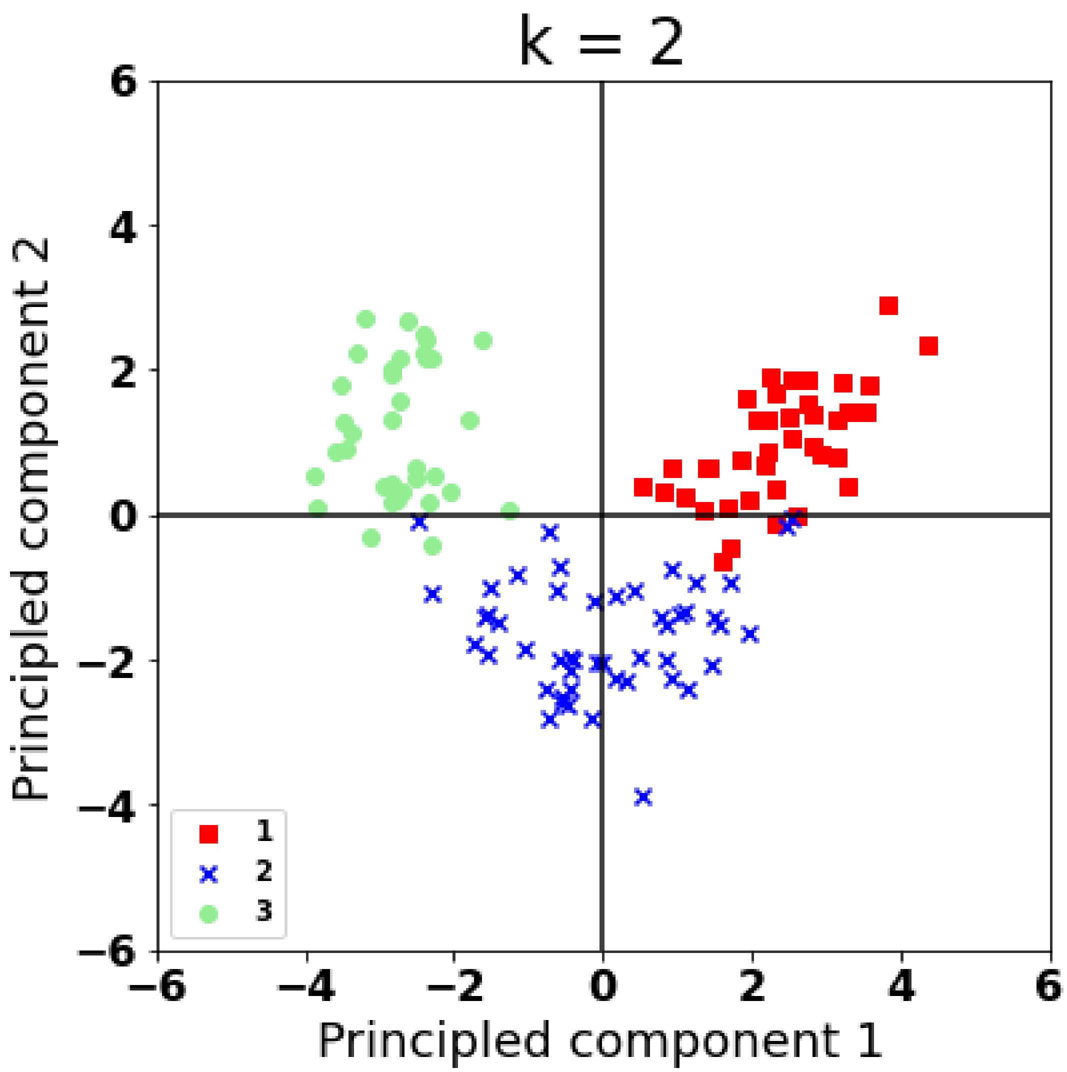
[18]:
```
## going back to the full feature matrix
print(eigen_vals)
print(np.max(eigen_vals))

## ratio
print(eigen_vals[0] / np.sum(eigen_vals) * 100)
print(np.min(eigen_vals) / np.sum(eigen_vals) * 100)
print(np.argmin(eigen_vals))
```

```
[4.8923083  2.46635032 1.42809973 1.01233462 0.84906459 0.60181514
 0.52251546 0.08414846 0.33051429 0.29595018 0.16831254 0.21432212
 0.2399553 ]
4.892308303273744
37.32964772349068
0.642075693386831
7
```

[19]:
```python
## book p. 133
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i])
              for i in range(len(eigen_vals))]
eigen_pairs.sort(reverse=True)

W = np.hstack((eigen_pairs[0][1][:, np.newaxis],
               eigen_pairs[1][1][:, np.newaxis]))

# w == eigen_vecs[:, :2] # is also true - first part is general
print('Weight matrix:\n', W)
```

```
Weight matrix:
 [[ 0.14669811  0.50417079]
 [-0.24224554  0.24216889]
 [-0.02993442  0.28698484]
 [-0.25519002 -0.06468718]
 [ 0.12079772  0.22995385]
 [ 0.38934455  0.09363991]
 [ 0.42326486  0.01088622]
 [-0.30634956  0.01870216]
 [ 0.30572219  0.03040352]
 [-0.09869191  0.54527081]
 [ 0.30032535 -0.27924322]
 [ 0.36821154 -0.174365  ]
 [ 0.29259713  0.36315461]]
```

[87]:
```python
Z = X_train_std @ W

colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
markers = ('s', 'x', 'o', '^', 'v')
plt.figure(figsize=(6, 6))
for target, color, marker in zip(np.unique(y_train), colors, markers):
    plt.scatter(Z[y_train == target, 0],
                Z[y_train == target, 1],
                color=color, label=target, marker=marker)
plt.xlabel('Principled component 1')
plt.ylabel('Principled component 2')
plt.legend(loc='lower left')
plt.xlim(-6, 6)
plt.ylim(-6, 6)
plt.axvline(color='k')
plt.axhline(color='k')
plt.title('k = 2')
plt.show()
```



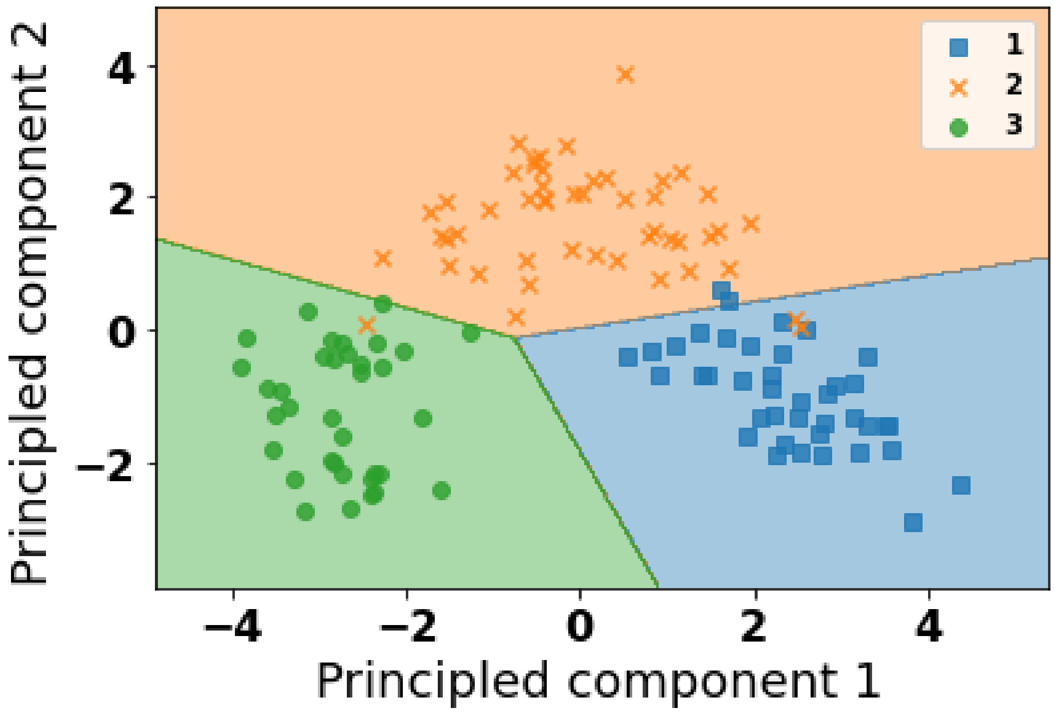
[29]:
```
from matplotlib.colors import ListedColormap
def plot_decision_regions(X, y, classifier, resolution=0.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('#1f77b4', '#ff7f0e', '#2ca02c', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    # plot the decision surface

    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
    np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
```

```
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
            alpha=0.8, c=cmap(idx),
            marker=markers[idx], label=cl)
```

[41]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
logr_pca = LogisticRegression(penalty='none')

X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)

logr_pca.fit(X_train_pca, y_train)

plot_decision_regions(X_train_pca, y_train, logr_pca)
plt.xlabel('Principled component 1')
plt.ylabel('Principled component 2')
plt.legend(loc='upper right')
plt.show()

score_pca = logr_pca.score(X_test_pca, y_test)
print(score_pca)
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a
single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a
single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a
single row if you intend to specify the same RGB or RGBA value for all points.
```




```
0.9629629629629629
```

[40]:
```
## no PCA
logr = LogisticRegression(penalty='none')
logr.fit(X_train_std, y_train)
score_normal = logr.score(X_test_std, y_test)
print(score_normal)
```

```
1.0
```

[45]:
```
## with PCA
pca = PCA(n_components=6)
logr_pca = LogisticRegression(penalty='none')

X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)

logr_pca.fit(X_train_pca, y_train)
score_pca = logr_pca.score(X_test_pca, y_test)
print(score_pca)
```

```
1.0
```

[78]:

```
# with cross-validation
from sklearn.model_selection import cross_val_score, StratifiedKFold
cv = StratifiedKFold(n_splits=10)

scores_list_pca = list()
n_components = np.arange(1, 14)

for n_component in n_components:
    print(n_component)
    pca = PCA(n_components=n_component)
    logr = LogisticRegression(penalty='none')

    X_pca = pca.fit_transform(X_std)


    scores_pca = cross_val_score(logr, X_pca, y, cv=cv)
    mean = np.mean(scores_pca)
    scores_list_pca.append(mean)
    print(mean)

scores_normal = cross_val_score(logr, X_std, y, cv=cv)
print(np.mean(scores_normal))
```

```
1
0.8496732026143791
2
0.9552287581699346
3
0.9277777777777778
4
0.938562091503268
5
0.9722222222222221
6
0.9666666666666668
7
0.9666666666666668
8
0.961111111111111
9
0.9833333333333332
10
0.9777777777777779
11
0.9777777777777779
12
0.9777777777777779
```

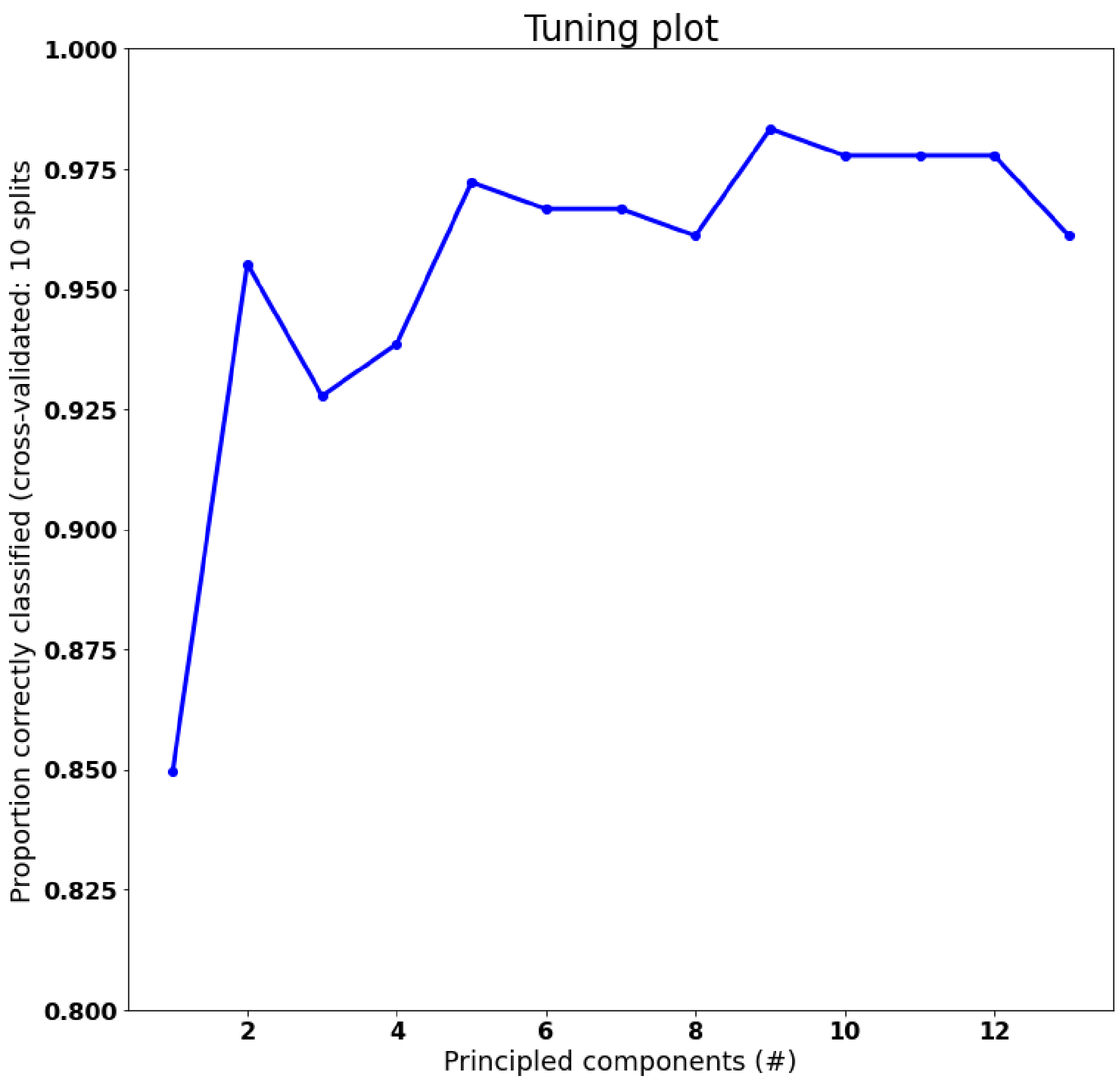
```
13
0.9611111111111111
0.9611111111111111
```

[86]:

```
plt.figure(figsize=(12, 12))
plt.plot(n_components, scores_list_pca, 'bo-', linewidth=3)
plt.ylim(0.8, 1)
plt.xlabel('Principled components (#)')
plt.ylabel('Proportion correctly classified (cross-validated: ' + str(cv.
 ↪get_n_splits()) + ' splits')
plt.title('Tuning plot')
```

[86]:

```
Text(0.5, 1.0, 'Tuning plot')
```



[99]:
```
eigen_vals_sorted = np.flip(np.sort(eigen_vals))
print(eigen_vals_sorted)

variance_explained_9 = 0
for i in range(9):
    variance_explained_9 += eigen_vals_sorted[i] / np.sum(eigen_vals_sorted)

variance_explained_2 = 0
for i in range(2):
    variance_explained_2 += eigen_vals_sorted[i] / np.sum(eigen_vals_sorted)

print(variance_explained_9)
print(variance_explained_2)
```

```
[4.8923083  2.46635032 1.42809973 1.01233462 0.84906459 0.60181514
 0.52251546 0.33051429 0.29595018 0.2399553  0.21432212 0.16831254
 0.08414846]
0.9460739298540608
0.5614857383009024
```