Chapter_5_Compressing_Data_via_Dimensionality_Reduction

March 18, 2024

0.1 Extracting the Principal components

[1]:
```
import pandas as pd
df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/
 ↪machine-learning-databases/wine/wine.data',header=None)
```

[2]:
```
from sklearn.model_selection import train_test_split
X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values
X_train, X_test, y_train, y_test = \
train_test_split(X, y, test_size=0.3,stratify=y,random_state=0)
# standardize the features
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

[3]:
```
import numpy as np
cov_mat = np.cov(X_train_std.T)
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
print('\nEigenvalues \n%s' % eigen_vals)
```

```
Eigenvalues
[4.84274532 2.41602459 1.54845825 0.96120438 0.84166161 0.6620634
 0.51828472 0.34650377 0.3131368  0.10754642 0.21357215 0.15362835
 0.1808613 ]
```

[4]:
```
eigen_vecs
```

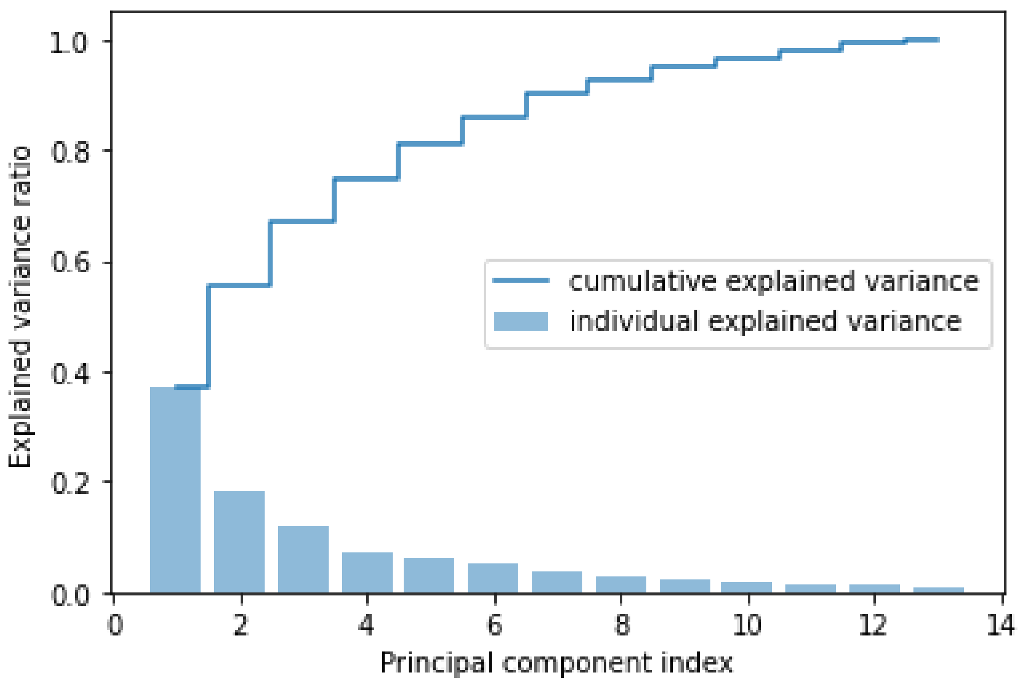
[4]:
```
array([[-1.37242175e-01,  5.03034778e-01, -1.37748734e-01,
        -3.29610003e-03, -2.90625226e-01,  2.99096847e-01,
         7.90529293e-02, -3.68176414e-01, -3.98377017e-01,
        -9.44869777e-02,  3.74638877e-01, -1.27834515e-01,
         2.62834263e-01],
       [ 2.47243265e-01,  1.64871190e-01,  9.61503863e-02,
         5.62646692e-01,  8.95378697e-02,  6.27036396e-01,
        -2.74002014e-01, -1.25775752e-02,  1.10458230e-01,
         2.63652406e-02, -1.37405597e-01,  8.06401578e-02,
        -2.66769211e-01],
```

```
[-2.54515927e-02,  2.44564761e-01,  6.77775667e-01,
 -1.08977111e-01, -1.60834991e-01,  3.89128239e-04,
  1.32328045e-01,  1.77578177e-01,  3.82496856e-01,
  1.42747511e-01,  4.61583035e-01,  1.67924873e-02,
 -1.15542548e-01],
[ 2.06945084e-01, -1.13529045e-01,  6.25040550e-01,
  3.38187002e-02,  5.15873402e-02, -4.05836452e-02,
  2.23999097e-01, -4.40592110e-01, -2.43373853e-01,
 -1.30485780e-01, -4.18953989e-01, -1.10845657e-01,
  1.99483410e-01],
[-1.54365821e-01,  2.89745182e-01,  1.96135481e-01,
 -3.67511070e-01,  6.76487073e-01,  6.57772614e-02,
 -4.05268966e-01,  1.16617503e-01, -2.58982359e-01,
 -6.76080782e-02,  1.00470630e-02,  7.93879562e-02,
  2.89018810e-02],
[-3.93769523e-01,  5.08010391e-02,  1.40310572e-01,
  2.40245127e-01, -1.18511144e-01, -5.89776247e-02,
 -3.47419412e-02,  3.50192127e-01, -3.42312860e-01,
  4.59917661e-01, -2.21254241e-01, -4.91459313e-01,
 -6.63868598e-02],
[-4.17351064e-01, -2.28733792e-02,  1.17053859e-01,
  1.87053299e-01, -1.07100349e-01, -3.01103180e-02,
  4.17835724e-02,  2.18718183e-01, -3.61231642e-02,
 -8.14583947e-01, -4.17513600e-02, -5.03074004e-02,
 -2.13349079e-01],
[ 3.05728961e-01,  9.04888470e-02,  1.31217777e-01,
 -2.29262234e-02, -5.07581610e-01, -2.71728086e-01,
 -6.31145686e-01,  1.97129425e-01, -1.71436883e-01,
 -9.57480885e-02, -8.87569452e-02,  1.75328030e-01,
  1.86391279e-01],
[-3.06683469e-01,  8.35232677e-03,  3.04309008e-02,
  4.96262330e-01,  2.01634619e-01, -4.39997519e-01,
 -3.23122775e-01, -4.33055871e-01,  2.44370210e-01,
  6.72468934e-02,  1.99921861e-01, -3.67595797e-03,
  1.68082985e-01],
[ 7.55406578e-02,  5.49775805e-01, -7.99299713e-02,
  1.06482939e-01,  5.73607091e-03, -4.11743459e-01,
  2.69082623e-01, -6.68411823e-02, -1.55514919e-01,
  8.73336218e-02, -2.21668868e-01,  3.59756535e-01,
 -4.66369031e-01],
[-3.26132628e-01, -2.07164328e-01,  5.30591506e-02,
 -3.69053747e-01, -2.76914216e-01,  1.41673377e-01,
 -3.02640661e-01, -4.59762295e-01,  2.11961247e-02,
  1.29061125e-01, -9.84694573e-02,  4.04669797e-02,
 -5.32483880e-01],
[-3.68610222e-01, -2.49025357e-01,  1.32391030e-01,
  1.42016088e-01, -6.66275572e-02,  1.75842384e-01,
```

```
        1.30540143e-01,  1.10827548e-01, -2.38089559e-01,
        1.87646268e-01,  1.91205783e-02,  7.42229543e-01,
        2.37835283e-01],
      [-2.96696514e-01,  3.80229423e-01, -7.06502178e-02,
       -1.67682173e-01, -1.28029045e-01,  1.38018388e-01,
        8.11335043e-04,  5.60817288e-03,  5.17278463e-01,
        1.21112574e-02, -5.42532072e-01,  3.87395209e-02,
        3.67763359e-01]])
```

[5]:

```python
tot = sum(eigen_vals)
var_exp = [(i / tot) for i in sorted(eigen_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
import matplotlib.pyplot as plt
plt.bar(range(1,14), var_exp, alpha=0.5, align='center',label='individual␣
↪explained variance')
plt.step(range(1,14), cum_var_exp, where='mid',label='cumulative explained␣
↪variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal component index')
plt.legend(loc='best')
plt.show()
```



[6]:

```python
# Make a list of (eigenvalue, eigenvector) tuples
```

```python
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i]) for i in␣
 ↪range(len(eigen_vals))]
# Sort the (eigenvalue, eigenvector) tuples from high to low
eigen_pairs.sort(key=lambda k: k[0], reverse=True)
```

[7]:
```python
eigen_pairs
```

[7]:
```
[(4.842745315655895,
  array([-0.13724218,  0.24724326, -0.02545159,  0.20694508, -0.15436582,
         -0.39376952, -0.41735106,  0.30572896, -0.30668347,  0.07554066,
         -0.32613263, -0.36861022, -0.29669651])),
 (2.4160245870352255,
  array([ 0.50303478,  0.16487119,  0.24456476, -0.11352904,  0.28974518,
          0.05080104, -0.02287338,  0.09048885,  0.00835233,  0.54977581,
         -0.20716433, -0.24902536,  0.38022942])),
 (1.548458248820353,
  array([-0.13774873,  0.09615039,  0.67777567,  0.62504055,  0.19613548,
          0.14031057,  0.11705386,  0.13121778,  0.0304309 , -0.07992997,
          0.05305915,  0.13239103, -0.07065022])),
 (0.9612043774977358,
  array([-0.0032961 ,  0.56264669, -0.10897711,  0.0338187 , -0.36751107,
          0.24024513,  0.1870533 , -0.02292622,  0.49626233,  0.10648294,
         -0.36905375,  0.14201609, -0.16768217])),
 (0.8416616104578439,
  array([-0.29062523,  0.08953787, -0.16083499,  0.05158734,  0.67648707,
         -0.11851114, -0.10710035, -0.50758161,  0.20163462,  0.00573607,
         -0.27691422, -0.06662756, -0.12802904])),
 (0.6620634040383044,
  array([ 2.99096847e-01,  6.27036396e-01,  3.89128239e-04, -4.05836452e-02,
          6.57772614e-02, -5.89776247e-02, -3.01103180e-02, -2.71728086e-01,
         -4.39997519e-01, -4.11743459e-01,  1.41673377e-01,  1.75842384e-01,
          1.38018388e-01])),
 (0.5182847213561956,
  array([ 0.07905293, -0.27400201,  0.13232805,  0.2239991 , -0.40526897,
         -0.03474194,  0.04178357, -0.63114569, -0.32312277,  0.26908262,
         -0.30264066,  0.13054014,  0.00081134])),
 (0.34650376641286756,
  array([-0.36817641, -0.01257758,  0.17757818, -0.44059211,  0.1166175 ,
          0.35019213,  0.21871818,  0.19712942, -0.43305587, -0.06684118,
         -0.45976229,  0.11082755,  0.00560817])),
 (0.31313680047208836,
  array([-0.39837702,  0.11045823,  0.38249686, -0.24337385, -0.25898236,
         -0.34231286, -0.03612316, -0.17143688,  0.24437021, -0.15551492,
          0.02119612, -0.23808956,  0.51727846])),
 (0.21357214660527382,
  array([ 0.37463888, -0.1374056 ,  0.46158303, -0.41895399,  0.01004706,
         -0.22125424, -0.04175136, -0.08875695,  0.19992186, -0.22166887,
```

```
        -0.09846946,  0.01912058, -0.54253207])),
 (0.1808613047949662,
  array([ 0.26283426, -0.26676921, -0.11554255,  0.19948341,  0.02890188,
        -0.06638686, -0.21334908,  0.18639128,  0.16808299, -0.46636903,
        -0.53248388,  0.23783528,  0.36776336])),
 (0.1536283500671103,
  array([-0.12783451,  0.08064016,  0.01679249, -0.11084566,  0.07938796,
        -0.49145931, -0.0503074 ,  0.17532803, -0.00367596,  0.35975654,
         0.04046698,  0.74222954,  0.03873952])),
 (0.1075464236967098,
  array([-0.09448698,  0.02636524,  0.14274751, -0.13048578, -0.06760808,
         0.45991766, -0.81458395, -0.09574809,  0.06724689,  0.08733362,
         0.12906113,  0.18764627,  0.01211126]))]
```

[8]:
```
w = np.hstack((eigen_pairs[0][1][:, np.newaxis],eigen_pairs[1][1][:, np.
 ↪newaxis]))
print('Matrix W:\n', w)
```

```
Matrix W:
 [[-0.13724218  0.50303478]
 [ 0.24724326  0.16487119]
 [-0.02545159  0.24456476]
 [ 0.20694508 -0.11352904]
 [-0.15436582  0.28974518]
 [-0.39376952  0.05080104]
 [-0.41735106 -0.02287338]
 [ 0.30572896  0.09048885]
 [-0.30668347  0.00835233]
 [ 0.07554066  0.54977581]
 [-0.32613263 -0.20716433]
 [-0.36861022 -0.24902536]
 [-0.29669651  0.38022942]]
```

[9]:
```
X_train_std[0].dot(w)
```

[9]:
```
array([2.38299011, 0.45458499])
```

[10]:
```
X_train_pca = X_train_std.dot(w)
```

[11]:
```
X_train_pca
```

[11]:
```
array([[ 2.38299011,  0.45458499],
       [-1.96578183,  1.65376939],
       [-2.53907598,  1.02909066],
       [-1.43010776,  0.6024011 ],
       [ 3.14147227,  0.66214979],
       [ 0.50253552, -2.08907131],
       [ 0.04867722, -2.27536044],
```

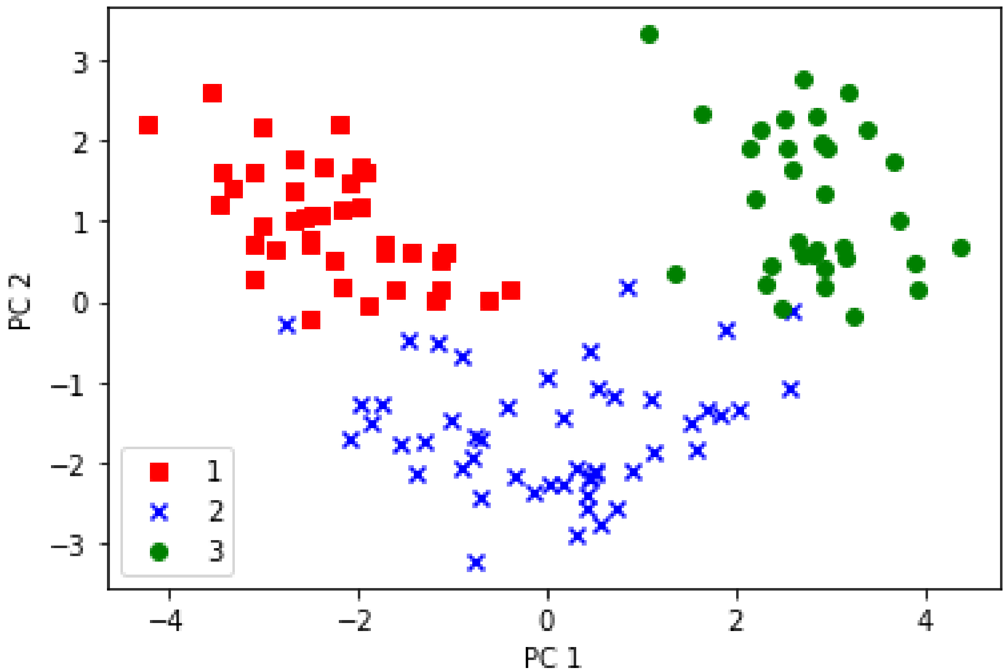
```
[ 2.47888989, -0.08603318],
[ 2.01900259, -1.3538719 ],
[ 0.75156583, -2.55367947],
[ 0.72268915, -1.18404391],
[-3.00366211,  0.94626934],
[ 2.57518878, -1.0697549 ],
[ 3.73151104,  1.01968876],
[-1.12276518,  0.13877   ],
[ 2.85996853,  2.28819559],
[-0.74717125, -3.21746061],
[-1.58427878,  0.16048055],
[ 3.38887101,  2.11550689],
[ 3.15405473,  0.54233966],
[-1.28036506, -1.72926871],
[-1.71438911,  0.71745249],
[-1.55040291, -1.7580591 ],
[ 1.10984489, -1.20480693],
[-0.69108418, -1.71385374],
[-2.086036  , -1.68453671],
[ 2.90393456,  1.95258805],
[-2.07635784,  1.47183304],
[-1.74756185, -1.25842546],
[ 2.59424456, -0.1056037 ],
[-2.50372355,  0.70412212],
[-2.19448402,  2.18657552],
[ 3.91634534,  0.16136475],
[-1.11739618,  0.51921086],
[-0.89996804, -2.04759575],
[-1.71469178,  0.61392169],
[-2.48581303,  0.76839561],
[-0.76080562, -1.67615627],
[ 2.9265371 ,  0.18854741],
[ 2.94423716,  1.34812388],
[-2.38993219,  1.0848074 ],
[ 2.63885049,  0.75274937],
[ 2.51009031,  2.25237953],
[ 3.65248086,  1.74839925],
[-2.65169609,  1.01997476],
[ 0.52544559, -2.13528249],
[ 2.70197573,  0.56476307],
[ 3.18414708,  2.58094695],
[ 1.12517041, -1.85054449],
[ 2.92366519,  0.41699915],
[-1.96122314, -1.28613661],
[ 0.54473673, -1.07897226],
[-0.77030308, -1.93386815],
[-1.16670455,  0.00489815],
```

```
[-1.36475309, -2.13572269],
[ 0.43563732, -2.56929607],
[ 2.96191745,  1.91091009],
[ 2.83609557,  0.65386032],
[ 1.90402089, -0.35296542],
[-2.4858391 , -0.21308835],
[-2.16575568,  1.1468486 ],
[ 0.00669776, -0.94337624],
[ 1.06560181,  3.31221025],
[ 2.13117911,  1.90551304],
[ 1.53543483, -1.50854979],
[-2.66783112,  1.75933599],
[ 0.57279998, -2.7511383 ],
[-0.70710916, -2.43798549],
[-0.99606577, -1.4772411 ],
[-2.67324153,  1.35779609],
[-2.36367378,  1.66537927],
[-0.39171875,  0.13747499],
[-2.98908845,  2.16983165],
[-1.91822539,  1.60141809],
[ 2.3114458 ,  0.207123  ],
[-1.06050503,  0.6004608 ],
[-2.74858609, -0.29016054],
[ 2.26650077,  2.14491758],
[-1.15517469, -0.50262909],
[ 0.16602503, -2.26850051],
[ 1.35589389,  0.33353007],
[-3.31185057,  1.39240115],
[-0.33245686, -2.15639865],
[-2.23205085,  0.52868143],
[ 0.18583758, -1.44446967],
[ 0.84560856,  0.17151684],
[ 2.69500472,  2.74522492],
[ 0.44645674, -0.62393943],
[-1.88961007, -0.04400723],
[-3.08131761,  1.59724429],
[-3.45716348,  1.21428442],
[ 3.87665629,  0.46446004],
[ 1.575516  , -1.82299839],
[-3.43344371,  1.6116814 ],
[-4.20642597,  2.20145366],
[-0.14042971, -2.36871639],
[ 1.82731521, -1.39485103],
[ 2.20564744,  1.28462066],
[ 1.64999054,  2.33211134],
[-1.4611033 , -0.46480324],
[-0.60047516,  0.00920072],
```

```
       [-3.08276231,  0.28287148],
       [ 0.45035749, -2.20263755],
       [ 0.90806897, -2.0881686 ],
       [ 3.24973637, -0.18273485],
       [-3.07882055,  0.69622621],
       [ 2.54277306,  1.88571652],
       [-2.84838157,  0.63274325],
       [-0.88997271, -0.67927226],
       [ 0.32368249, -2.07006175],
       [ 0.32007527, -2.88708519],
       [ 0.44889188, -2.14872532],
       [-2.46582558,  1.0745577 ],
       [ 2.81678113,  0.56344444],
       [-2.16983025,  0.16644199],
       [-2.66728229,  1.38137702],
       [-3.53223924,  2.57906029],
       [-1.96637688,  1.18319185],
       [ 1.68741216, -1.35075321],
       [ 0.43521077, -2.40355817],
       [ 2.59045115,  1.63852921],
       [ 4.35308397,  0.66536041],
       [-1.84315373, -1.50688415],
       [-0.40860955, -1.29720607]])
```

[12]:

```
colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']
for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_pca[y_train==l, 0],X_train_pca[y_train==l, 1],c=c,␣
 ↪label=l, marker=m)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.show()
```

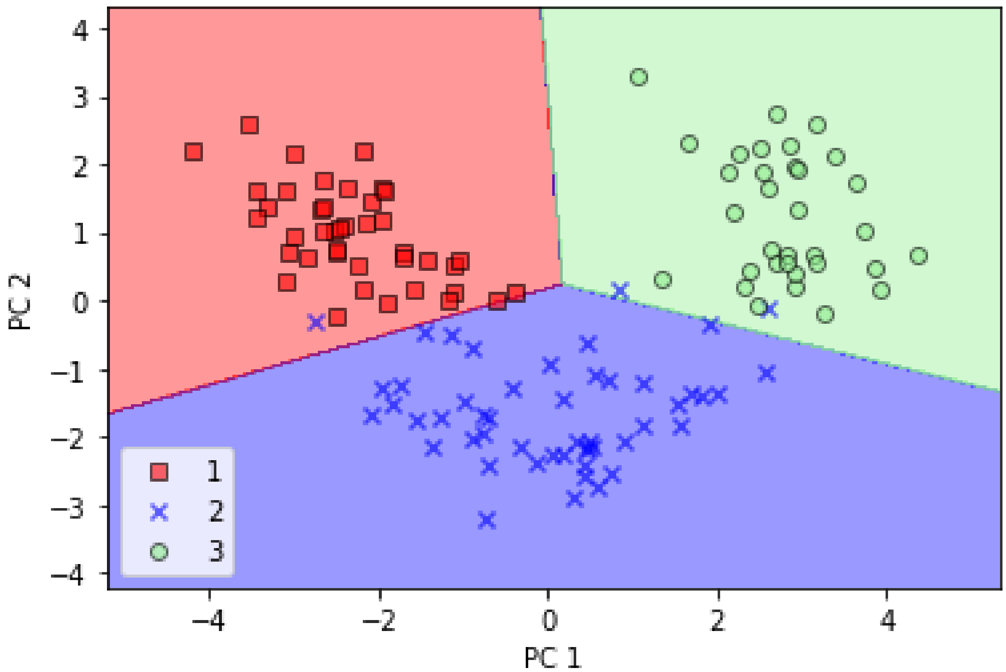


0.2 Principal component analysis in scikit-learn

[13]:
```
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 = ('red', 'blue', 'lightgreen', '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.
↪6,c=cmap(idx),edgecolor='black',marker=markers[idx],label=cl)
```

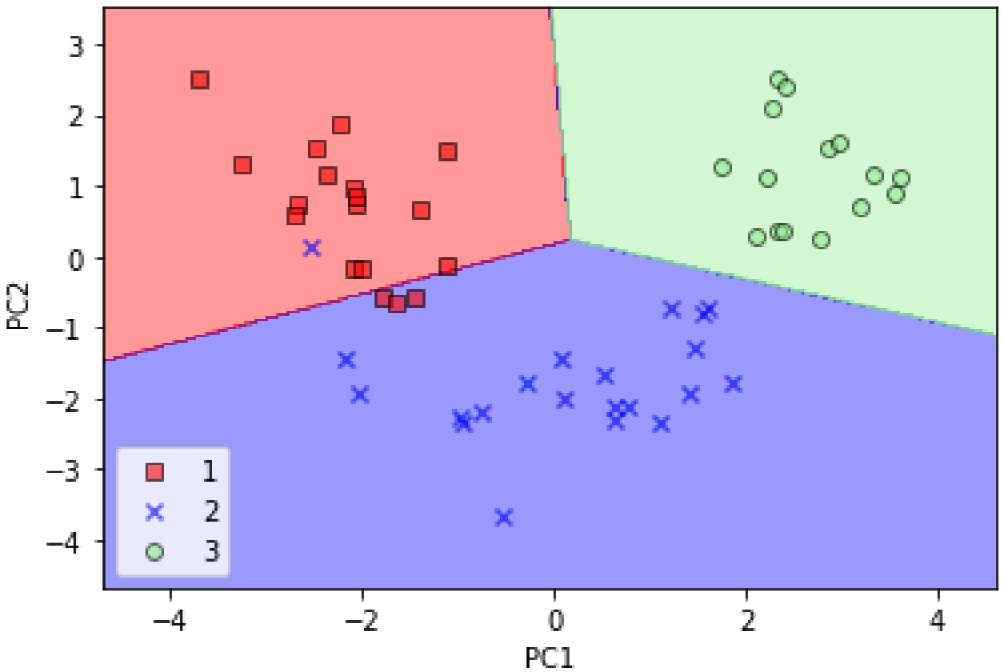
[14]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
lr = LogisticRegression()
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
lr.fit(X_train_pca, y_train)
plot_decision_regions(X_train_pca, y_train, classifier=lr)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.show()
```

```
C:\Users\ankit19.gupta\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_R
aschka\venv_python_3.6\lib\site-packages\sklearn\linear_model\logistic.py:433:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ankit19.gupta\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_R
aschka\venv_python_3.6\lib\site-packages\sklearn\linear_model\logistic.py:460:
FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify
the multi_class option to silence this warning.
  "this warning.", FutureWarning)
*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 2-D 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 2-D 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 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
```



[15]:
```
plot_decision_regions(X_test_pca, y_test, classifier=lr)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend(loc='lower left')
plt.show()
```

```
*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 2-D 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 2-D 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 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
```



[16]:
```
pca = PCA(n_components=None)
X_train_pca = pca.fit_transform(X_train_std)
pca.explained_variance_ratio_
```

[16]:
```
array([0.36951469, 0.18434927, 0.11815159, 0.07334252, 0.06422108,
       0.05051724, 0.03954654, 0.02643918, 0.02389319, 0.01629614,
       0.01380021, 0.01172226, 0.00820609])
```

0.3 Supervised data compression via linear discriminant analysis

0.4 Computing the scatter matrices

[17]:
```
np.set_printoptions(precision=4)
mean_vecs = []
for label in range(1,4):
    mean_vecs.append(np.mean(X_train_std[y_train==label], axis=0))
    print('MV %s: %s\n' %(label, mean_vecs[label-1]))
```

```
MV 1: [ 0.9066 -0.3497  0.3201 -0.7189  0.5056  0.8807  0.9589 -0.5516  0.5416
  0.2338  0.5897  0.6563  1.2075]

MV 2: [-0.8749 -0.2848 -0.3735  0.3157 -0.3848 -0.0433  0.0635 -0.0946  0.0703
 -0.8286  0.3144  0.3608 -0.7253]

MV 3: [ 0.1992  0.866   0.1682  0.4148 -0.0451 -1.0286 -1.2876  0.8287 -0.7795
```

```
 0.9649 -1.209  -1.3622 -0.4013]
```

[18]:
```python
d = 13 # number of features
S_W = np.zeros((d, d))
for label, mv in zip(range(1, 4), mean_vecs):
    class_scatter = np.zeros((d, d))
    for row in X_train_std[y_train == label]:
        row, mv = row.reshape(d, 1), mv.reshape(d, 1)
        class_scatter += (row - mv).dot((row - mv).T)
    S_W += class_scatter
print('Within-class scatter matrix: %sx%s' % (S_W.shape[0], S_W.shape[1]))
```

```
Within-class scatter matrix: 13x13
```

[19]:
```python
S_W
```

[19]:
```
array([[ 5.0722e+01,  3.1007e+00, -7.9323e+00, -5.7848e+00, -2.8879e+00,
         7.8990e+00,  2.4543e+00,  9.3932e-01,  9.0781e-01,  1.5486e+01,
         7.0293e+00, -1.8659e+00,  4.9370e+00],
       [ 3.1007e+00,  9.0179e+01,  4.7074e+00,  1.4750e+01, -1.0900e+01,
        -8.8059e-02,  9.7797e-01,  8.4547e+00,  4.4732e+00, -1.4494e+01,
        -2.0361e+01,  3.5876e+00, -1.1176e+01],
       [-7.9323e+00,  4.7074e+00,  1.1189e+02,  7.0126e+01,  2.2213e+01,
         1.5505e+01,  1.4856e+01,  2.0454e+01, -2.3344e+00,  1.3787e+00,
         5.6585e+00,  8.1917e+00,  3.2570e-01],
       [-5.7848e+00,  1.4750e+01,  7.0126e+01,  9.2147e+01,  1.2485e+01,
         6.2091e+00,  6.2783e+00,  1.2735e+01, -4.7597e+00, -4.1511e+00,
         1.1779e+00,  1.1633e+01, -4.5296e+00],
       [-2.8879e+00, -1.0900e+01,  2.2213e+01,  1.2485e+01,  1.0605e+02,
         1.0950e+01,  5.2875e+00, -2.1136e+01,  1.3076e+01,  5.9930e+00,
         8.4568e+00, -5.3128e-01,  1.1845e+01],
       [ 7.8990e+00, -8.8059e-02,  1.5505e+01,  6.2091e+00,  1.0950e+01,
         5.7194e+01,  2.8971e+01, -7.4850e+00,  3.0810e+01,  1.5516e+01,
         6.6816e-01,  1.9382e+01,  6.3808e+00],
       [ 2.4543e+00,  9.7797e-01,  1.4856e+01,  6.2783e+00,  5.2875e+00,
         2.8971e+01,  3.1388e+01, -1.0236e+01,  2.5069e+01,  1.4922e+01,
        -1.2394e+00,  1.2737e+01,  2.7636e+00],
       [ 9.3932e-01,  8.4547e+00,  2.0454e+01,  1.2735e+01, -2.1136e+01,
        -7.4850e+00, -1.0236e+01,  8.8416e+01, -1.5290e+01, -6.5190e-01,
         2.8315e+00, -1.7076e+01, -7.0906e+00],
       [ 9.0781e-01,  4.4732e+00, -2.3344e+00, -4.7597e+00,  1.3076e+01,
         3.0810e+01,  2.5069e+01, -1.5290e+01,  9.1676e+01,  2.2137e+01,
        -5.9150e+00,  1.1376e+01,  5.9764e+00],
       [ 1.5486e+01, -1.4494e+01,  1.3787e+00, -4.1511e+00,  5.9930e+00,
         1.5516e+01,  1.4922e+01, -6.5190e-01,  2.2137e+01,  5.6702e+01,
        -1.0507e+01, -5.3682e+00,  1.0706e+01],
       [ 7.0293e+00, -2.0361e+01,  5.6585e+00,  1.1779e+00,  8.4568e+00,
```

```
 6.6816e-01, -1.2394e+00,  2.8315e+00, -5.9150e+00, -1.0507e+01,
 5.6566e+01,  3.2692e+00,  9.6829e+00],
[-1.8659e+00,  3.5876e+00,  8.1917e+00,  1.1633e+01, -5.3128e-01,
 1.9382e+01,  1.2737e+01, -1.7076e+01,  1.1376e+01, -5.3682e+00,
 3.2692e+00,  3.8599e+01, -4.6404e+00],
[ 4.9370e+00, -1.1176e+01,  3.2570e-01, -4.5296e+00,  1.1845e+01,
 6.3808e+00,  2.7636e+00, -7.0906e+00,  5.9764e+00,  1.0706e+01,
 9.6829e+00, -4.6404e+00,  3.2604e+01]])
```

[20]:
```python
print('Class label distribution: %s'% np.bincount(y_train)[1:])
```

```
Class label distribution: [41 50 33]
```

[21]:
```python
d = 13 # number of features
S_W = np.zeros((d, d))
for label,mv in zip(range(1, 4), mean_vecs):
    class_scatter = np.cov(X_train_std[y_train==label].T)
    S_W += class_scatter
print('Scaled within-class scatter matrix: %sx%s'% (S_W.shape[0], S_W.shape[1]))
```

```
Scaled within-class scatter matrix: 13x13
```

[22]:
```python
mean_overall = np.mean(X_train_std, axis=0)
d = 13 # number of features
S_B = np.zeros((d, d))
for i, mean_vec in enumerate(mean_vecs):
    n = X_train[y_train == i + 1, :].shape[0]
    mean_vec = mean_vec.reshape(d, 1) # make column vector
    mean_overall = mean_overall.reshape(d, 1)
    S_B += n * (mean_vec - mean_overall).dot((mean_vec - mean_overall).T)
print('Between-class scatter matrix: %sx%s' % (S_B.shape[0], S_B.shape[1]))
```

```
Between-class scatter matrix: 13x13
```

0.5 Selecting linear discriminants for the new feature subspace

[23]:
```python
eigen_vals, eigen_vecs =np.linalg.eig(np.linalg.inv(S_W).dot(S_B))
```

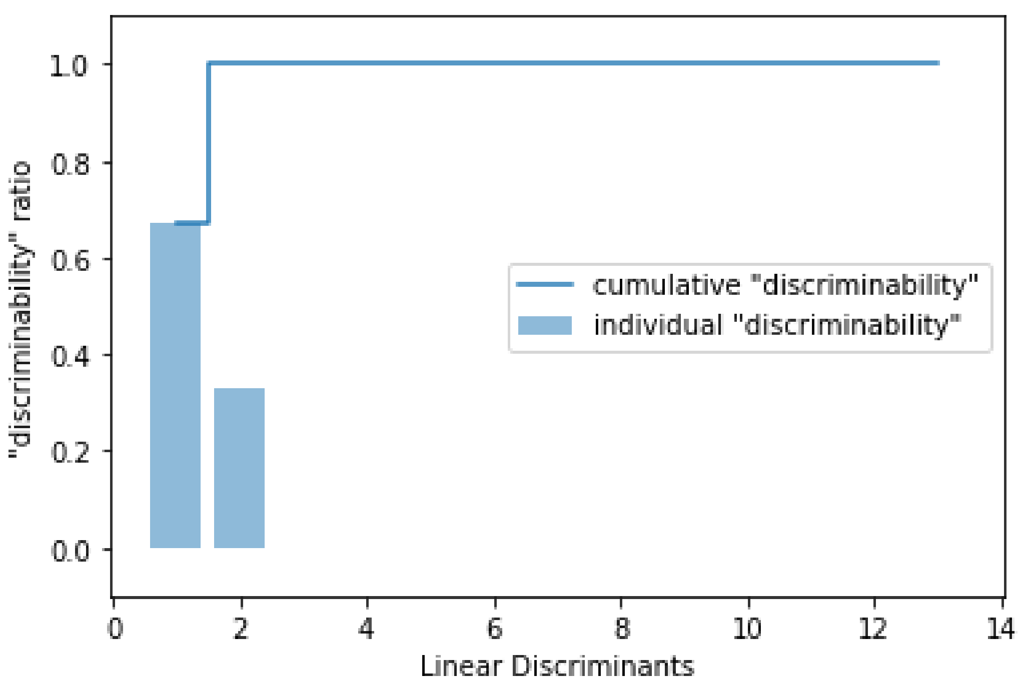
[24]:
```python
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:,i]) for i in␣
 ↪range(len(eigen_vals))]
eigen_pairs = sorted(eigen_pairs,key=lambda k: k[0], reverse=True)
print('Eigenvalues in descending order:\n')
for eigen_val in eigen_pairs:
    print(eigen_val[0])
```

```
Eigenvalues in descending order:

349.6178089059941
172.76152218979388
```

```
3.2173955615499985e-14
3.2173955615499985e-14
1.5112666868767082e-14
1.5112666868767082e-14
1.16438184925797e-14
9.07625765962904e-15
9.07625765962904e-15
7.18576573615579e-15
3.2470704291470134e-15
3.2470704291470134e-15
0.0
```

[25]:
```python
tot = sum(eigen_vals.real)
discr = [(i / tot) for i in sorted(eigen_vals.real, reverse=True)]
cum_discr = np.cumsum(discr)
plt.bar(range(1, 14), discr, alpha=0.5, align='center',label='individual "discriminability"')
plt.step(range(1, 14), cum_discr, where='mid',label='cumulative "discriminability"')
plt.ylabel('"discriminability" ratio')
plt.xlabel('Linear Discriminants')
plt.ylim([-0.1, 1.1])
plt.legend(loc='best')
plt.show()
```

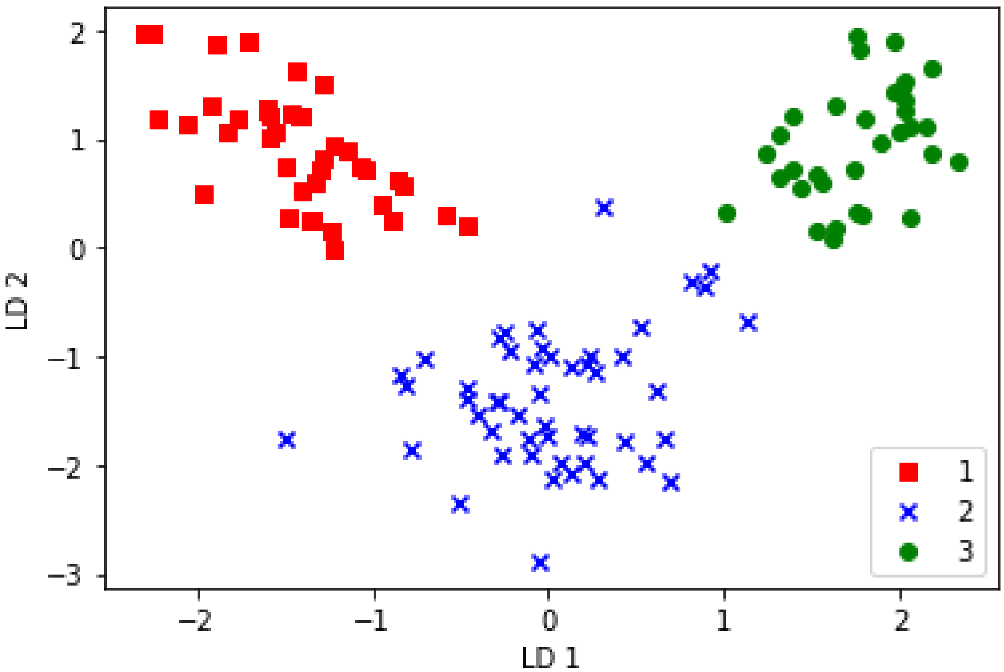


[26]:
```
w = np.hstack((eigen_pairs[0][1][:, np.newaxis].real,eigen_pairs[1][1][:, np.
    ↪newaxis].real))
print('Matrix W:\n', w)
```

```
Matrix W:
 [[-0.1481 -0.4092]
 [ 0.0908 -0.1577]
 [-0.0168 -0.3537]
 [ 0.1484  0.3223]
 [-0.0163 -0.0817]
 [ 0.1913  0.0842]
 [-0.7338  0.2823]
 [-0.075  -0.0102]
 [ 0.0018  0.0907]
 [ 0.294  -0.2152]
 [-0.0328  0.2747]
 [-0.3547 -0.0124]
 [-0.3915 -0.5958]]
```

1 Projecting samples onto the new feature space

[27]:
```
X_train_lda = X_train_std.dot(w)
colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']
for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_lda[y_train==l, 0],X_train_lda[y_train==l, 1] *␣
    ↪(-1),c=c, label=l, marker=m)
plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.legend(loc='lower right')
plt.show()
```



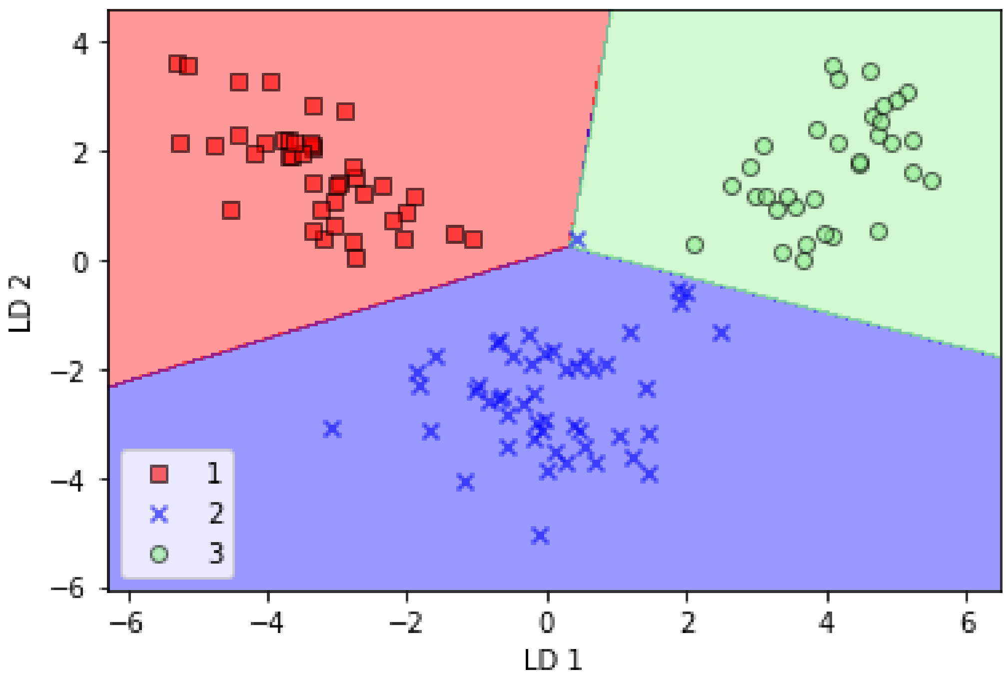
1.1 LDA via scikit-learn

[28]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train_std, y_train)
```

[29]:
```
lr = LogisticRegression()
lr = lr.fit(X_train_lda, y_train)
plot_decision_regions(X_train_lda, y_train, classifier=lr)
plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.legend(loc='lower left')
plt.show()
```

```
C:\Users\ankit19.gupta\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_R
aschka\venv_python_3.6\lib\site-packages\sklearn\linear_model\logistic.py:433:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ankit19.gupta\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_R
aschka\venv_python_3.6\lib\site-packages\sklearn\linear_model\logistic.py:460:
FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify
the multi_class option to silence this warning.
  "this warning.", FutureWarning)
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
```

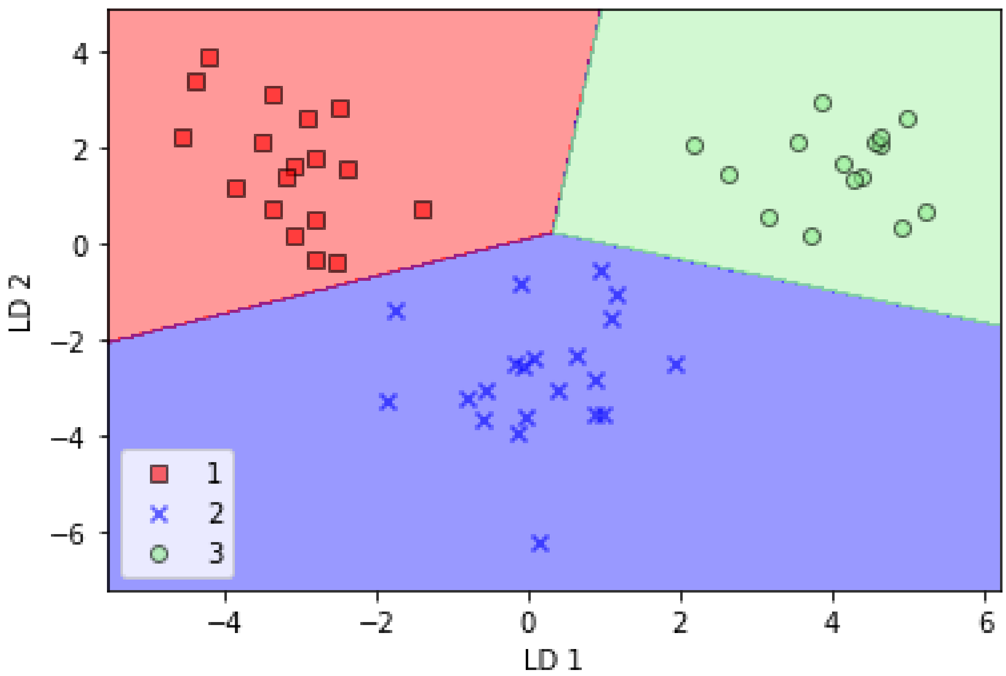
avoided as value-mapping will have precedence in case its length matches with
x & *y*. Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
c argument looks like a single numeric RGB or RGBA sequence, which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



[30]:
```
X_test_lda = lda.transform(X_test_std)
plot_decision_regions(X_test_lda, y_test, classifier=lr)
plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.legend(loc='lower left')
plt.show()
```

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 2-D 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 2-D 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 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
```



1.2 Using kernel principal component analysis for nonlinear mappings

1.3 Implementing a kernel principal component analysis in Python

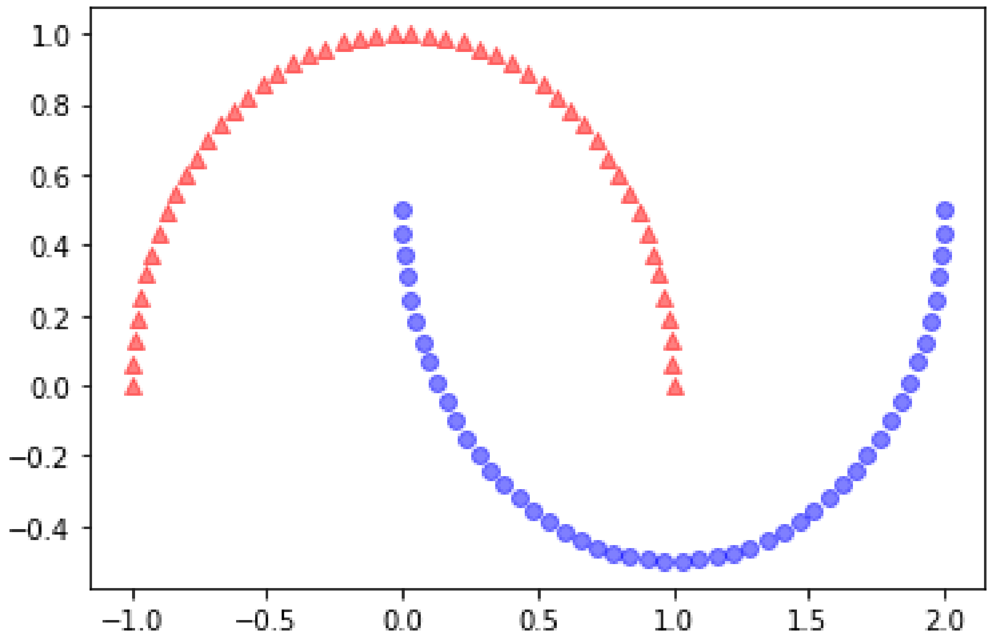
[52]:
```python
from scipy.spatial.distance import pdist, squareform
from scipy import exp
from scipy.linalg import eigh
import numpy as np

def rbf_kernel_pca(X, gamma, n_components):
    """
    RBF kernel PCA implementation.
    Parameters
    ------------
    X: {NumPy ndarray}, shape = [n_samples, n_features]
    gamma: float
    Tuning parameter of the RBF kernel
    n_components: int
```

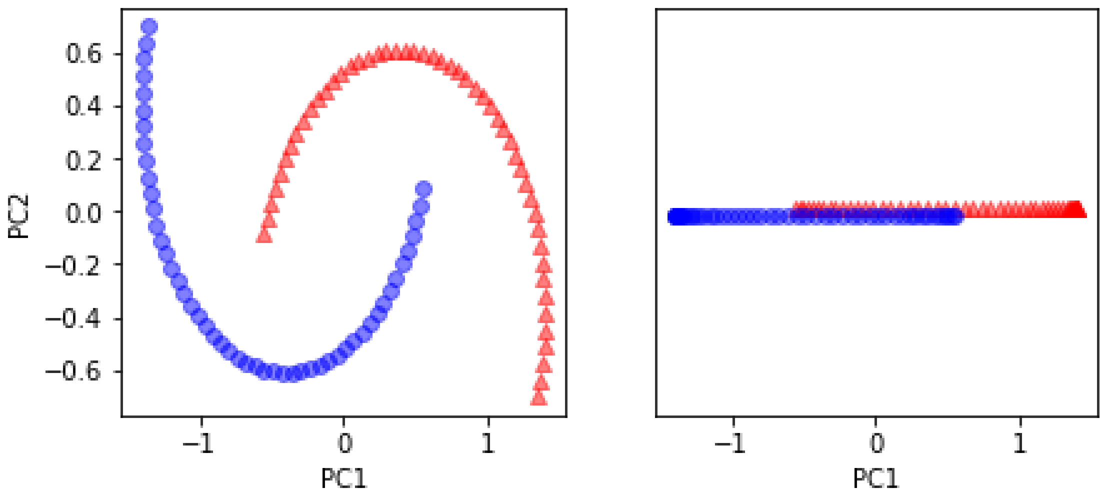
```
    Number of principal components to return
    Returns
    ------------
    X_pc: {NumPy ndarray}, shape = [n_samples, k_features]
    Projected dataset
    """
    # Calculate pairwise squared Euclidean distances
    # in the MxN dimensional dataset.
    sq_dists = pdist(X, 'sqeuclidean')
    # Convert pairwise distances into a square matrix.
    mat_sq_dists = squareform(sq_dists)
    # Compute the symmetric kernel matrix.
    K = exp(-gamma * mat_sq_dists)
    # Center the kernel matrix.
    N = K.shape[0]
    one_n = np.ones((N,N)) / N
    K = K - one_n.dot(K) - K.dot(one_n) + one_n.dot(K).dot(one_n)
    # Obtaining eigenpairs from the centered kernel matrix
    # scipy.linalg.eigh returns them in ascending order
    eigvals, eigvecs = eigh(K)
    eigvals, eigvecs = eigvals[::-1], eigvecs[:, ::-1]
    # Collect the top k eigenvectors (projected samples)
    X_pc = np.column_stack((eigvecs[:, i] for i in range(n_components)))
    return X_pc
```

[53]:
```
## example 1
```

[54]:
```
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=100, random_state=123)
plt.scatter(X[y==0, 0], X[y==0, 1],color='red', marker='^', alpha=0.5)
plt.scatter(X[y==1, 0], X[y==1, 1],color='blue', marker='o', alpha=0.5)
plt.show()
```

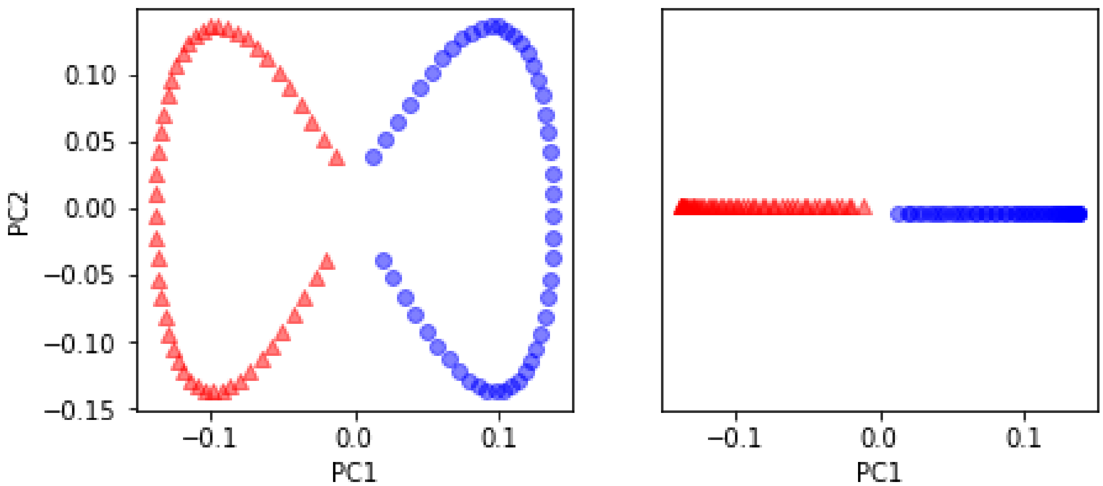


[55]:
```
from sklearn.decomposition import PCA
scikit_pca = PCA(n_components=2)
X_spca = scikit_pca.fit_transform(X)
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(7,3))
ax[0].scatter(X_spca[y==0, 0], X_spca[y==0, 1],color='red', marker='^', alpha=0.
 ↪5)
ax[0].scatter(X_spca[y==1, 0], X_spca[y==1, 1],color='blue', marker='o',␣
 ↪alpha=0.5)
ax[1].scatter(X_spca[y==0, 0], np.zeros((50,1))+0.02,color='red', marker='^',␣
 ↪alpha=0.5)
ax[1].scatter(X_spca[y==1, 0], np.zeros((50,1))-0.02,color='blue', marker='o',␣
 ↪alpha=0.5)
ax[0].set_xlabel('PC1')
ax[0].set_ylabel('PC2')
ax[1].set_ylim([-1, 1])
ax[1].set_yticks([])
ax[1].set_xlabel('PC1')
plt.show()
```



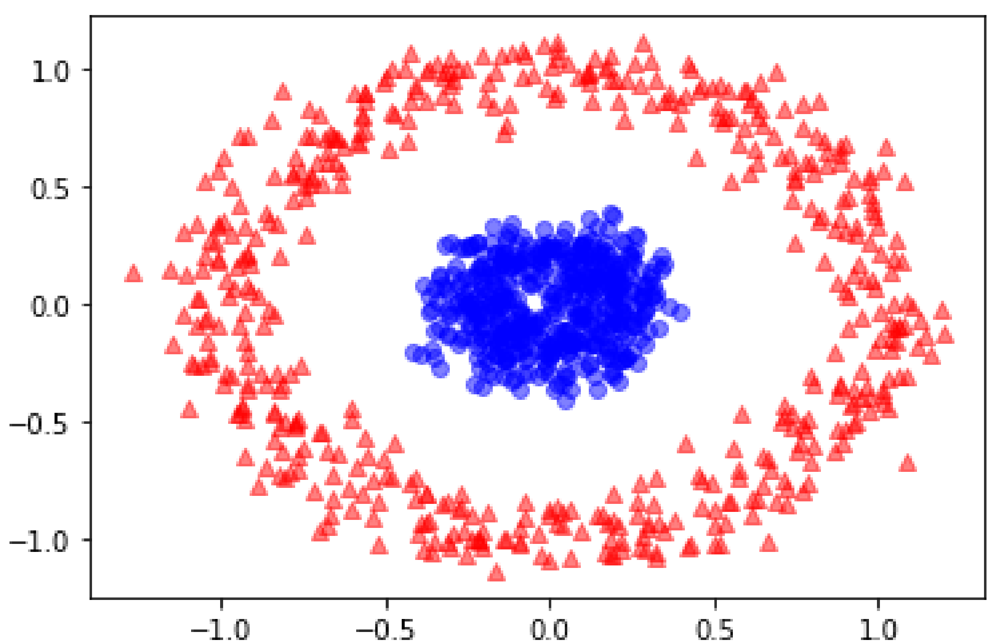
[56]:
```
X_kpca = rbf_kernel_pca(X, gamma=15, n_components=2)
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(7,3))
ax[0].scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],color='red', marker='^', alpha=0.
 ↪5)
ax[0].scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],color='blue', marker='o',␣
 ↪alpha=0.5)
ax[1].scatter(X_kpca[y==0, 0], np.zeros((50,1))+0.02,color='red', marker='^',␣
 ↪alpha=0.5)
ax[1].scatter(X_kpca[y==1, 0], np.zeros((50,1))-0.02,color='blue', marker='o',␣
 ↪alpha=0.5)
ax[0].set_xlabel('PC1')
ax[0].set_ylabel('PC2')
ax[1].set_ylim([-1, 1])
ax[1].set_yticks([])
ax[1].set_xlabel('PC1')
plt.show()
```

```
C:\Users\ankit19.gupta\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_R
aschka\venv_python_3.6\lib\site-packages\ipykernel_launcher.py:27:
DeprecationWarning: scipy.exp is deprecated and will be removed in SciPy 2.0.0,
use numpy.exp instead
C:\Users\ankit19.gupta\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_R
aschka\venv_python_3.6\lib\site-packages\ipykernel_launcher.py:37:
FutureWarning: arrays to stack must be passed as a "sequence" type such as list
or tuple. Support for non-sequence iterables such as generators is deprecated as
of NumPy 1.16 and will raise an error in the future.
```

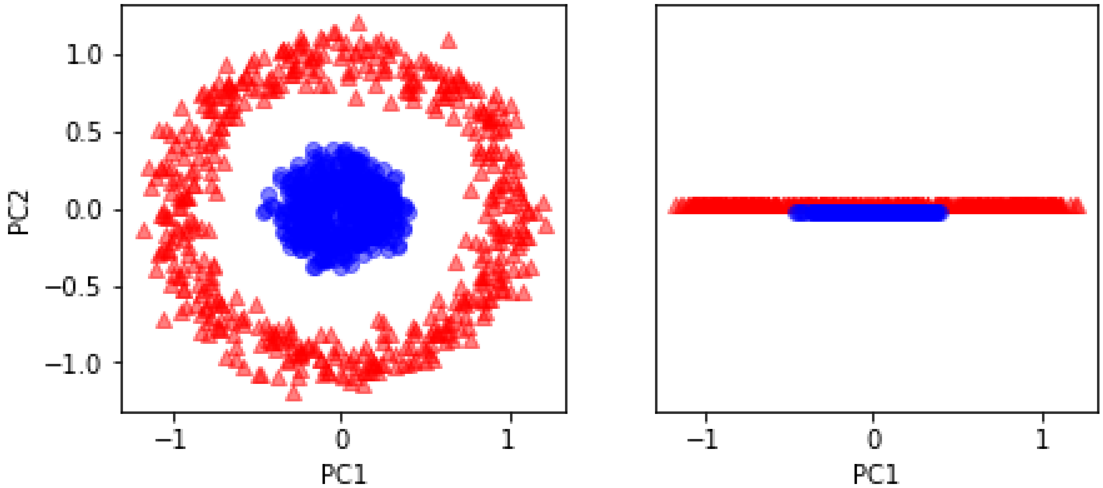


[57]:
```
## example 2
```

[58]:
```
from sklearn.datasets import make_circles
X, y = make_circles(n_samples=1000,random_state=123, noise=0.1, factor=0.2)
plt.scatter(X[y==0, 0], X[y==0, 1],color='red', marker='^', alpha=0.5)
plt.scatter(X[y==1, 0], X[y==1, 1],color='blue', marker='o', alpha=0.5)
plt.show()
```



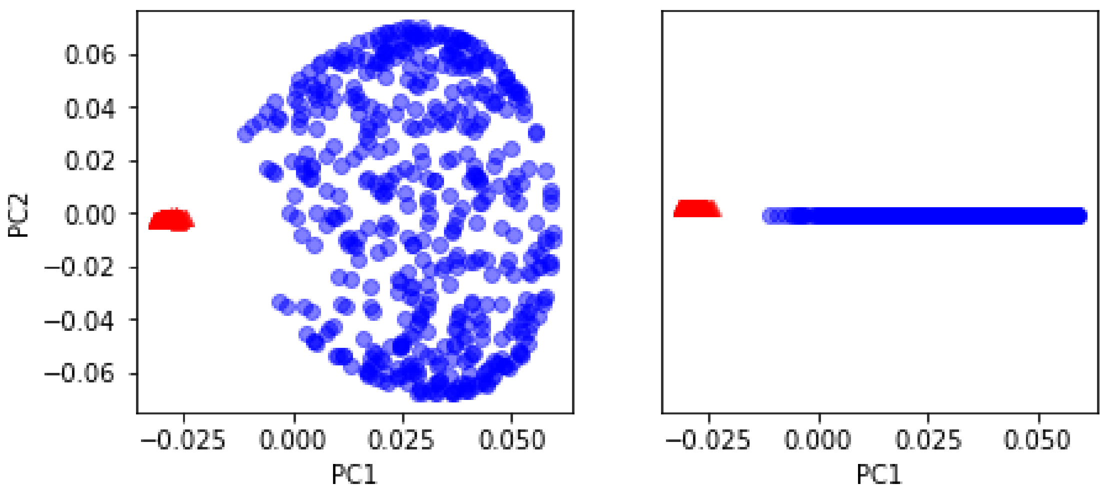
[59]:
```
scikit_pca = PCA(n_components=2)
X_spca = scikit_pca.fit_transform(X)
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(7,3))
ax[0].scatter(X_spca[y==0, 0], X_spca[y==0, 1],color='red', marker='^', alpha=0.
 ↪5)
ax[0].scatter(X_spca[y==1, 0], X_spca[y==1, 1],color='blue', marker='o',␣
 ↪alpha=0.5)
ax[1].scatter(X_spca[y==0, 0], np.zeros((500,1))+0.02,color='red', marker='^',␣
 ↪alpha=0.5)
ax[1].scatter(X_spca[y==1, 0], np.zeros((500,1))-0.02,color='blue', marker='o',␣
 ↪alpha=0.5)
ax[0].set_xlabel('PC1')
ax[0].set_ylabel('PC2')
ax[1].set_ylim([-1, 1])
ax[1].set_yticks([])
ax[1].set_xlabel('PC1')
plt.show()
```



[60]:
```
X_kpca = rbf_kernel_pca(X, gamma=15, n_components=2)
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(7,3))
ax[0].scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],color='red', marker='^', alpha=0.
 ↪5)
ax[0].scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],color='blue', marker='o',␣
 ↪alpha=0.5)
ax[1].scatter(X_kpca[y==0, 0], np.zeros((500,1))+0.02,color='red', marker='^',␣
 ↪alpha=0.5)
ax[1].scatter(X_kpca[y==1, 0], np.zeros((500,1))-0.02,color='blue', marker='o',␣
 ↪alpha=0.5)
ax[0].set_xlabel('PC1')
```

```
ax[0].set_ylabel('PC2')
ax[1].set_ylim([-1, 1])
ax[1].set_yticks([])
ax[1].set_xlabel('PC1')
plt.show()
```

```
C:\Users\ankit19.gupta\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_R
aschka\venv_python_3.6\lib\site-packages\ipykernel_launcher.py:27:
DeprecationWarning: scipy.exp is deprecated and will be removed in SciPy 2.0.0,
use numpy.exp instead
C:\Users\ankit19.gupta\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_R
aschka\venv_python_3.6\lib\site-packages\ipykernel_launcher.py:37:
FutureWarning: arrays to stack must be passed as a "sequence" type such as list
or tuple. Support for non-sequence iterables such as generators is deprecated as
of NumPy 1.16 and will raise an error in the future.
```



1.4 Projecting new data points

[61]:
```
from scipy.spatial.distance import pdist, squareform
from scipy import exp
from scipy.linalg import eigh
import numpy as np
def rbf_kernel_pca(X, gamma, n_components):
    """
    RBF kernel PCA implementation.
    Parameters
    ------------
    X: {NumPy ndarray}, shape = [n_samples, n_features]
    gamma: float
    Tuning parameter of the RBF kernel
```

```
    n_components: int
    Chapter 5
    [ 181 ]
    Number of principal components to return
    Returns
    ------------
    X_pc: {NumPy ndarray}, shape = [n_samples, k_features]
    Projected dataset
    lambdas: list
    Eigenvalues
    """
    # Calculate pairwise squared Euclidean distances
    # in the MxN dimensional dataset.
    sq_dists = pdist(X, 'sqeuclidean')
    # Convert pairwise distances into a square matrix.
    mat_sq_dists = squareform(sq_dists)
    # Compute the symmetric kernel matrix.
    K = exp(-gamma * mat_sq_dists)
    # Center the kernel matrix.
    N = K.shape[0]
    one_n = np.ones((N,N)) / N
    K = K - one_n.dot(K) - K.dot(one_n) + one_n.dot(K).dot(one_n)
    # Obtaining eigenpairs from the centered kernel matrix
    # scipy.linalg.eigh returns them in ascending order
    eigvals, eigvecs = eigh(K)
    eigvals, eigvecs = eigvals[::-1], eigvecs[:, ::-1]
    # Collect the top k eigenvectors (projected samples)
    alphas = np.column_stack((eigvecs[:, i] for i in range(n_components)))
    # Collect the corresponding eigenvalues
    lambdas = [eigvals[i] for i in range(n_components)]
    return alphas, lambdas
```

[62]:
```
X, y = make_moons(n_samples=100, random_state=123)
alphas, lambdas = rbf_kernel_pca(X, gamma=15, n_components=1)
```

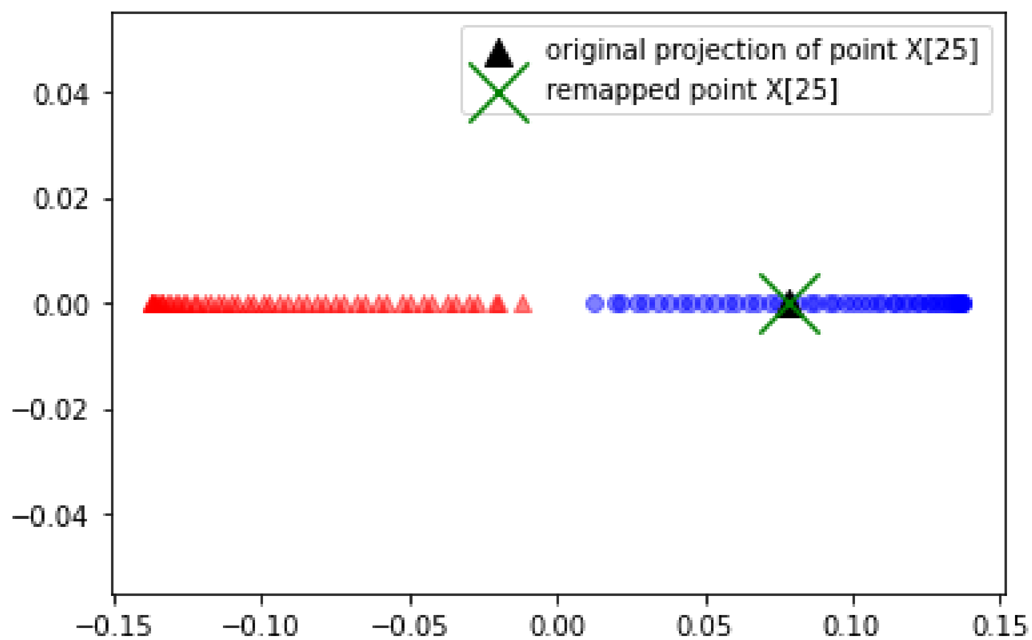
```
C:\Users\ankit19.gupta\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_R
aschka\venv_python_3.6\lib\site-packages\ipykernel_launcher.py:30:
DeprecationWarning: scipy.exp is deprecated and will be removed in SciPy 2.0.0,
use numpy.exp instead
C:\Users\ankit19.gupta\Desktop\Self_Projects\Python_Machine_Learning_Sebastian_R
aschka\venv_python_3.6\lib\site-packages\ipykernel_launcher.py:40:
FutureWarning: arrays to stack must be passed as a "sequence" type such as list
or tuple. Support for non-sequence iterables such as generators is deprecated as
of NumPy 1.16 and will raise an error in the future.
```

[63]:
```
x_new = X[25]
x_proj = alphas[25] # original projection
def project_x(x_new, X, gamma, alphas, lambdas):
```

```
    pair_dist = np.array([np.sum((x_new-row)**2) for row in X])
    k = np.exp(-gamma * pair_dist)
    return k.dot(alphas / lambdas)
```

[64]:
```
x_reproj = project_x(x_new, X,gamma=15, alphas=alphas, lambdas=lambdas)
```

[65]:
```
plt.scatter(alphas[y==0, 0], np.zeros((50)),color='red', marker='^',alpha=0.5)
plt.scatter(alphas[y==1, 0], np.zeros((50)),color='blue', marker='o', alpha=0.5)
plt.scatter(x_proj, 0, color='black',label='original projection of point␣
 ↪X[25]',marker='^', s=100)
plt.scatter(x_reproj, 0, color='green',label='remapped point X[25]',marker='x',␣
 ↪s=500)
plt.legend(scatterpoints=1)
plt.show()
```

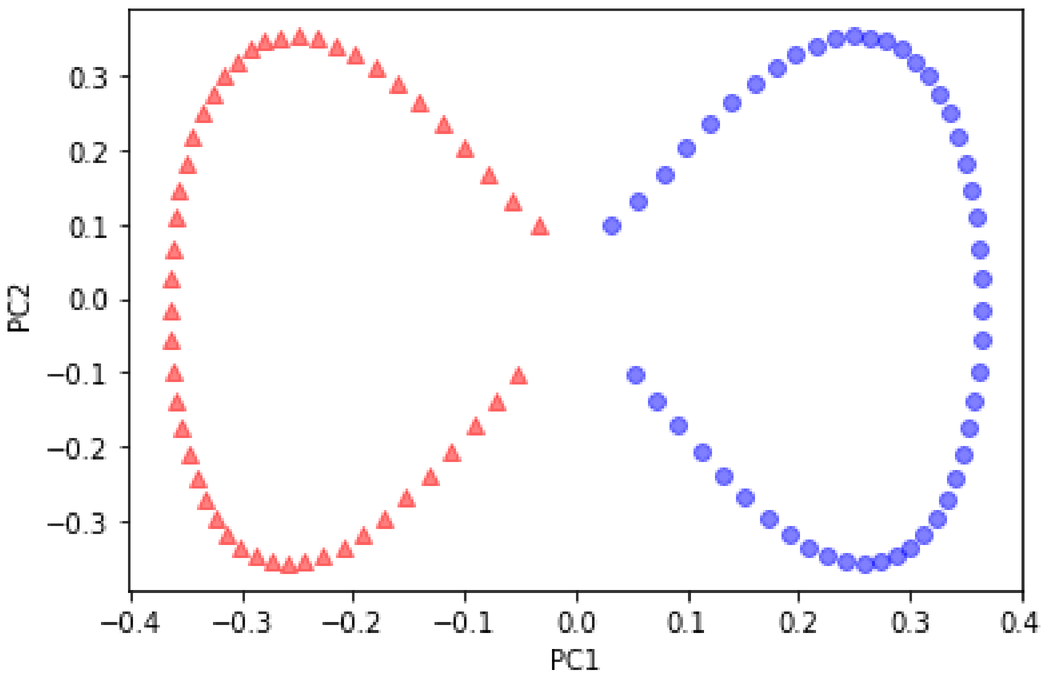


1.5 Kernel principal component analysis in scikit-learn

[66]:
```
from sklearn.decomposition import KernelPCA
X, y = make_moons(n_samples=100, random_state=123)
scikit_kpca = KernelPCA(n_components=2,kernel='rbf', gamma=15)
X_skernpca = scikit_kpca.fit_transform(X)
```

[67]:
```
plt.scatter(X_skernpca[y==0, 0], X_skernpca[y==0, 1],color='red', marker='^',␣
 ↪alpha=0.5)
```

```
plt.scatter(X_skernpca[y==1, 0], X_skernpca[y==1, 1],color='blue', marker='o',␣
↪alpha=0.5)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```



[ ]:

```

```