NEW

May 26, 2016

1 PCA as a tool for preprocessing and Kernel PCA

This problem sheet explores applications and extensions of PCA. The first two exercises deal with PCA as a method for preprocessing and the third one illustrates how to find nonlinear structure via kernel PCA.

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg
import scipy.spatial.distance
import pandas as pd
from PIL import Image
%matplotlib inline

!python --version
```

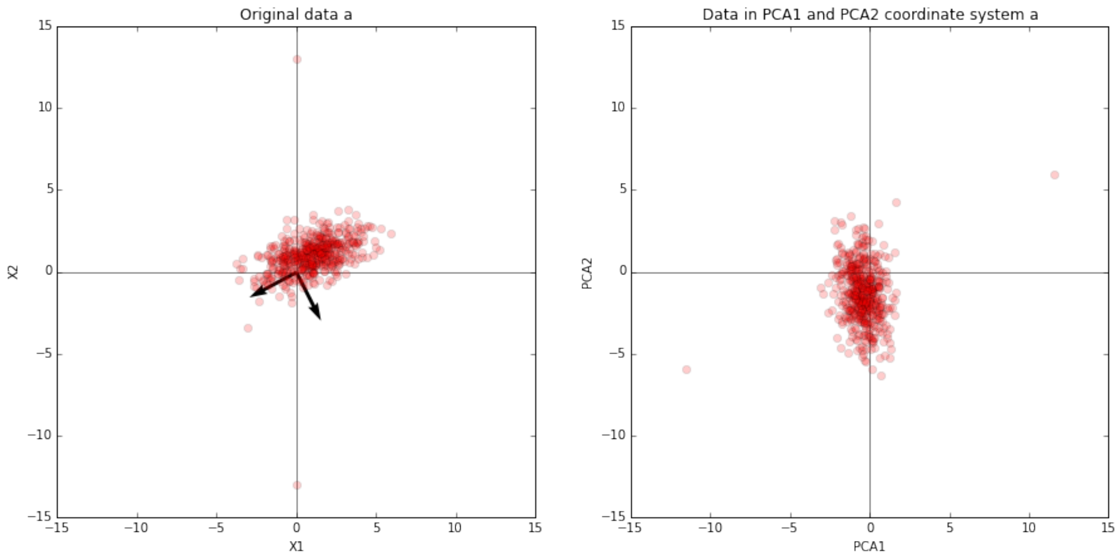
```
Python 3.4.3 :: Anaconda 2.3.0 (x86_64)
```

1.1 4.1 Preprocessing

1. Load the dataset pca2.csv. Compte the Principal Components PC1 and PC2 and plot the data in the coordinate system PC1 vs. PC2 - What do you observe?
2. Remove Observations 17 and 157 and redo the first two steps. What is the difference?

In [10]:
```
# (a)
pca2 = np.loadtxt(open("pca2.csv"),delimiter=",",skiprows=1)

pca2_centered = pca2 - pca2.mean(0)
cov = np.cov(pca2_centered.T)

# eigen-values and -vectors
values, vectors = np.linalg.eigh(cov)

fig = plt.figure(figsize=(15,7))
plt.subplot(1,2,1)
plt.axhline(0, color="black", alpha=0.5)
plt.axvline(0, color="black", alpha=0.5)
plt.title("Original data a")
plt.plot(pca2[:,0], pca2[:,1], 'ro', alpha=0.2)
plt.quiver(0,0,vectors[0][0],vectors[0][1],angles='xy',scale_units='xy',scale=0.3)
plt.quiver(0,0,vectors[1][0],vectors[1][1],angles='xy',scale_units='xy',scale=0.3)
plt.xlabel("X1")
plt.ylabel("X2")
plt.axis([-15, 15, -15, 15])
```

```
# transform the data into the pcas coordinate system
pca2_data_transormed = np.dot(pca2, vectors)

plt.subplot(1,2,2)
plt.axhline(0, color="black", alpha=0.5)
plt.axvline(0, color="black", alpha=0.5)
plt.title("Data in PCA1 and PCA2 coordinate system a")
plt.plot(pca2_data_transormed[:,0],pca2_data_transormed[:,1] ,'ro', alpha=0.2)
plt.xlabel("PCA1")
plt.ylabel("PCA2")
plt.axis([-15, 15, -15, 15])

plt.show()
```

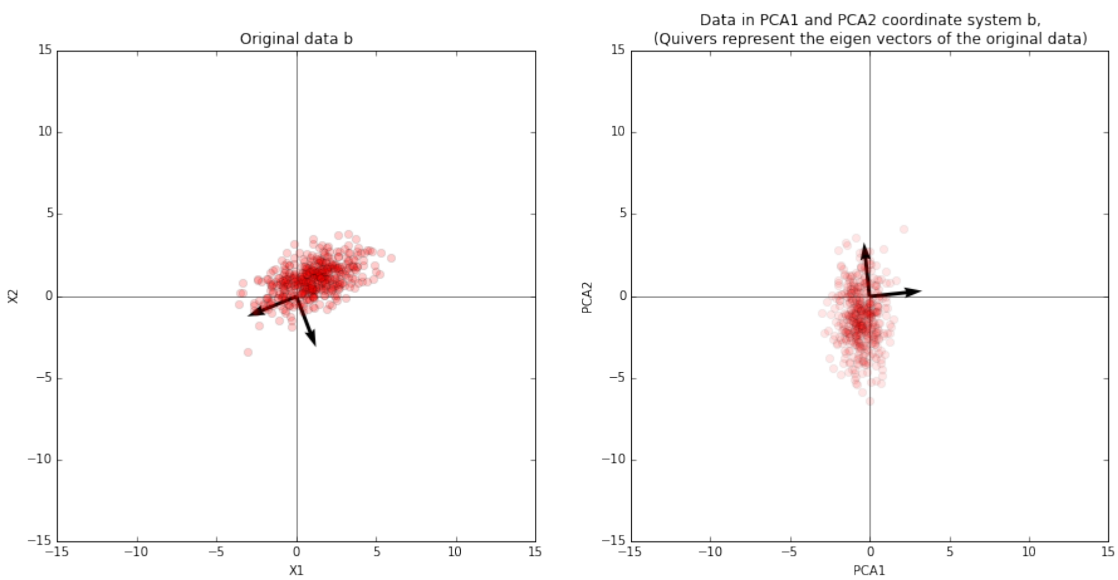


Observation:

When transformed into the PCA1 - PCA2 coordinate system, one would think the data would fit along the a

```
In [30]: # (a)
         pca2 = np.delete(pca2, [16,156],0)

         center = pca2.mean(0)
         pca2_centered = pca2 - center

         cov = np.cov(pca2_centered.T)

         # eigen-values and -vectors
         values2, vectors2 = np.linalg.eigh(cov)

         fig = plt.figure(figsize=(15,7))

         plt.subplot(1,2,1)
         plt.axhline(0, color="black", alpha=0.5)
         plt.axvline(0, color="black", alpha=0.5)
```

```python
plt.title("Original data b")
plt.plot(pca2[:,0], pca2[:,1], 'ro', alpha=0.2)
plt.quiver(0,0,vectors2[0][0],vectors2[0][1],angles='xy',scale_units='xy',scale=0.3)
plt.quiver(0,0,vectors2[1][0],vectors2[1][1],angles='xy',scale_units='xy',scale=0.3)
plt.xlabel("X1")
plt.ylabel("X2")
plt.axis([-15, 15, -15, 15])

# tranform data into pcas coordinate system
pca2_data_transormed = np.dot(pca2, vectors2)

x = np.empty([len(pca2), 2])
for i in range (len(pca2)):
    x[i,:]=np.dot(pca2[i,:],vectors2)

vectors_transformed = np.dot(vectors2, vectors)


plt.subplot(1,2,2)
plt.axhline(0, color="black", alpha=0.5)
plt.axvline(0, color="black", alpha=0.5)
plt.title("Data in PCA1 and PCA2 coordinate system b, \n(Quivers represent the eigen vectors o
plt.plot(pca2_data_transormed[:,0],pca2_data_transormed[:,1] ,'ro', alpha=0.1)
plt.quiver(0,0,vectors_transformed[0][0],vectors_transformed[0][1],angles='xy',scale_units='xy
plt.quiver(0,0,vectors_transformed[1][0],vectors_transformed[1][1],angles='xy',scale_units='xy
plt.xlabel("PCA1")
plt.ylabel("PCA2")
plt.axis([-15, 15, -15, 15])

plt.show()
```



We observe that the data in coordinate system b is rotated a little bit from the data in a. The quivers represent the eigen vectors of the data from a. Them being off from the x and y axis show this rotation.

1.2 4.2 Whitening

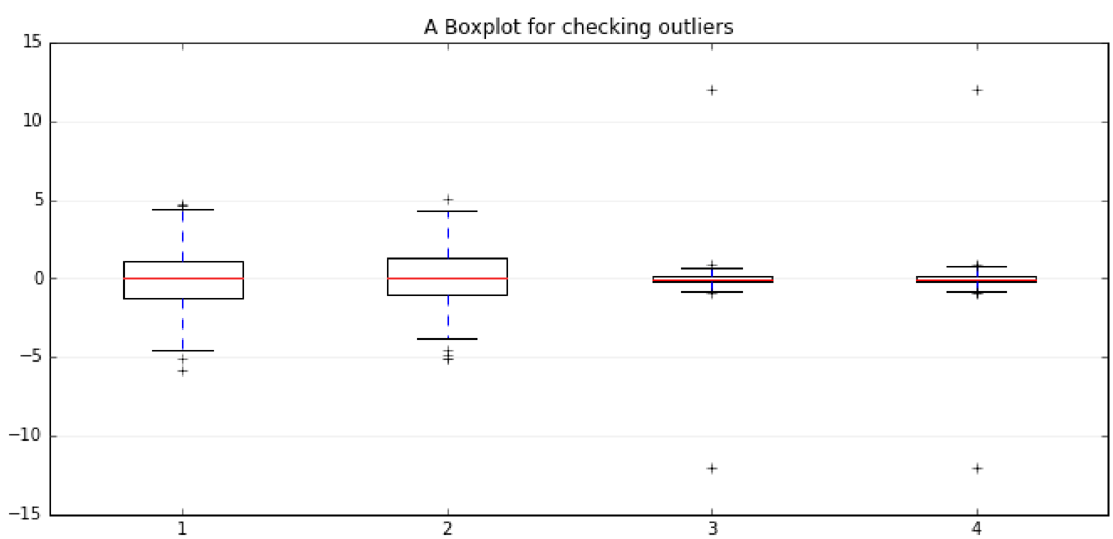
1. Load the dataset pca4.csv and check for outliers in the individual variables.
2. Do PCA on a reasonable subset of this data. Use a scree plot to determine how many PCs represent the data well.
3. "Whiten" the data, i.e. create a set of 4 *uncorrelated* variables with *mean 0* and *standard deviation equal to 1.* This can be done e.g. using the transformation

$$Z = X \ ED^{-\frac{1}{[}2]}$$

   The new variables zi form the columns of Z, E is a matrix containing the normalized eigenvectors of the covariance matrix Sigma of the centered data X and D is a diagonal matrix containing the corresponding eigenvalues.

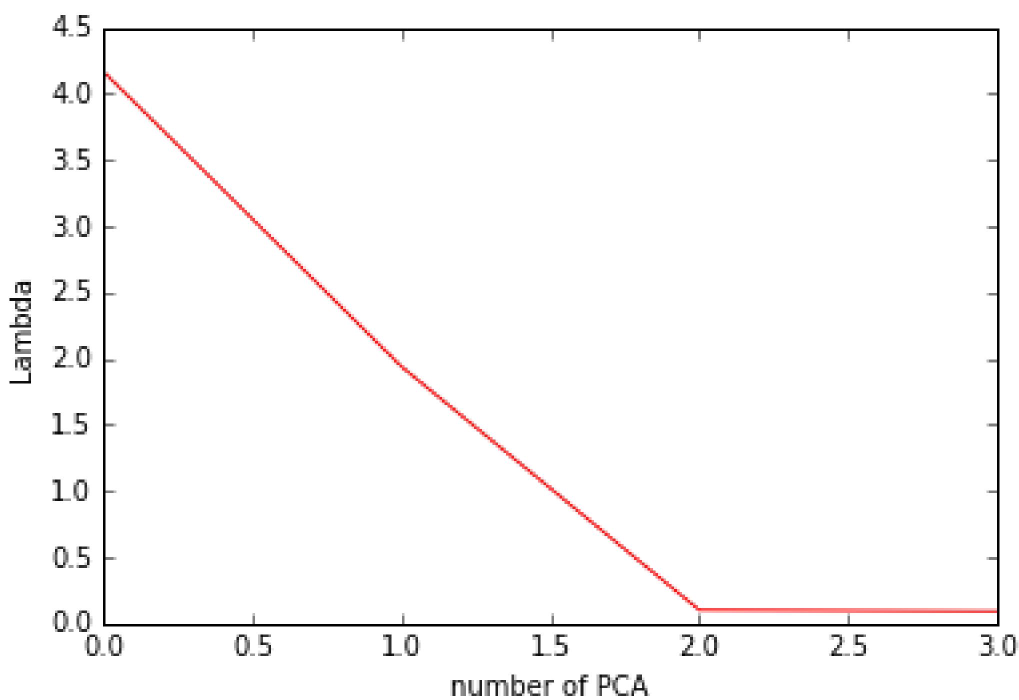
4. Make 3 heat plots of the (i) 4x4 covariance matrix Sigma, (ii) the covariance matrix of the data projected onto PC1-PC4, and (iii) of the whitened variables.

```
In [5]: # (a)
        pca4 = np.loadtxt(open("pca4.csv"),delimiter=",",skiprows=1)


        fig, ax1 = plt.subplots(figsize=(10,6))
        plt.title('A Boxplot for checking outliers')
        plt.subplots_adjust(left=0.075, right=0.95, top=0.9, bottom=0.25)

        bp = plt.boxplot(pca4, notch=0, sym='+', vert=1, whis=1.5)
        plt.setp(bp['boxes'], color='black')
        plt.setp(bp['whiskers'], color='blue')
        plt.setp(bp['fliers'], color='green', marker='+')
        ax1.yaxis.grid(True, linestyle='-', which='major', color='lightgrey',alpha=0.5)
        plt.show()
```



Interpretation: In X1 and X2 the data is spread out widely. Outliers are still visible according to the boxplots, but there are located very near to the whiskers. We won't treat these values as outliers. In X3 and X4 however, the data is much more dense. Four observations lie far off the rest of the data. These are clearly outliers.

In [6]:
```python
# (b)
# first we chose a subset: complete data except outliers
outliers = []
for i in range(pca4.shape[0]) :
    if (abs(pca4[i][2]) > 10 or abs(pca4[i][3]) > 10) :
        outliers.append(i)

pca4_subset = np.delete(pca4, outliers, axis=0)

pca4_sub_centerd = pca4_subset - pca4_subset.mean(0)

pca4_sub_cov = np.cov(pca4_sub_centerd.T)

val_sub, vec_sub = np.linalg.eig(pca4_sub_cov)

line = plt.plot(val_sub, color='r')
plt.xlabel('number of PCA')
plt.ylabel('Lambda')
plt.show()
print "as we can see in the graph, we only need the first and second PCs to represeent the data
```

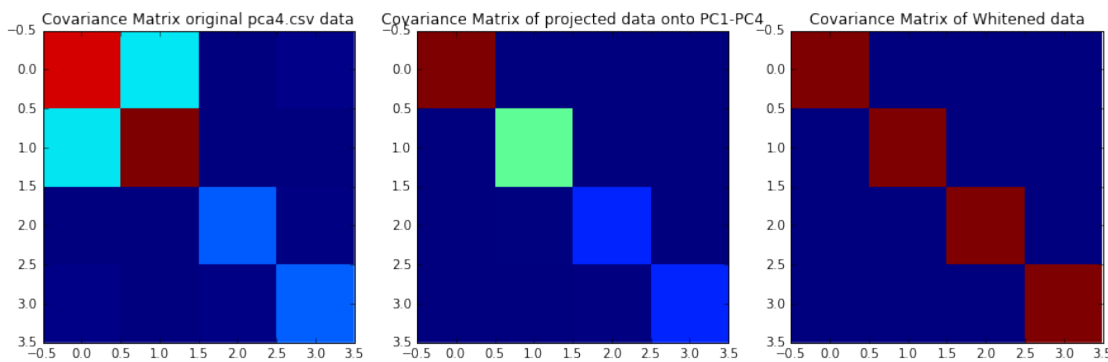



```
as we can see in the graph, we only need the first and second PCs to represeent the data well
```

In [7]:
```python
# (c)
# E matrix of eigenvectors
pca4_centerd = pca4 - pca4.mean(0)

pca4_cov = np.cov(pca4_centerd.T)
```

```
val_pca4, vec_pca4 = np.linalg.eig(pca4_cov)

# eigenvalues and vector are sorted
E = vec_pca4.copy()

# eigenvalues
D = np.diag(val_pca4)

Z = (pca4_centerd.dot(E)).dot(np.diag(1/np.sqrt(val_pca4)))
```

In [8]:
```
# (d) Make 3 heat-plots of the
#     (i) 4x4 covariance matrix E,
#     (ii) the covariance matrix of the data projected onto PC1-PC4, and
#     (iii) of the whitened variables.

proj = pca4.dot(vec_pca4)
proj_cov = np.cov(proj.T)

fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(1,3,1)
ax1.imshow(pca4_cov, interpolation='none')
ax1.set_title("Covariance Matrix original pca4.csv data")
ax2 = fig.add_subplot(1,3,2)
ax2.imshow(proj_cov, interpolation='none')
ax2.set_title("Covariance Matrix of projected data onto PC1-PC4")
ax3 = fig.add_subplot(1,3,3)
ax3.imshow(np.cov(Z.T), interpolation='none')
ax3.set_title("Covariance Matrix of Whitened data")
plt.show()
```

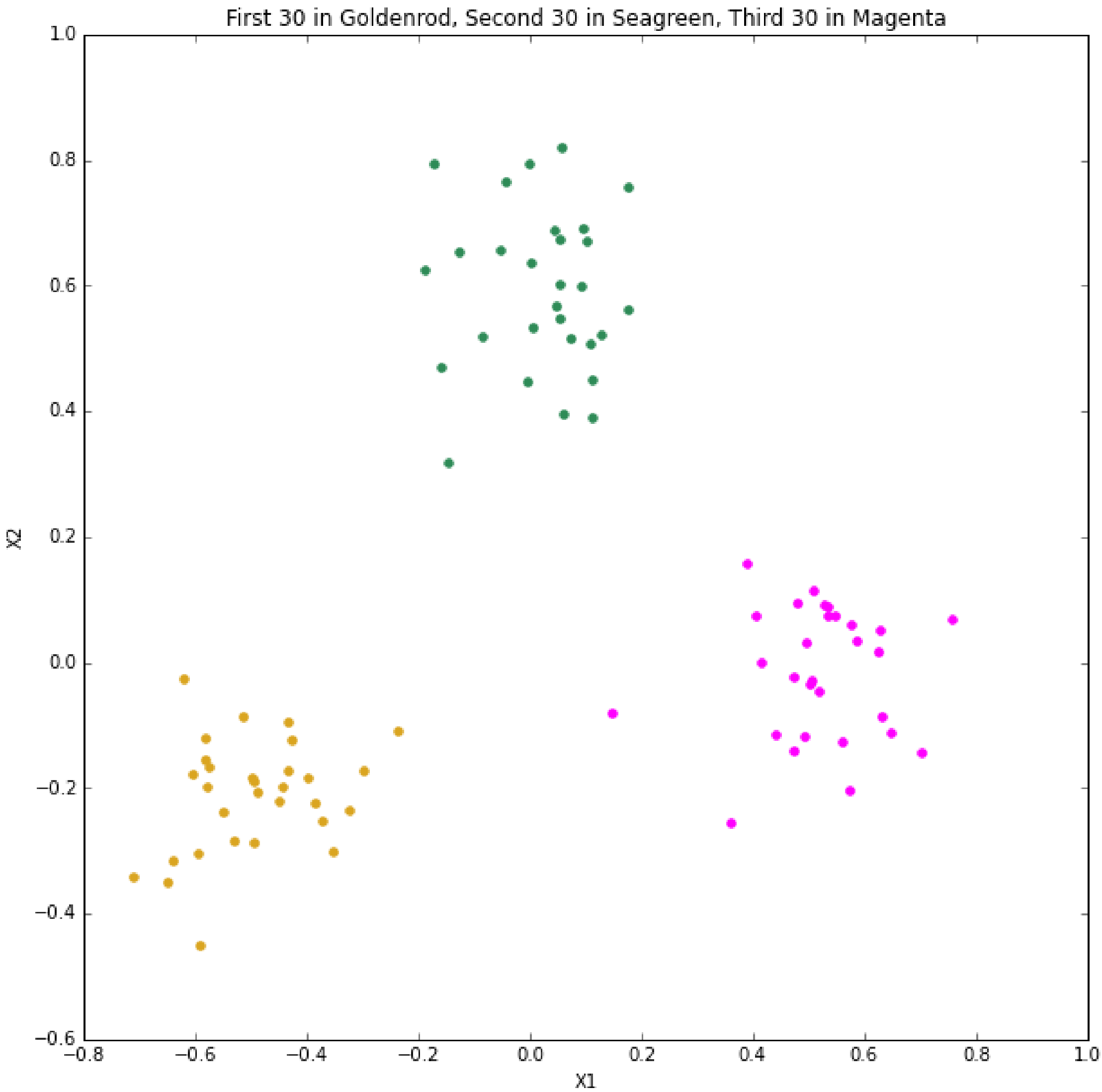


1.3 4.3 Kernel PCA: Toy Data

1. Create a toy dataset of 2-dimensional data points …

In [6]:
```
data = np.zeros((2,90))
dataVis = np.zeros((2,7500))
# center: (-0.5, -0.2)
data[0,0:30] = np.random.normal(-0.5, 0.1, 30)
```

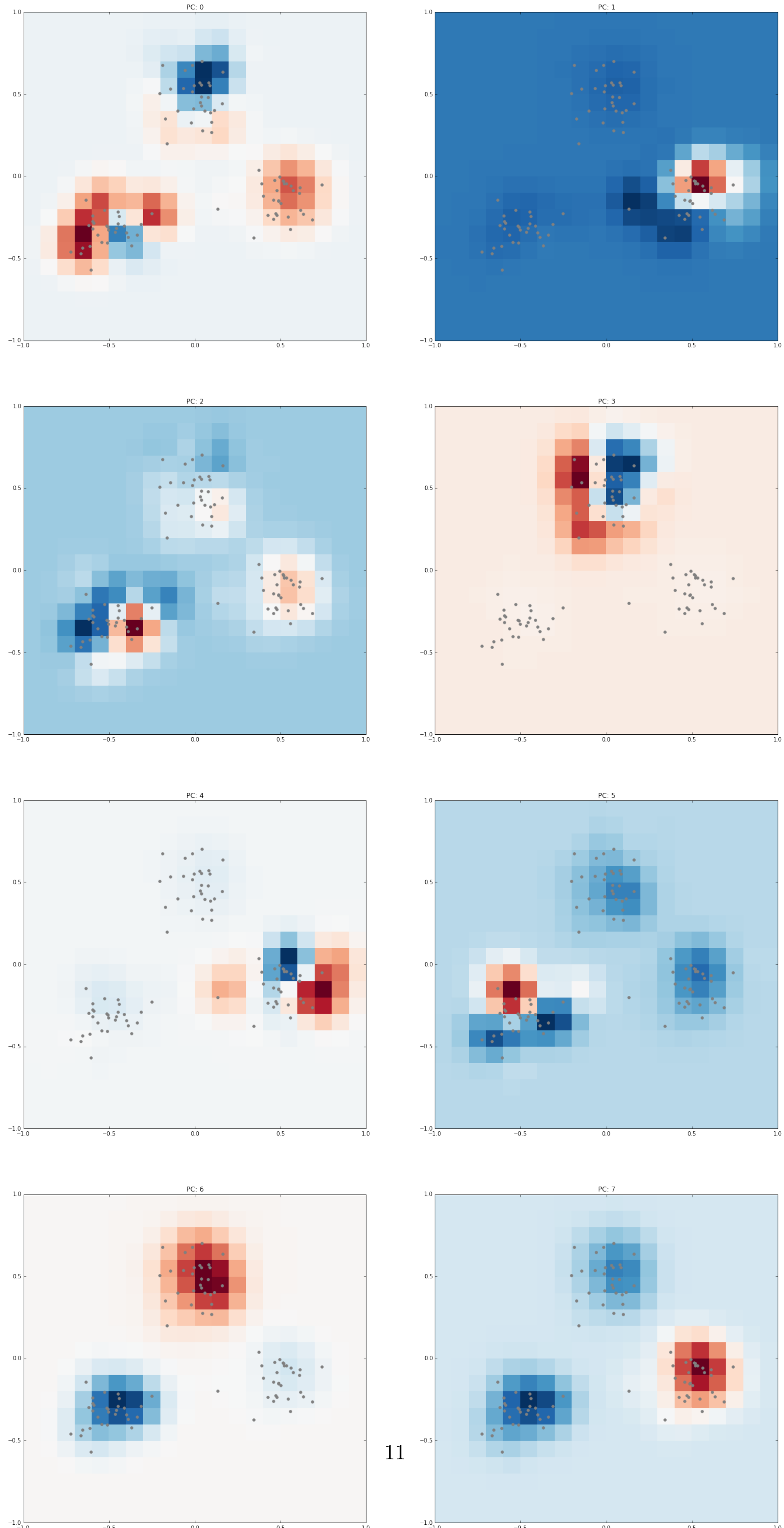
```python
data[1,0:30] = np.random.normal(-0.2, 0.1, 30)
dataVis[0,0:2500] = np.random.normal(-0.5, 0.1, 2500)
dataVis[1,0:2500] = np.random.normal(-0.2, 0.1, 2500)

# center: (0.0, 0.1)
data[0,30:60] = np.random.normal(0.0, 0.1, 30)
data[1,30:60] = np.random.normal(0.6, 0.1, 30)
dataVis[0,2500:5000] = np.random.normal(0.0, 0.1, 2500)
dataVis[1,2500:5000] = np.random.normal(0.6, 0.1, 2500)

# center: (-0.5, 0)
data[0,60:90] = np.random.normal(0.5, 0.1, 30)
data[1,60:90] = np.random.normal(0.0, 0.1, 30)
dataVis[0,5000:7500] = np.random.normal(0.5, 0.1, 2500)
dataVis[1,5000:7500] = np.random.normal(0.0, 0.1, 2500)

plt.figure(figsize=(10,10))
plt.scatter(data[0,0:30],data[1,0:30], color='goldenrod')
plt.scatter(data[0,31:60],data[1,31:60], color='seagreen')
plt.scatter(data[0,61:90],data[1,61:90], color='magenta')
ax = plt.gca()
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_title('First 30 in Goldenrod, Second 30 in Seagreen, Third 30 in Magenta')
plt.show()
```

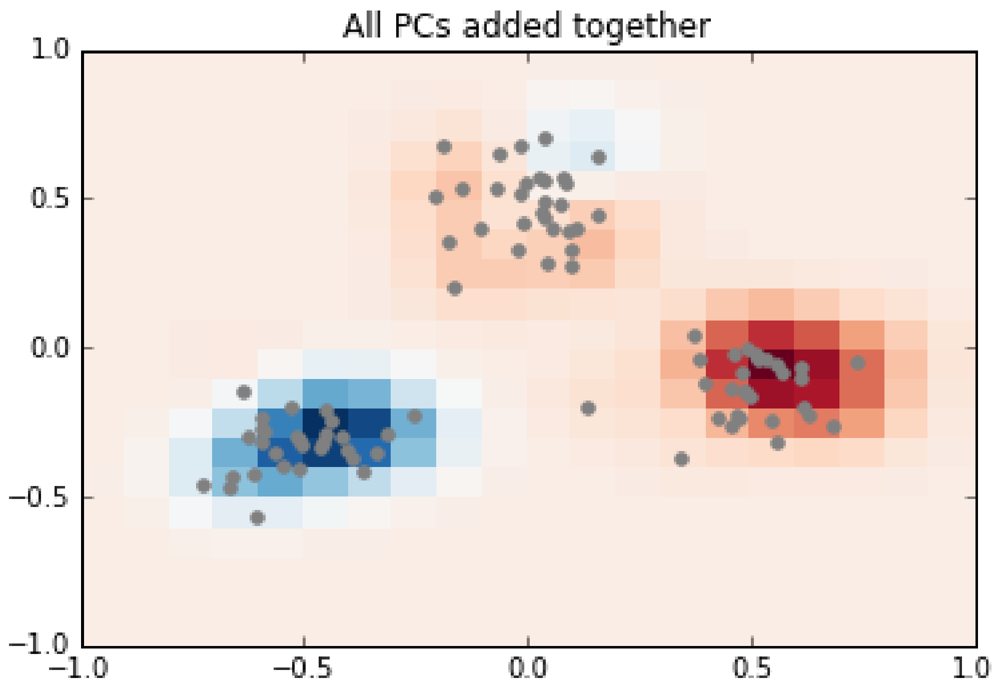



```
In [190]: #(b)

          #(0) center data
          #print(np.mean(data.T, axis=0).shape)
          datac = (data - np.mean(data.T, axis=0).reshape(-1,1)).T

          #(1) calculate the unnormalized kernel matrix
          sigma = .1
          p = datac.shape[0]
          #pxp
          Khat = np.zeros( (p, p) )

          for a in range(p):
              for b in range(p):
                  Khat[a, b] = np.exp(- (np.power(np.linalg.norm(datac[b] - datac[a]), 2))/(2*np.power(s
```

```
#(2) normalize to 0 mean
#K = K - rowavg - colavg + matrixavg
K = Khat - Khat.sum(axis=0)/p - Khat.sum(axis=1)/p + Khat.sum()/np.power(p,2)

#(3) solve the eigenvalue problem
D, U = np.linalg.eigh(K)
Dsub = D[-8:][::-1]
Usub = U[:,-8:][::-1]

#(4) normalize eigen vectors to unit length
Unorm = Usub / (np.sqrt(p * Dsub) * np.linalg.norm(Usub))

#(5) project onto eigen vectors
proj = np.dot(Unorm.T, K)

#(c)
datam = np.zeros((1,2))
xs = np.arange(-1,1.1,.1)
ys = np.arange(-1,1.1,.1)
for x in xs:
    for y in ys:
        datam = np.vstack( (datam, [x, y]) )
datam = datam[1:]

Kbd = np.zeros( (p, len(datam)) )

for b in range(p):
    for d in range(len(datam)):
        Kbd[b, d] = np.exp(- (np.power(np.linalg.norm(datac[b] - datam[d]), 2))/(2*np.power(s
rowavg = Kbd.sum(axis=0)/Kbd.shape[1]
colavg = Kbd.sum(axis=1)/Kbd.shape[0]
matrixavg = Kbd.sum()/(Kbd.shape[0]*Kbd.shape[1])

for b in range(Kbd.shape[0]):
    Kbd[b,:] -= rowavg
for d in range(Kbd.shape[1]):
    Kbd[:,d] -= colavg
for b in range(Kbd.shape[0]):
    for d in range(Kbd.shape[1]):
        Kbd[b, d] += matrixavg

datap = np.dot(Unorm.T, Kbd)

fig, axarray = plt.subplots(4,2)
fig.set_figheight(48)
fig.set_figwidth(24)


for i in range(8):
    axarray[np.floor(i/2),i%2].pcolor(np.repeat(xs.reshape(-1,1), 21, axis=1), np.repeat(ys.r
    axarray[np.floor(i/2),i%2].scatter(datac.T[0], datac.T[1], color="grey")
    axarray[np.floor(i/2),i%2].set_title('PC: '+str(i))
    axarray[np.floor(i/2),i%2].axis([-1,1,-1,1])
```

```
plt.show()

plt.pcolor(np.repeat(xs.reshape(-1,1), 21, axis=1), np.repeat(ys.reshape(1,-1), 21, axis=0), c
plt.scatter(datac.T[0], datac.T[1], color="grey")
plt.title('All PCs added together')
plt.axis([-1,1,-1,1])
plt.show()
```


PC: 0
PC: 1
PC: 2
PC: 3
PC: 4
PC: 5
PC: 6
PC: 7



Each different PC seems to show a cluster.