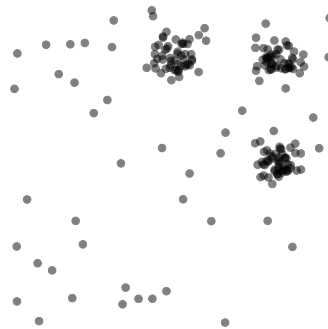


**project 3:****clustering and dimensionality reduction****solution(s) due:**

**Feb 4, 2016 at 12:00** via email to **bauckhag@bit.uni-bonn.de**

**problem specification:**

**task 3.1: fun with k-means clustering:** Download the 2D data in the file `data-clustering-1.csv` and plot it; the result should look something like this



Next, implement

- Lloyd's algorithm for  $k$ -means clustering (e.g. simply using `scipy`)
- Hartigan's algorithm for  $k$ -means clustering
- MacQueen's algorithm for  $k$ -means clustering

For  $k = 3$ , run each algorithm on the above data and plot your results. In fact, run each of them several times and look at the results. What do you observe? Are the results always the same or do they vary from run to run?

Measure the run times of each of your implementations (run them each at least 10 times and determine their average run times). What do you observe?

**task 3.2: spectral clustering:** Download `data-clustering-2.csv` and plot the 2D data in this file; the result should look something like this



Set  $k = 2$  and apply the  $k$ -means algorithms you implemented in the previous task to this data. What do you observe?

Next, implement *spectral clustering*. Proceed as follows: Let

$$X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^2$$

be the given data. First, compute an  $n \times n$  *similarity matrix*  $S$  where

$$S_{ij} = e^{-\beta \|\mathbf{x}_i - \mathbf{x}_j\|^2}$$

and then compute the *Laplacian matrix*  $L = D - S$  where the diagonal matrix  $D$  is given by

$$D_{ij} = \begin{cases} \sum_j S_{ij} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Note that row  $i$  in  $L$  can be understood as a feature vector  $f(\mathbf{x}_i)$  for data point  $\mathbf{x}_i$ . Next, compute the eigenvalues  $\lambda_i$  and eigenvectors  $\mathbf{u}_i$  of  $L$  and sort them in descending order. That is, let  $\lambda_n$  denote the largest eigenvalue and  $\mathbf{u}_n$  denote the corresponding eigenvector.

The eigenvector  $\mathbf{u}_2$  that corresponds to the second smallest eigenvalue  $\lambda_2$  is called the *Fiedler vector* and is of significance in clustering. You will find that some of its entries are greater than 0 and some are less than 0. To cluster the given data into two clusters  $C_1$  and  $C_2$ , do the following: If entry  $i$  of  $\mathbf{u}_2$  is greater than 0 assign  $\mathbf{x}_i$  to cluster  $C_1$ , if it is less than zero, assign  $\mathbf{x}_i$  to cluster  $C_2$ .

Set  $\beta$  to some value (say  $\beta = 1$ ), cluster the data as described, and plot your results. What do you observe?

**task 3.3: dimensionality reduction:** The file `data-dimred-X.csv` contains a  $500 \times 150$  data matrix  $X$ , that is, 150 data vectors  $x_i \in \mathbb{R}^{500}$ .

In fact, these data vectors are from three classes and you can find the corresponding class labels  $y_i \in \{1, 2, 3\}$  in the file `data-dimred-y.csv`.

The goal of this task is to explore mappings  $\mathbb{R}^{500} \rightarrow \mathbb{R}^2$  that allow us to visualize (plot) high-dimensional data.

First of all, perform dimensionality reduction using *PCA*. That is, normalize the data in  $X$  to zero mean and compute the eigen-decomposition of the corresponding covariance matrix. Then, use the two eigenvectors  $u_1$  and  $u_2$  of the two largest eigenvalues to project the (normalized!) data into  $\mathbb{R}^2$ . What do you observe?

Second of all, perform dimensionality reduction using *multiclass LDA*. To this end, make use of the fact that the data in  $X$  are from three classes. Compute the within class scatter matrix  $S_W$  and the between class scatter matrix  $S_B$  and then the eigen-decomposition of the matrix  $S_W^{-1}S_B$ . Again, use the two eigenvectors  $u_1$  and  $u_2$  of the two largest eigenvalues to project the data into  $\mathbb{R}^2$ . What do you observe? Does the result differ from the one you obtained via PCA?

What if you project the data from  $\mathbb{R}^{500}$  to  $\mathbb{R}^3$ ? For both approaches, this can be accomplished using the first three eigenvectors and creating 3D plots.

**task 3.4: exploring numerical instabilities:** This task revisits task 2.1 and is not such much a task in itself but an eye opener! **Everybody, i.e. every member of each team, must do it!**

Download the file `whData.dat`, remove the outliers and collect the remaining height and weight data in two `numpy` arrays `hgt` and `wgt` and fit a 10-th order polynomial. Use the following code:

```
import numpy as np
import numpy.linalg as la
import numpy.polynomial.polynomial as poly
import matplotlib.pyplot as plt

hgt = ...
wgt = ...

xmin = hgt.min()-15
xmax = hgt.max()+15
ymin = wgt.min()-15
ymax = wgt.max()+15
```

```
def plot_data_and_fit(h, w, x, y):
    plt.plot(h, w, 'ko', x, y, 'r-')
    plt.xlim(xmin, xmax)
    plt.ylim(ymin, ymax)
    plt.show()

def trsf(x):
    return x / 100.

n = 10
x = np.linspace(xmin, xmax, 100)

# method 1:
# regression using ployfit
c = poly.polyfit(hgt, wgt, n)
y = poly.polyval(x, c)
plot_data_and_fit(hgt, wgt, x, y)

# method 2:
# regression using the Vandermonde matrix and pinv
X = poly.polyvander(hgt, n)
c = np.dot(la.pinv(X), wgt)
y = np.dot(poly.polyvander(x, n), c)
plot_data_and_fit(hgt, wgt, x, y)

# method 3:
# regression using the Vandermonde matrix and lstsq
X = poly.polyvander(hgt, n)
c = la.lstsq(X, wgt)[0]
y = np.dot(poly.polyvander(x, n), c)
plot_data_and_fit(hgt, wgt, x, y)

# method 4:
# regression on transformed data using the Vandermonde
# matrix and either pinv or lstsq
X = poly.polyvander(trsf(hgt), n)
c = np.dot(la.pinv(X), wgt)
y = np.dot(poly.polyvander(trsf(x), n), c)
plot_data_and_fit(hgt, wgt, x, y)
```

What is going on here? Report what you observe! Think about what this implies! What if you were working in aerospace engineering where sloppy code and careless implementations could have catastrophic results ...

## **general hints and remarks**

- see previous project sheets