project 3:

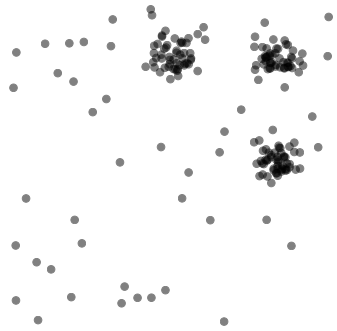
clustering, dimensionality reduction, and non-monotonous neurons

solution(s) due:

Jul 1, 2019 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



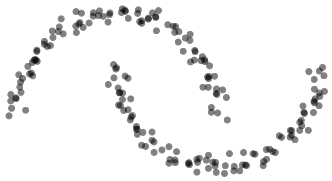
Next, implement

- 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

$$X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^2$$

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

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

and then compute the *Laplacian matrix* $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{S}$ where the diagonal matrix $\boldsymbol{D}$ 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 $\boldsymbol{L}$ can be understood as a feature vector $\boldsymbol{f}(\boldsymbol{x}_i)$ for data point $\boldsymbol{x}_i$. Next, compute the eigenvalues λ_i and eigenvectors $\boldsymbol{u}_i$ of $\boldsymbol{L}$ and sort them in descending order. That is, let λ_n denote the largest eigenvalue and $\boldsymbol{u}_n$ denote the corresponding eigenvector.

The eigenvector $\boldsymbol{u}_2$ that corresponds to the second smallest eigenvalue λ_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 $\boldsymbol{u}_2$ is greater than 0 assign $\boldsymbol{x}_i$ to cluster C_1, it it is less than zero, assign $\boldsymbol{x}_i$ to cluster C_2.

Set β 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 500×150 data matrix $\boldsymbol{X}$, that is, 150 data vectors $\boldsymbol{x}_i \in \mathbb{R}^{500}$.

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`.

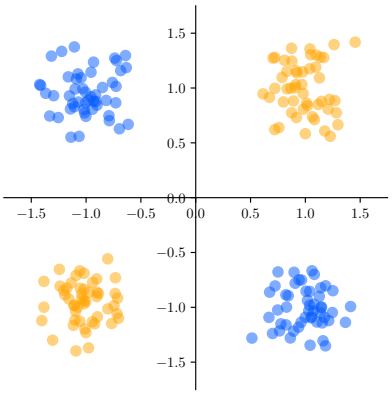
The goal of this task is to explore mappings $\mathbb{R}^{500} \rightarrow \mathbb{R}^2$ that allow us to visualize (plot) high-dimensional data.

First of all, perform dimensionality reduction using *PCA*. That is, normalize the data in $\boldsymbol{X}$ to zero mean and compute the eigen-decomposition of the corresponding covariance matrix. Then, use the two eigenvectors $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ 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* (as discussed in lecture 14). To this end, make use of the the fact that the data in $\boldsymbol{X}$ are from three classes. Compute the within class scatter matrix $\boldsymbol{S}_W$ and the between class scatter matrix $\boldsymbol{S}_B$ and then the eigen-decomposition of the matrix $\boldsymbol{S}_W^{-1}\boldsymbol{S}_B$. Again, use the two eigenvectors $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ 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: non-monotonous neurons: The two files `xor-X.csv` and `xor-y.csv` contain data points $\boldsymbol{x}_i \in \mathbb{R}^2$ and label values $y_i \in \{-1, +1\}$ which when plotted appropriately should lead to a picture like this



Note that XOR problems like this pose nonlinear classification problems, because there is no single hyperplane that would separate the blue from the orange dots. XOR problems are therefore famously used to prove the limitations of a single perceptron

$$y(\boldsymbol{x}) = f\big(\boldsymbol{w}^T\boldsymbol{x} - \theta\big)$$

where f is a monotonous activation function such as

$$f(z) = \tanh\big(\beta z\big).$$

However, this limitation is a historical artifact, because monotonous activation functions are a persistent *meme* that arose in the 1940s. Yet, there is nothing that would prevent us from considering more flexible neural networks composed of neurons with non-monotonous activations.

Mandatory sub-task: Given the above data, train a non-monotonous neuron

$$y(\boldsymbol{x}) = f\big(\boldsymbol{x}, \boldsymbol{w}, \theta\big)$$

where

$$f\big(\boldsymbol{x}, \boldsymbol{w}, \theta\big) = 2 \cdot \exp\left[-\frac{1}{2}\big(\boldsymbol{w}^T\boldsymbol{x} - \theta\big)^2\right] - 1.$$

In order to do so, perform gradient descend over the loss function

$$E = \frac{1}{2} \sum_{i=1}^{n} \big(y(\boldsymbol{x}_i) - y_i\big)^2.$$

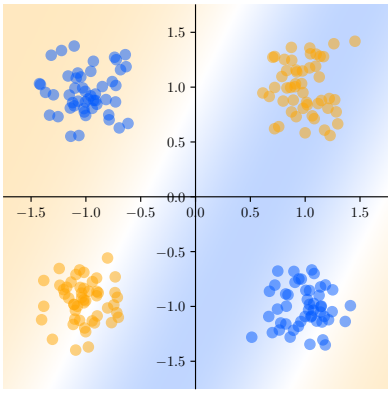
That is, randomly initialize $\boldsymbol{w}_0 \in \mathbb{R}^2$ and $\theta_0 \in \mathbb{R}$ and then iteratively compute the updates

$$\theta_{t+1} = \theta_t - \eta_\theta \frac{\partial E}{\partial \theta}$$

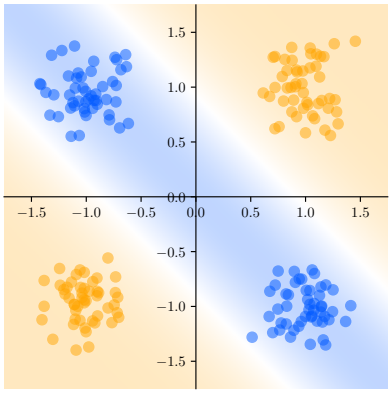
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_{\boldsymbol{w}} \frac{\partial E}{\partial \boldsymbol{w}}$$

Note: Good choices for the step sizes are $\eta_\theta = 0.001$ and $\eta_{\boldsymbol{w}} = 0.005$, but you are encourages to experiment with these parameters and see how they influence the behavior and outcome of the training procedure.

If all goes well, and say you also implement a function that can visualize a classifier, you should observe something like this



result obtained from $\boldsymbol{w}_0, \theta_0$



result obtained from $\boldsymbol{w}_{50}, \theta_{50}$

Voluntary sub-task: If you want to impress your professor, then also train a kernel SVM on the above data using a polynomial kernel

$$k\big(\boldsymbol{a}, \boldsymbol{b}\big) = \big(\boldsymbol{a}^T \boldsymbol{b} + c\big)^d.$$

task 3.5: exploring numerical instabilities: This task revisits task 2.1 and is not such much a task in itself but an eye opener! The goal is to raise awareness for the fact that doing math on digital computers may lead to unreliable results! **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 and untested implementations could have catastrophic results . . .

general hints and remarks

- see previous project sheets