Exercises for the course
Machine Learning 1
Winter semester 2020/21

Abteilung Maschinelles Lernen
Institut für Softwaretechnik und theoretische Informatik
Fakultät IV, Technische Universität Berlin
Prof. Dr. Klaus-Robert Müller
Email: klaus-robert.mueller@tu-berlin.de

Exercise Sheet 3

Exercise 1: Lagrange Multipliers (10 + 10 P)

Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^d$ be a dataset of N data points. We consider the objective function

$$J(\boldsymbol{\theta}) = \sum_{k=1}^{N} ||\boldsymbol{\theta} - \boldsymbol{x}_k||^2$$

to be minimized with respect to the parameter $\boldsymbol{\theta} \in \mathbb{R}^d$. In absence of constraints, the parameter $\boldsymbol{\theta}$ that minimizes this objective is given by the empirical mean $\boldsymbol{m} = \frac{1}{N}\sum_{k=1}^{N} \boldsymbol{x}_k$. However, this is generally not the case when the parameter $\boldsymbol{\theta}$ is constrained.

(a) Using the method of Lagrange multipliers, *find* the parameter $\boldsymbol{\theta}$ that minimizes $J(\boldsymbol{\theta})$ subject to the constraint $\boldsymbol{\theta}^\top \boldsymbol{b} = 0$, with $\boldsymbol{b}$ some unit vector in $\mathbb{R}^d$. Give a geometrical interpretation to your solution.

(b) Using the same method, *find* the parameter $\boldsymbol{\theta}$ that minimizes $J(\boldsymbol{\theta})$ subject to $||\boldsymbol{\theta} - \boldsymbol{c}||^2 = 1$, where $\boldsymbol{c}$ is a vector in $\mathbb{R}^d$ different from $\boldsymbol{m}$. Give a geometrical interpretation to your solution.

Exercise 2: Principal Component Analysis (10 + 10 P)

We consider a dataset $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^d$. Principal component analysis searches for a unit vector $\boldsymbol{u} \in \mathbb{R}^d$ such that projecting the data on that vector produces a distribution with maximum variance. Such vector can be found by solving the optimization problem:

$$\arg\max_{\boldsymbol{u}} \ \frac{1}{N}\sum_{k=1}^{N}\Big[\boldsymbol{u}^\top \boldsymbol{x}_k - \frac{1}{N}\Big(\sum_{l=1}^{N} \boldsymbol{u}^\top \boldsymbol{x}_l\Big)\Big]^2 \quad \text{with} \quad \|\boldsymbol{u}\|^2 = 1$$

(a) *Show* that the problem above can be rewritten as

$$\arg\max_{\boldsymbol{u}} \ \boldsymbol{u}^\top \boldsymbol{S}\boldsymbol{u} \quad \text{with} \quad \|\boldsymbol{u}\|^2 = 1$$

where $\boldsymbol{S} = \sum_{k=1}^{N}(\boldsymbol{x}_k - \boldsymbol{m})(\boldsymbol{x}_k - \boldsymbol{m})^\top$ is the scatter matrix, and $\boldsymbol{m} = \frac{1}{N}\sum_{k=1}^{N} \boldsymbol{x}_k$ is the empirical mean.

(b) *Show* using the method of Lagrange multipliers that the problem above can be reformulated as solving the eigenvalue problem

$$\boldsymbol{S}\boldsymbol{u} = \lambda \boldsymbol{u}$$

and retaining the eigenvector $\boldsymbol{u}$ associated to the highest eigenvalue λ.

Exercise 3: Bounds on Eigenvalues (5 + 5 + 5 + 5 P)

Let λ_1 denote the largest eigenvalue of the matrix $\boldsymbol{S}$. The eigenvalue λ_1 quantifies the variance of the data when projected on the first principal component. Because its computation can be expensive, we study how the latter can be bounded with the diagonal elements of the matrix $\boldsymbol{S}$.

(a) *Show* that $\sum_{i=1}^{d} \boldsymbol{S}_{ii}$ is an upper bound to the eigenvalue λ_1.

(b) *State* the conditions on the data for which the upper bound is tight.

(c) *Show* that $\max_{i=1}^{d} \boldsymbol{S}_{ii}$ is a lower bound to the eigenvalue λ_1.

(d) *State* the conditions on the data for which the lower bound is tight.

Exercise 4: Iterative PCA (10 P)

When performing principal component analysis, computing the full eigendecomposition of the scatter matrix $\boldsymbol{S}$ is typically slow, and we are often only interested in the first principal components. An efficient procedure to find the first principal component is *power iteration*. It starts with a random unit vector $\boldsymbol{w}^{(0)} \in \mathbb{R}^d$, and iteratively applies the parameter update

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{S}\boldsymbol{w}^{(t)} \big/ \|\boldsymbol{S}\boldsymbol{w}^{(t)}\|$$

until some convergence criterion is met. Here, we would like to show the exponential convergence of power iteration. For this, we look at the error terms

$$\mathcal{E}_k(\boldsymbol{w}) = \left| \frac{\boldsymbol{w}^\top \boldsymbol{u}_k}{\boldsymbol{w}^\top \boldsymbol{u}_1} \right| \qquad \text{with} \quad k = 2, \ldots, d,$$

and observe that they should all converge to zero as $\boldsymbol{w}$ approaches the eigenvector $\boldsymbol{u}_1$ and becomes orthogonal to other eigenvectors.

(a) Show that $\mathcal{E}_k(\boldsymbol{w}^{(T)}) = |\lambda_k/\lambda_1|^T \cdot \mathcal{E}_k(\boldsymbol{w}^{(0)})$, i.e. the convergence of the algorithm is exponential with the number of time steps T. *(Hint: to show this, it is useful to rewrite the scatter matrix in terms of eigenvalues and eigenvectors, i.e.* $\boldsymbol{S} = \sum_{i=1}^d \boldsymbol{u}_i \boldsymbol{u}_i^\top \lambda_i$*.)*

Exercise 5: Programming (30 P)

Download the programming files on ISIS and follow the instructions.

In [1]:

```
import numpy,sklearn,sklearn.datasets,utils
%matplotlib inline
```

Principal Component Analysis

In this exercise, we will experiment with two different techniques to compute the PCA components of a dataset:

- **Singular Value Decomposition (SVD)**
- **Power Iteration**: A technique that iteratively optimizes the PCA objective.

We consider a random subset of the Labeled Faces in the Wild (LFW) dataset, readily accessible from sklearn, and we apply some basic preprocessing to discount strong variations of luminosity and contrast.

In [2]:

```
D = sklearn.datasets.fetch_lfw_people(resize=0.5)['images']
D = D[numpy.random.mtrand.RandomState(1).permutation(len(D))[:2000]]*1.0
D = D - D.mean(axis=(1,2),keepdims=True)
D = D / D.std(axis=(1,2),keepdims=True)
print(D.shape)
```

```
(2000, 62, 47)
```

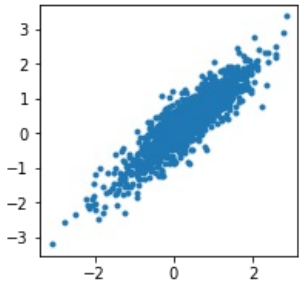
Two functions are provided for your convenience and are available in `utils.py` that is included in the zip archive. The functions are the following:

- **`utils.scatterplot`** produces a scatter plot from a two-dimensional data set.
- **`utils.render`** takes an array of data points or objects of similar shape, and renders them in the IPython notebook.

Some demo code that makes use of these functions is given below.

In [3]:

```
utils.scatterplot(D[:,32,20],D[:,32,21]) # Plot relation between adjacent pixels
utils.render(D[:30],15,2,vmax=5)         # Display first 10 examples in the data
```



PCA with Singular Value Decomposition (15 P)

Principal components can be found computing a singular value decomposition. Specifically, we assume a matrix $\bar{X}$ whose columns contain the data points represented as vectors, and where the data points have been centered (i.e. we have substracted to each of them the mean of the dataset). The matrix $\bar{X}$ is of size $d \times N$ where d is the number of input features and N is the number of data points. This matrix, more specifically, the rescaled matrix $Z = \frac{1}{\sqrt{N}}\bar{X}$ is then decomposed using singular value decomposition:

$$U\Lambda V = Z$$

The k principal components can then be found in the first k columns of the matrix U.

Tasks:

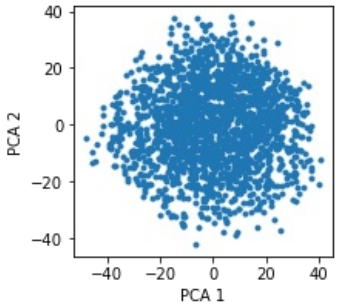
- **Compute the principal components of the data using the function `numpy.linalg.svd`.**
- **Measure the computational time required to find the principal components. Use the function `time.time()` for that purpose. Do *not* include in your estimate the computation overhead caused by loading the data, plotting and rendering.**
- **Plot the projection of the dataset on the first two principal components using the function `utils.scatterplot`.**
- **Visualize the 60 leading principal components using the function `utils.render`.**

Note that if the algorithm runs for more than 3 minutes, there may be some error in your implementation.

In [4]:

```
### REPLACE BY YOUR CODE
import solutions; solutions.pca_svd(D)
###
```

```
Time: 42.489 seconds
```



When looking at the scatter plot, we observe that much more variance is expressed in the first two principal components than in individual dimensions as it was plotted before. When looking at the principal components themselves which we render as images, we can see that the first principal components correspond to low-frequency filters that select for coarse features, and the following principal components capture progressively higher-frequency information and are also becoming more noisy.

PCA with Power Iteration (15 P)

The first PCA algorithm based on singular value decomposition is quite expensive to compute. Instead, the power iteration algorithm looks only for the first component and finds it using an iterative procedure. It starts with an initial weight vector $\boldsymbol{w} \in \mathrm{R}^d$, and repeatedly applies the update rule
$\boldsymbol{w} \leftarrow S\boldsymbol{w} \; / \; \|$

where S is the covariance matrix defined as S = \frac1N \bar{X}\bar{X}^\top. Like for standard PCA, the objective that iterative PCA optimizes is J(\boldsymbol{w}) = \boldsymbol{w}^\top S \boldsymbol{w} subject to the unit norm constraint for \boldsymbol{w}. We can therefore keep track of the progress of the algorithm after each iteration.

Tasks:

- **Implement the power iteration algorithm. Use as a stopping criterion the value of J(\boldsymbol{w}) between two iterations increasing by less than 0.01.**
- **Print the value of the objective function J(\boldsymbol{w}) at each iteration.**
- **Measure the time taken to find the principal component.**
- **Visualize the the eigenvector \boldsymbol{w} obtained after convergence using the function `utils.render`.**

Note that if the algorithm runs for more than 1 minute, there may be some error in your implementation.

In [5]:

```
### REPLACE BY YOUR CODE
import solutions; solutions.pca_powit(D)
###
```

```
iteration  0   J(w) =      0.656
iteration  1   J(w) =    126.401
iteration  2   J(w) =    207.474
iteration  3   J(w) =    221.764
iteration  4   J(w) =    231.388
iteration  5   J(w) =    241.061
iteration  6   J(w) =    250.240
iteration  7   J(w) =    257.989
iteration  8   J(w) =    263.840
iteration  9   J(w) =    267.884
iteration 10   J(w) =    270.508
iteration 11   J(w) =    272.142
iteration 12   J(w) =    273.133
iteration 13   J(w) =    273.725
iteration 14   J(w) =    274.075
iteration 15   J(w) =    274.281
iteration 16   J(w) =    274.402
iteration 17   J(w) =    274.473
iteration 18   J(w) =    274.514
iteration 19   J(w) =    274.538
iteration 20   J(w) =    274.552
iteration 21   J(w) =    274.561
iteration 22   J(w) =    274.565
iteration 23   J(w) =    274.568
iteration 24   J(w) =    274.570
Time: 5.962 seconds
```

We observe that the computation time has decreased significantly. The difference of performance becomes larger as the number of dimensions and data points increases. We can observe that the principal component is the same (sometimes up to a sign flip) as the one obtained by the SVD algorithm.

1 a) $f(\vec{\theta}) = \sum_{k=1}^{N} \|\vec{\theta} - \vec{x}_k\|^2$

Without constraint: $\operatorname{argmin}_{\theta} f(\vec{\theta}) \Rightarrow \vec{m} = \frac{1}{N} \sum_{k=1}^{N} \vec{x}_k = \vec{\theta}$

With constraint $\vec{\theta}^T \vec{b} = 0$: Application of Lagrange multiplier.

$$\frac{\partial \mathcal{L}(\vec{\theta}, \lambda)}{\partial \vec{\theta}} = \frac{\partial}{\partial \theta}\left(f(\vec{\theta}) + \lambda g(\vec{\theta})\right) = \frac{\partial}{\partial \theta}\left(f(\vec{\theta}) + \lambda \vec{\theta}^T \vec{b}\right) = \frac{\partial}{\partial \theta} \sum_{k=1}^{N} \|\vec{\theta} - \vec{x}_k\|^2 + \lambda \vec{\theta} \vec{b}$$

$$= \frac{\partial}{\partial \theta} \sum_{k=1}^{N} \|\vec{\theta}\|^2 - 2\vec{\theta}^T \vec{x}_k + \|\vec{x}_k\|^2 + \lambda \vec{\theta} \vec{b} = \frac{\partial}{\partial \theta} \|\vec{\theta}\|^2 - 2\vec{\theta}^T \vec{m} + \|\vec{m}\|^2 + \lambda \vec{\theta} \vec{b}$$

$$= \frac{\partial}{\partial \vec{\theta}} \|\theta - m\|^2 + \lambda \vec{\theta} \vec{b} = 2(\vec{\theta} - \vec{m}) + \lambda \vec{b} \stackrel{!}{=} 0$$

$$\Rightarrow \vec{\theta} = \frac{2\vec{m} - \lambda \vec{b}}{2}$$

$$2\vec{\theta} - 2\vec{m} + \lambda \vec{b} = 0 \Leftrightarrow \underbrace{2\vec{b}^T \vec{\theta}}_{0} - 2\vec{b}^T \vec{m} + \lambda \underbrace{\vec{b}^T \vec{b}}_{1} = \underbrace{\vec{b}^T \vec{0}}_{0}$$

$$\Leftrightarrow -2\vec{b}^T \vec{m} + \lambda = 0 \Leftrightarrow \lambda = 2\vec{b}^T \vec{m}$$

$$\Rightarrow \theta = \frac{2\vec{m} - \lambda \vec{b}}{2} = \frac{2\vec{m} - 2\vec{b}^T \vec{m} \vec{b}}{2} = \vec{m} - \vec{b} \vec{b}^T \vec{m}$$

The optimized parameter θ corresponds to the mean $\vec{m}$ with the subtracted projection of the mean on the vector $\vec{b}$.

b) $\mathcal{L}(\vec{\theta}, \lambda) = f(\vec{\theta}) + \lambda\, g(\vec{\theta}) = \|\vec{\theta} - \vec{m}\| + \lambda \|\vec{\theta} - \vec{c}\|^2 - \lambda$

$$\frac{\partial \mathcal{L}(\vec{\theta}, \lambda)}{\partial \vec{\theta}} = \frac{\partial}{\partial \vec{\theta}} \left(\|\vec{\theta} - \vec{m}\|^2 + \lambda \left(\|\vec{\theta} - \vec{c}\|^2 - 1 \right) \right)$$

$$= \chi(\vec{\theta} - \vec{m}) + \lambda (\vec{\theta} - \vec{c}) \overset{!}{=} 0$$

$$\Leftrightarrow (1 + \lambda)(\vec{\theta} - \vec{c}) - (\vec{m} - \vec{c}) = 0$$

1) $$\Leftrightarrow (1 + \lambda)(\vec{\theta} - \vec{c}) = \vec{m} - \vec{c}$$

$$\Leftrightarrow (1 + \lambda)^2 \underbrace{\|\vec{\theta} - \vec{c}\|^2}_{=1} = \|\vec{m} - \vec{c}\|^2$$

2) $$\Leftrightarrow |1 + \lambda| = \pm \|\vec{m} - \vec{c}\|$$

2) in 1): $\pm \|\vec{m} - \vec{c}\| (\vec{\theta} - \vec{c}) = \vec{m} - \vec{c}$

$$\Leftrightarrow \vec{\theta} = \vec{c} \pm \frac{\vec{m} - \vec{c}}{\|\vec{m} - \vec{c}\|}$$

Only one of these two solutions minimizes the objective $f(\vec{\theta})$. The constraint $g(\vec{\theta}) = \|\vec{\theta} - \vec{c}\|^2 - 1 = 0$ corresponds to a circle with center $\vec{c}$. One solution maximizes the distance to the mean $\vec{m}$, the other one minimizes it. But both solution lie on the same circle.

2

a) $$\underset{u}{\operatorname{argmax}} \frac{1}{N} \sum_{k=1}^{N} \left[\vec{u}^T \vec{x}_k - \frac{1}{N} \left(\sum_{l=1}^{N} \vec{u}^T \vec{x}_l \right) \right]^2$$

1) $$\frac{1}{N} \sum_{l=1}^{N} \vec{x}_l = \vec{m}$$

$$\overset{1)}{=} \underset{u}{\operatorname{argmax}} \frac{1}{N} \sum_{k=1}^{N} \left[\vec{u}^T \vec{x}_k - \vec{u}^T \cdot \vec{m} \right]^2$$

$$= \underset{u}{\operatorname{argmax}} \frac{1}{N} \sum_{k=1}^{N} \left(\vec{u}^T \vec{x}_k - \vec{u}^T \vec{m} \right) \left(\vec{u}^T \vec{x}_k - \vec{u}^T \vec{m} \right)^T$$

$$= \underset{u}{\operatorname{argmax}} \frac{1}{N} \sum_{k=1}^{N} \vec{u}^T (\vec{x}_k - \vec{m}) (\vec{x}_k - \vec{m})^T \vec{u}$$

$$= \underset{u}{\operatorname{argmax}} \frac{1}{N} \vec{u}^T \vec{S} \vec{u}$$

$$= \underset{u}{\operatorname{argmax}} \vec{u}^T \vec{S} \vec{u} \quad \square$$

b) $$\mathcal{L}(\vec{u}, \vec{S}) = \vec{u}^T \vec{S} \vec{u} + \lambda (\|\vec{u}\|^2 - 1)$$

$$\frac{\partial \mathcal{L}(\vec{u}, \vec{S})}{\partial \vec{u}} = \vec{S} \vec{u} + \vec{S}^T \vec{u} + 2 \lambda \vec{u}$$

$$\overset{\vec{S} = \vec{S}^T}{=} 2 \cdot \vec{S} \vec{u} + 2 \lambda \vec{u} \overset{!}{=} 0$$

$$1) \iff \vec{S} \vec{u} = + \lambda \vec{u}$$

In a) we have $\underset{u}{\operatorname{argmax}} \vec{u}^T \vec{S} \vec{u}$ which becomes

$\underset{u}{\operatorname{argmax}} \vec{u}^T \vec{S} \vec{u} = \underset{u}{\operatorname{argmax}} \vec{u}^T \lambda \vec{u}$ using 1)

$= \underset{u}{\operatorname{argmax}} \underbrace{\vec{u}^T \vec{u}}_{=1} \lambda = \underset{u}{\operatorname{argmax}} \lambda = \max \lambda_i$

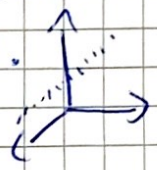
which corresponds to eigenvector $\vec{u}$ with maximum eigenvalue λ_i

3

a) $\sum_{i=1}^{d} S_{ii} = tr(\vec{S}) \geq \max_i \lambda_i$ since the trace of a matrix corresponds to the sum of its eigenvalues:

$$tr(\vec{A}) = \prod_{i=1}^{N} \lambda_i$$

b) The condition is tight if $\prod_{i=1}^{N} \lambda_i = \max_i \lambda_i$ the sum of all eigenvalues corresponds to the maximum eigenvalue. In that case all other eigenvalues must be 0. This is the case if the data is one-dimensional.



c) According to the previous exercise

$\lambda_1 = \vec{u}_1^T \vec{S} \vec{u}_1$ which corresponds to $\arg\max_u \vec{u}^T \vec{S} \vec{u}$

This maximized $\vec{u}$ vector can always be represented as a unit vector: $\vec{e}_i$

Therefore, $\arg\max \vec{u}^T \vec{S} u \geq \arg\max_i \vec{e}_i^T \vec{S} \vec{e}_i = \arg\max_i S_{ii}$

since $\vec{e}_i^T$ and $\vec{e}_i$ select the rows and columns of $\vec{S}$.

d) The vector $\vec{u}$ can always be represented as a linear combination of unit vectors: $\alpha_1 \vec{e}_1 + \alpha_2 \vec{e}_2 + \cdots + \alpha_d \vec{e}_d$

If $\vec{u}$ corresponds to a unit vector, meaning $\forall \alpha_i = 0$ except for one α_j, then the bound is tight.

4 a)

$$\Sigma_k(w^{(t+1)}) = \left| \frac{w^{(t+1)} \mu_k}{w^{(t+1)} \mu_1} \right|$$

$$= \left| \frac{\left(S w^{(t)} / \| S w^{(t)} \|\right)^T \cdot \mu_k}{\left(S w^{(t)} / \| S w^{(t)} \|\right)^T \cdot \mu_1} \right|$$

$$= \left| \frac{w^{(t)T} \cdot S \cdot \mu_k}{w^{(t)T} S \cdot \mu_1} \right| = \left| \frac{w^{(t)T} \left(\sum_{i=1}^{d} \mu_i \underbrace{\mu_i^T \mu_k}_{\delta_{ik}} \lambda_i \right)}{w^{(t)T} \left(\sum_{i=1}^{d} \mu_i \underbrace{\mu_i^T \mu_1}_{\delta_{i1}} \lambda_i \right)} \right|$$

$$= \left| \frac{w^{(t)T} \mu_k \lambda_k}{w^{(t)T} \mu_1 \lambda_1} \right| = \underbrace{\left| \frac{w^{(t)T} \mu_k}{w^{(t)T} \mu_1} \right|}_{\varepsilon_k} \left| \frac{\lambda_k}{\lambda_1} \right| = \left| \frac{\lambda_k}{\lambda_1} \right|^T \cdot \varepsilon_k(w^{(t)}$$

□