10907 Pattern Recognition

Lecturers
Prof. Dr. Thomas Vetter ⟨thomas.vetter@unibas.ch⟩

Assistants
Dr. Adam Kortylewski ⟨adam.kortylewski@unibas.ch⟩
Dennis Madsen ⟨dennis.madsen@unibas.ch⟩
Dana Rahbani ⟨dana.rahbani@unibas.ch⟩

Exercise 5 — PCA

Introduction	19.11
Deadline	**27.11** Upload code to Courses.
	26.11+27.11 Group presentations, U1.001, see schedule online

In this series you will implement Principal Component Analysis using the SVD. In the first exercise, you will implement the basic algorithm and try it out on a toy dataset. In the second exercise, you will implement a simple face recognition system, where PCA is used for feature extraction. In the third exercise, you learn a 3D face model from 11 surface scans of faces.

You can download the data needed for this exercise and a code skeleton from the following repository: `https://bitbucket.org/gravis-unibas/pattern-recognition-2018`.
A number of helper functions are provided - your task is to implement the *TODO* parts in the python (.py) files.

Remember to upload your code to courses in a ZIP file. Do NOT include the data folder. Only the python files you edited should be included + a file containing the group members names. Only 1 person from the team should upload!

Data:

For the first exercise, a 2-dimensional dataset consisting of 100 samples is provided in `toy_data.m`.

For the face recognition part, the provided data are 192×128 grayscale images along with a person identifier tag. For the recognition task, two sets of 144 faces are provided:

- `gallery.mat` : to be used to learn the principal components
- `novel.mat` : test set to be used to evaluate the recognition performance

For the 3D face model part, the provided data are 11 `.ply` files contained in the folder `data/face-data`. The function to load the data is also provided for the exercise. Every `.ply` file is a 3D mesh of a face.

PCA and its properties

Todo 1 (Implement PCA)

Implement the Principal Component Analysis (PCA) class using the SVD. In particular, implement the following functions:

train A function that takes a $n \times p$ data matrix, and stores 3 values in the object: (μ, U, S) where $\mu \in \mathbb{R}^n$ is the mean of the data matrix, $U \in \mathbb{R}^{n \times m}$ is an (orthonormal) matrix containing the m principal components and $S \in \mathbb{R}^m$ is a vector, where the i-th entry contains the i-th variance value λ corresponding to the i-th principal component. Remember to limit the number of principal components to the value given by `self._maxComponents` (all principal components should be used if it is set to -1). *Hint: avoid computing the full matrices by setting* `full_matrices=False`

to_pca A function that takes a vector $x \in \mathbb{R}^n$ as an input and returns the coordinates $\alpha \in \mathbb{R}^m$ in the space spanned by the principal components. Do this by using the stored values (μ, U, S).

from_pca A function that takes as input a vector of PCA coordinates $\alpha \in R^m$. It returns a vector $x \in \mathbb{R}^n$, of the data in the original coordinate system. Do this by using the stored values (μ, U, S)

project A function that takes as an argument a vector $x \in \mathbb{R}^n$ and the dimensionality of the subspace k onto which it should be projected. It should return the projection $\hat{x} = P_k[x] \in \mathbb{R}^n$, where $P_k[x]$ denotes the projection of the k dimensional subspace spanned by the first k principal components. *Hint: use the functions you implemented before.*

1 Toy dataset

Todo 2 (Test PCA on Toy dataset)

Load the data given in `toy_data.m`. Perform the following experiments.

1. Plot the data and the principal axes (use e.g. the provided function `plot_pca`)
2. Compute the variance of the toy dataset and compare it to the vector S. What do you observe? Explain.
3. Project the data onto the first principal component. Compute the variance in the data (the same way as in the first step) and compare it to the vector S. What do you observe now?

Hint: Make use of the function `plot_pca` *to visually debug your code.*

2 Face recognition with PCA

Todo 3 (Compute Eigenfaces)

Use the functions implemented above to perform PCA on the set of face images provided in the data `gallery.mat`. Similarly to the previous case, your method should compute the mean face and the set of the 25 most important eigenvectors (eigenfaces). Limit it to 25 components by setting the `maxComponents` parameter in the `pca` class.

Hint: In order to do this, you must assemble a data matrix by stacking each image $m \times n$ into a a column vector $mn \times 1$ and concatenate all column vectors horizontally.

Plot the variance of the 25 principal components (x-axis is principal component index and y-axis is variance).

Todo 4 (Use eigenfaces for face-recognition)

Load the file `novel.mat`. This dataset contains face images of the same persons as in the gallery, taken under different conditions (pose, lighting, etc.)

For each new face in the `novel.mat` dataset perform the following:

1. Project the new face into the PCA basis to obtain its feature coordinates (α vector)
2. Attribute the new face to the face with the closest feature coordinates in the `gallery.mat` dataset.

Using the provided labels in the data, evaluate your recognition error rate (number of faces falsely attributed). Do this with 2 different distance measures:

1. Using the Euclidean distance to compare feature vectors
2. Using Mahalanobis distance to compare feature vectors

Performing the face recognition on feature vectors of a halfened dimensionality.

- What is the effect of using the Mahalanobis distance vs. the Euclidean ?
- What is the effect of reducing the dimensionality on the recognition results ?

Visualize some of the correctly and wrongly classified images together with the projected novel image. Example of the visualization is shown in figure 1

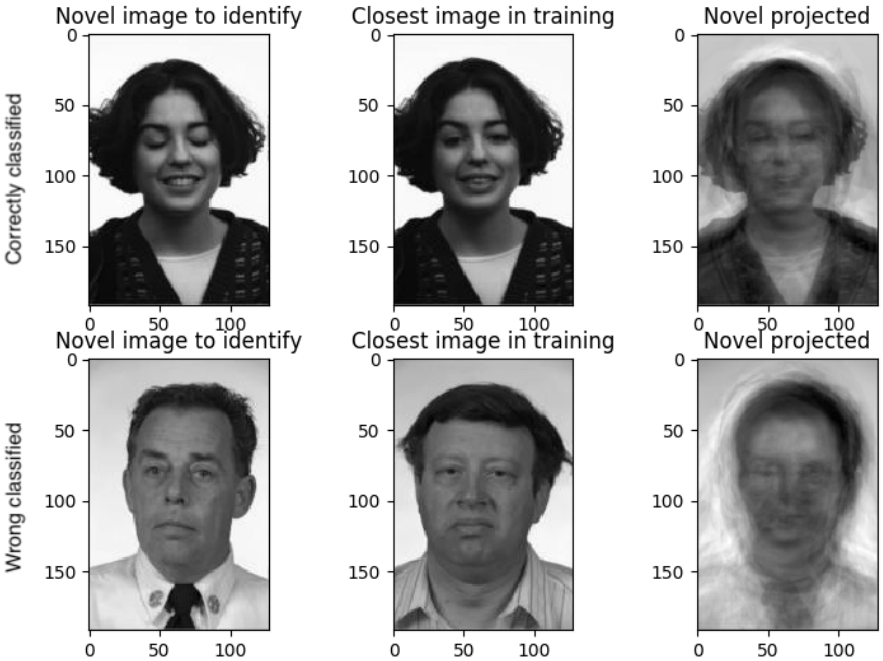

Figure 1: Image recognition visualization example.

3 3D Face Models

Todo 5 (Create and use a 3D face model)

In this exercise you will use the implemented PCA functions to work with 3D-meshes of faces. You are provided with 11 samples of faces and the helper functions to render the faces. Use a random vector as the input for your previously created function `from_pca` in order to create a random face.

1. **Initialization** Code to load the meshes and visualize a face from the dataset is already given in `ex5-PCA_3_3DFaces.py`. The mesh visualizer function `renderFace` makes use of the `plotly` library which can be installed via `pip`. The `renderFace` function creates a html file with the name given to the function. This file is then automatically opened by your default browser.

2. **PCA & Random Faces** Use your previously implemented `pca` class to compute the PCA of the `faces` matrix. In the function `renderRandomFace` we want to draw random samples from the face model. For this, create a vector `alpha` with random numbers in the range from -1.0 to 1.0 and use this as the input for your `from_pca` function.

3. **Low-rank Reconstruction** Take one sample of the face and create a low-rank reconstruction. This is done in the `lowRankApproximation` function. Use the previously created `project` function to project the face into a k maximum rank. This limits the number of eigenvectors used to construct the face. Explore different ranks from $k = 1$ to $k = 11$ and discuss what you see.