3. Assignment

Discriminant functions, PCA and Perceptron

Deadline: 25.12.2017

Description of assignment

This assignment covers the following topics:

- Discriminant functions (task 1)
- Principal component analysis (task 2)
- Perceptron (task 3)

It is based on the lectures 03 to 05.

Guidelines

It is of utmost importance that your **code is executable**. Use relative paths. Make sure that all results are printed and plotted by executing `main.m` and no errors occur.

Create a **new window for each result** (hint: command `figure`). Do not overwrite results!

In the **second line of every file** fill in the following information as a comment: first name, last name and student id.

All provided script and function files have to be used and completed if necessary. If you create additional files make sure that the code is still executable by calling `main.m`.

All files have to be collected in a **zip-archive** and be uploaded in TUWEL (until the deadline).

Fisher's Iris data set

This data set contains 50 samples of three lilies. Four features have been collected for each sample:

(1) length of sepalum
(2) width of sepalum

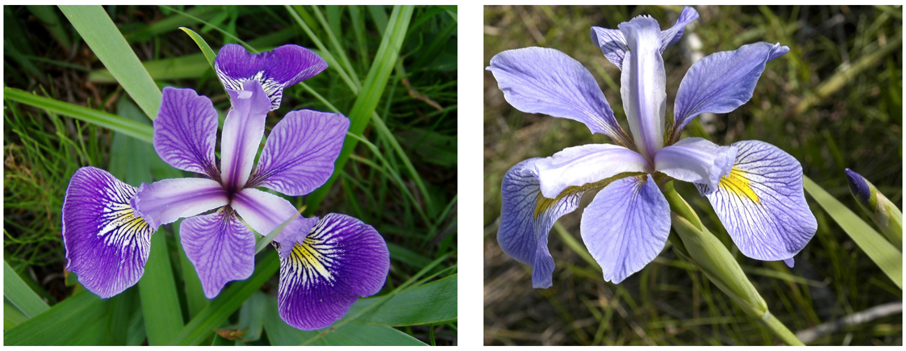
FIGURE 1. Left: Iris virginica (class 1); Right: Iris versicolor (class 2).

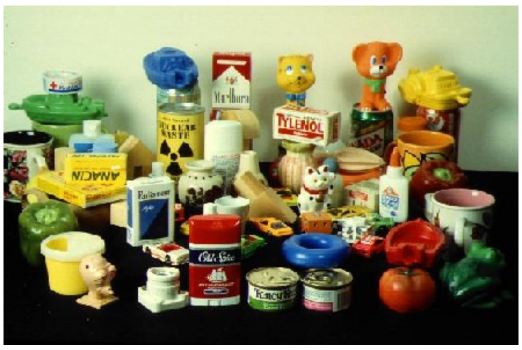
FIGURE 2. COIL-100 data set.

(3) length of petalum
(4) width of petalum

For the toy example in task 3, this data set is reduced to a *two-class-problem*. In Figure 1 you can see the two lilies used for this assignment. The lily data set has already been imported for you in `percep.m`.

COIL-100 data set

The second data set of this assignment is called COIL-100. It was produced by the university of Columbia[1]. The data set consists of 100 objects (classes). For each of these objects there are 72 color images showing the object from different viewing angles (0 - 355 degree). Figure 2 gives an overview of the objects covered in the data set.

[1]http://www.cs.columbia.edu/CAVE/software/softlib/coil-100.php

Task 1: Discriminant functions

Total points: 15

Find the discriminant functions for class A and B, and the decision border based on this training data:

$$A = \left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \end{pmatrix} \right) B = \left(\begin{pmatrix} 4 \\ 1 \end{pmatrix}, \begin{pmatrix} 4 \\ 2 \end{pmatrix}, \begin{pmatrix} 5 \\ 1 \end{pmatrix}, \begin{pmatrix} 5 \\ 2 \end{pmatrix} \right) \quad (1)$$

You can assume that both priors are 0,5.

Do this for the following cases:

(1) General case, each class has its own covariance matrix
(2) Special case 1 (see lecture 03) with $\sigma^2 = 1$.

Task 1 is to be solved "**on paper**" (without MATLAB). The solution should show all necessary steps of calculation and intermediate results:

- mean vectors [0,5 point]
- covariance matrices (for general case) [1 points]
- determinants (for general case) [0,5 points]
- inverses (for general case) [1 point]
- discriminant functions (for both cases) [6 points]
- decision border (for both cases) [6 points]

It is up to you if you do a scan (or good quality photograph) of your handwritten solution or if you prefer to use some kind of formula editor.

Important remark: Make sure that your hand-in is easily readable. Hand-in the solution of this task as "**discriminant.pdf**".

Task 2: Principal component analysis

Total points: 8
Files: `pca.m`, `loadData.m`, `determineMeanVector.m`, `determineMeanSubtracted.m`, `eigsort.m`, `determineBasis.m`, `reconstructEval.m`

The aim of this task is to determine the Eigenvectors of a basis from a training set and to reconstruct training and test samples with the help of this basis and the corresponding coefficients.

You will experiment with different numbers of Eigenvectors in the basis to show the difference between reconstruction based on the whole basis and a reduced basis.

Task 2.1: Read-in data set [1 Point]. Download the COIL-100 data set and implement the function `loadData(...)` (see `Task 2.1` in `pca.m`). This function should generate a training and a test set from the COIL-100 data set.

The COIL-100 data set consists of 7200 images (72 for each class). Choose five classes and select for each class eight views (images of the class) to build your training set. This results in a training set of 40 images. For the test set choose 20 images of the five classes, which are **NOT** in the training set.

> **Remark**: Do not hand-in the files of the data set. Hence, you have to read-in the files as they are. From the path set in `loadData.m`.

Each selected image needs to be processed to be used in the data sets:

- convert the images from color to gray-scale (look for a suitable MATLAB function)
- transform the resulting 2D gray-scale images into 1D feature vectors

In the end `loadData(...)` should output two matrices: `test` and `training`.

Task 2.2: Determine mean vector [0,5 Points]. Implement the function `[meanVector] = determineMeanVector(training)` (see `Task 2.2` in `pca.m`). The `meanVector` is like the "mean object" of the chosen training samples.

Task 2.3: Determine difference vector [0,5 Points]. Implement the function `determineMeanSubtracted(...)` (see `Task 2.3` in `pca.m`). The output of this function is a matrix A consisting of the difference vectors of the training set.

Task 2.4: Find Eigenvalues and Eigenvectors [1 Point]. In the lecture you learned that you should find the Eigenvalues and -vectors based on a covariance matrix $\sum = AA^T$ calculated from your training data. Unfortunately, this matrix would be very big for this example. Hence, you should use the so-called "**transpose trick**" in this assignment.

In a nutshell, you can determine the Eigenvalues and -vectors based on the matrix $A^T A$ instead of AA^T (see `Task 2.4` in `pca.m`). The trick works as follows:

If $\mathbf{v}_i$ is an Eigenvector of A^TA and λ_i its Eigenvalue, then

$$\begin{aligned}(A^TA)\mathbf{v}_i &= \lambda_i\mathbf{v}_i\\ A(A^TA)\mathbf{v}_i &= A(\lambda_i\mathbf{v}_i)\\ AA^T(A\mathbf{v}_i) &= \lambda_i(A\mathbf{v}_i)\end{aligned}$$

Hence, if $\mathbf{v}_i$ is an Eigenvector of A^TA, then $A\mathbf{v}_i$ is an Eigenvector of AA^T with the same Eigenvalue. $A\mathbf{v}_1, A\mathbf{v}_2, ..., A\mathbf{v}_n$ are the Eigenvectors of Σ, where n is the number of images in the training set. The Eigenobjects (vectors) are $\mathbf{u}_1 = A\mathbf{v}_1$, $\mathbf{u}_2 = A\mathbf{v}_2$, . . . , $\mathbf{u}_n = A\mathbf{v}_n$.

Hint: You are allowed to use MATLAB functions to find the Eigenvectors and -values (look for `eig`). It is possible to solve this task in one line of code.

Task 2.5: Sort Eigenvalues and -vectors [0,5 Points]. Implement the function in the file `eigsort.m` (see `Task 2.5` in `pca.m`). The task of this function is to sort the found Eigenvectors based on their corresponding Eigenvalues in a descending fashion (i.e. Eigenvector of biggest Eigenvalue comes first).

Task 2.6: Determine basis [0,5 Points]. Now it's time to create the basis U from the sorted Eigenvectors. Hint: As you used the transpose trick, it is necessary to do a final operation (see Task 2.4).

Furthermore, you should normalize the Eigenvectors with the help of the MATLAB function `normc`.

Do the implementation of this task in `determineBasis.m` (see `Task 2.6` in `pca.m`).

Task 2.7: Reconstruction and evaluation [4 Points]. Implement the function `reconstructEval(...)` (see `Task 2.7` in `pca.m`). The task of this function is to calculate the reconstruction error for a given data set (input parameter) for all possible bases.

The different bases result from the reduction of basis U. For example for a training set of 40 samples you can determine a basis with 40 Eigenvectors. By reduction of the basis 40 different bases result (basis with 40 Eigenvectors is reduced by one vector at a time).

For the reconstruction of each sample (image) in a data set, it is necessary to determine its coefficients of reconstruction. Then this coefficients can be applied to reconstruct the sample from a certain basis as described in lecture 04.

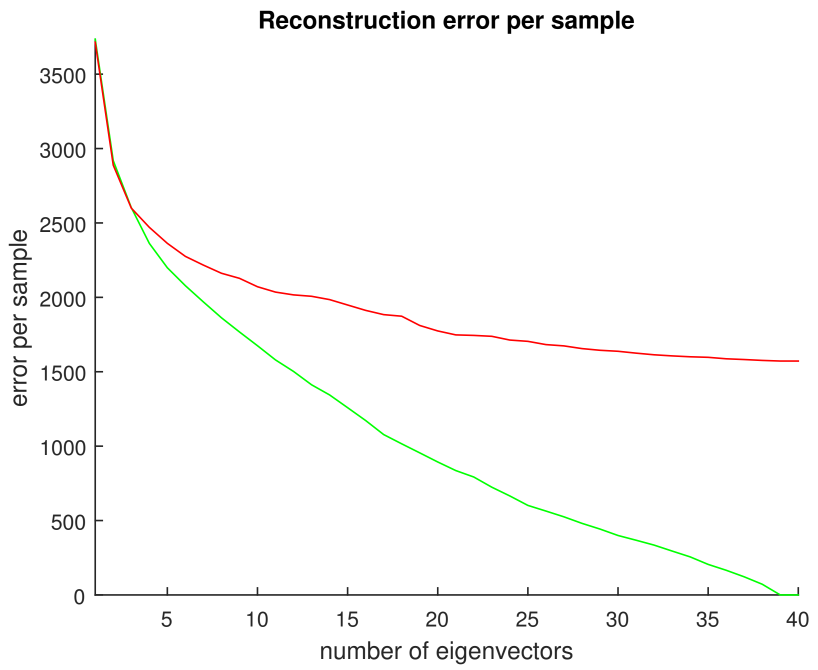

Figure 3. Mean reconstruction error per sample (y-axis) for basis with different size (x-axis). Green: training data set. Red: test data set.

The function `reconstructEval(...)` should calculate the mean reconstruction error per sample for each basis and store the results in the output vector `error`. The error is measured by the Euclidean distance between the between original and reconstruction.

If you apply this function on the test and the training set, curves like in Figure 3 will result from drawing the error against the size of the basis.

Important remark: If you are using the MATLAB function `eig` to calculate the Eigenvectors and -values, you need to use the difference vectors (difference between sample and mean vector) for the calculation of the coefficients!

Task 3: Perceptron

Total points: 7
Files: `percep.m`, `percepAlgo.m`

The aim of this task is to understand and implement the perceptron algorithm for data in $\mathbb{R}^d$.

The maximum number of epochs is set to 100 (and should not be changed for the hand-in).

Task 3.1: Your own perceptron algorithm [5 Points]. Implement your own preceptron algorithm in the script file `percepAlgo.m` with the predefined input and output parameters (see `Task 3.1` in `percepAlgo.m`). It is not allowed to use the existing perceptron function of MATLAB.

Your algorithm has to terminate when the maximum number of epochs is reached or when it found a linear decision boundary, which classifies the samples in the training set error-free.

Besides the output parameter w (weight vector), your function should also provide a text output on the console, which shows the necessary number of epochs (Hint: Use `disp`).

Task 3.2: Application on Iris data set [2 Points]. Use the Iris data set to test your perceptron algorithm. The Iris data set has already been divided into a test and training set (see source code in `percep.m`).

First try to find a linear decision boundary with your perceptron algorithm (maximum epochs is still 100). Then, classify the training and test set based on the found decision boundary and evaluate how many lilies are classified wrongly (see `Task 3.2` in `percep.m`).

For the evaluation you can use the function `evaluate(...)` from the previous assignment. Print the percentage of wrongly classified lilies (for training and test set) onto the console.

Find the best combination of features based on the classification result on the test data set. (In this case, the test data set acts like a validation data set and helps us to decide for the best feature configuration. To keep it simple you only need to work with the pre-defined split into training and test set. In pratice one would for example use random sub-sampling and calculate a mean error.)

Remark: The best combination should be kept in the source code (in `percep.m` line 16).