FGV EMAp
ESCOLA DE MATEMÁTICA APLICADA

Fundamentos de Data Science
Assessed Coursework 2

This assignment must be submitted by August 26th, 2021 at 8:00 am.
Late submissions will NOT be accepted.

This coursework is assessed and mandatory
and is worth 30% of your total final grade for this course.

Learning outcomes assessed

This coursework will test your understanding of some important machine learning algorithms through the writing of Matlab code that implements them.

Instructions

Identifier
Please choose a random number of 6 digits. Make sure that you keep a copy of that number as it will be used to provide the feedback (please avoid trivial numbers, such as 000000 or 123456. Also please avoid numbers starting with zero).

Submission
Compress your files into a unique zip file and rename it with your random digit number (so the zip file name becomes something like 723923.zip). Then submit it through ECLASS. The zip file you submit cannot be overwritten by anyone else, and it cannot be read by any other student. You can, however, overwrite your submission as often as you like, by resubmitting, though only the last version submitted will be kept. Submission after the deadline will NOT be accepted.

If you have issues, at the very last minute, email your zip file as an attachment at alberto.paccanaro@fgv.br with the subject "URGENT – COURSEWORK 2 SUBMISSION". In the body of the message, explain the reason for not submitting through ECLASS.

For this assignment you will have to submit the following files:

- A file containing function ***LinearRegressionWD*** required for exercise 1
- Two files containing function ***MultiLayerPerceptron*** and ***ROCcurve*** required for exercise 2
- Two files containing the scripts ***myPCAtoy*** and ***myPCAfaces*** required for exercise 3
- A file containing function ***mykmeans*** required for exercise 4

Please use the function name described above (and in the exercises) – do not use different names.

IMPORTANT: If you will write any other extra function or script as part of your solution for these exercises, you must also submit these. For these extra functions or scripts, you can choose the file name.

On Eclass, you will also find the function `DisplayData` that you will need for exercise 3.

All the work you submit should be solely your own work.
Coursework submissions will be checked for this.

DATASET DESCRIPTION

You will be using 5 different datasets, that you will find on the Eclass page for this coursework.

1) `Housing.txt`: this dataset is constituted by *506* points in 14 dimensions. Each point represents a house in the Boston area, and the 14 attributes that you find orderly in each column are the following:

   CRIM - per capita crime rate by town
   ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
   INDUS - proportion of non-retail business acres per town.
   CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
   NOX - nitric oxides concentration (parts per 10 million)
   RM - average number of rooms per dwelling
   AGE - proportion of owner-occupied units built prior to 1940
   DIS - weighted distances to five Boston employment centres
   RAD - index of accessibility to radial highways
   TAX - full-value property-tax rate per $10,000
   PTRATIO - pupil-teacher ratio by town
   B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
   LSTAT - % lower status of the population
   MEDV - Median value of owner-occupied homes in $1000's

2) *Occupacy* dataset: points are in 5 dimensions. Each point refers to a room, which is described by 5 attributes (first 5 columns). Each room belongs to one of possible 2 classes (last column). I provide one training dataset (`occupancy_training_set.csv`) and 2 testing datasets (`occupancy_test_set_1.csv` and `occupancy_test_set_2.csv`).
   The description of the attributes is as follows:
   Temperature, in Celsius
   Relative Humidity, in percentage
   Light, in Lux
   CO2, in ppm
   Humidity Ratio, in kgwater-vapor/kg-air

3) `pcadata.mat`: a small dataset of 50 random points, Gaussian distributed, in 2D

4) `pcafaces.mat`: A dataset of 5000 images of faces of famous people, taken from a public repository. Each image is a 32x32 matrix of pixel which has been linearized into a vector of size 1024.

5) `DataForKmeans.mat`: a dataset of points in 2D

EXERCISE 1 (value: 5 %)

Write a Matlab function "***LinearRegressionWD***" that implements Linear Regression with weight decay regularization using stochastic gradient descent (note: not the normal equations with weight decay).

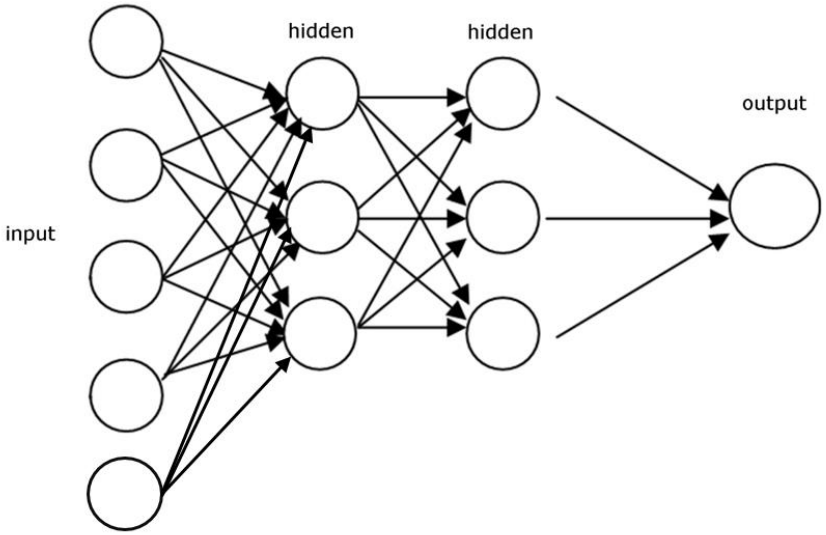
Your implementation will predict the MEDV value (Median value of owner-occupied homes in $1000's) of the `Housing.txt` dataset, given the other 13 attributes.

The implementation will plot the training error as a function of the iteration. The algorithm will stop when the relative improvement in the error function between two successive iterations will be smaller than the value of a user-defined threshold τ.

Your implementation will also report the test error obtained on a 10% held out dataset.

EXERCISE 2 (value: 9 %)

Write a Matlab function *"**MultiLayerPerceptron**"* that implements the neural network specified in the following diagram for predicting room occupancy for the *Occupacy* data.



The implementation will plot the training error as a function of the iteration. The algorithm will stop when the relative improvement in the error function between two successive iterations will be smaller than the value of a user-defined threshold τ.

After training your model (on the training dataset), assess the performance of your model on the two testing datasets by creating a figure displaying the Receiver operating characteristic (ROC) curve.

(note: in this exercise you are also required to implement a function called *"**ROCcurve**"* that will calculate and plot the ROC curve).

EXERCISE 3 (value: 9 %)

In this exercise you will explore Principal Components Analysis (PCA). The exercise is divided in 2 parts. In the first part, you will work with a small toy dataset of artificial data, developing functions needed for carrying out PCA. In the second part, you will work with a larger dataset of real world data (images of faces of famous people), and you will be able to re-use the functions you wrote for the small toy dataset on a larger scale.

Note that this is a good approach to use when implementing algorithms: make sure that they work well on small datasets and then apply them to your real problem. *Remember that the implementation for the small dataset needs to be efficient, otherwise its execution time will be very large when applied to the larger dataset*.

For both datasets, you will:

1. Calculate the principal components of your data
2. Project your high dimensional data onto (a smaller space defined by) a few principal components
3. Recover your original high dimensional data by re-projecting back the projected data onto the original space.

Note also that this exercise is about implementing PCA yourself, so you cannot use any of the different functions available in Matlab which implement it (e.g. `pca`). You will have to calculate the principal components through the eigendecomposition of the covariance matrix of the data (you will need to use the Matlab function `eig` for the eigendecomposition. Note that there is a similar function in Matlab called `eigs`, but it returns only the first 6 eigenvalues, so you should not use it).

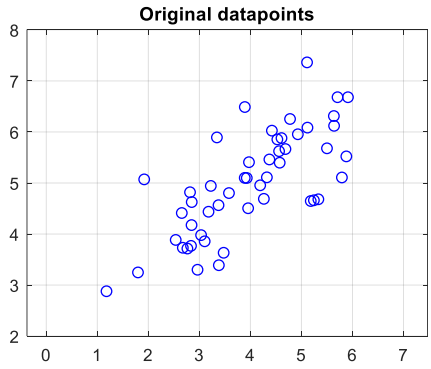
Part 1

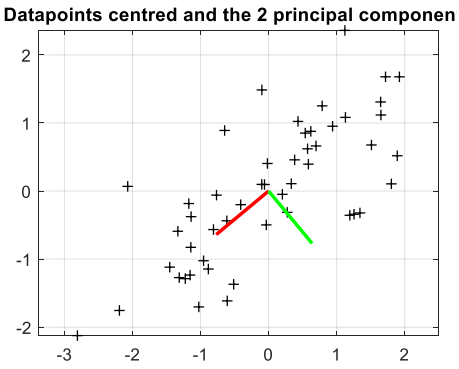
Write a script called "***myPCAtoy***" that will:

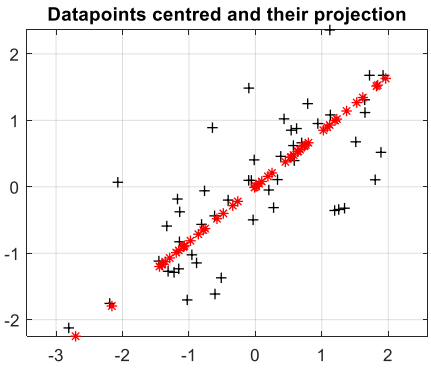
a) load the 2D datapoints in `pcadata.mat` and plot them;
b) centre the data on the origin, calculate the 2 principal components and plot them together with the centred points;
c) project these points onto the first principal components and display the centred points together with the projected points;
d) plot the original datapoints and their reconstruction in the original space.

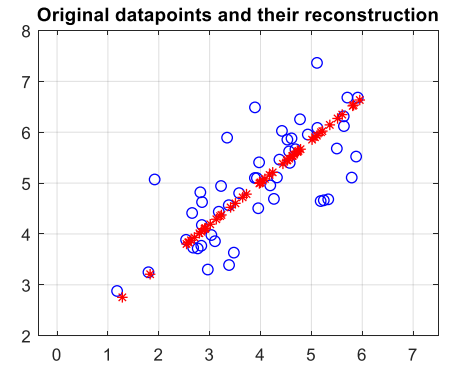
All the plots will be produced as subplots of the same figure. At the end, your figure should look like the one shown below.

(note: you will need to organize your code by creating different functions that are called from your script. In this way you will be able to use the same functions on part 2 – and you will already know that they work correctly, because you have seen them working on part1 😊. For these functions, you can choose the function name).









Part 2

In this part of the exercise you will repeat on a larger real world dataset exactly the same analysis that you have already performed on the small toy dataset in part 1.

Your script "***myPCAfaces***" will use the functions you have written for the small toy dataset and use them on the dataset of 5000 images of faces contained in `pcafaces.mat`.

Note that the function "`displayData`" is also provided on Eclass to display images. To use it: assuming that the images are contained in a matrix called X (of size 5000x1024), to display the first *m* of them you will write: `displayData(X(1:m, :))`

Your script will project the faces onto the first *k* principal components, where the value of *k* is set by the user. It will then create a figure like the one given below. This figure will contain 2 subplots. The first subplot will display the first 100 images of the original data. The second subplot will display the first 100 images of the reconstructed data – in this way you will be able to compare how good your reconstruction is!

For fun, you can experiment and check the quality of the reconstructed faces for different number of principal components… 😊

Original faces

Recovered faces with 200 PCs

EXERCISE 4 (value: 7 %)

In this exercise, you will write the function ***"mykmeans"*** implementing a vanilla version of the k-means clustering algorithm for data points in 2 dimensions.

"mykmeans" will receive two inputs:

1) a dataset of *n* points in 2D
2) the number of clusters *k*.

The algorithm will stop when the relative change in the position of the centroids between two successive iterations will be smaller than the value of a user-defined threshold τ.

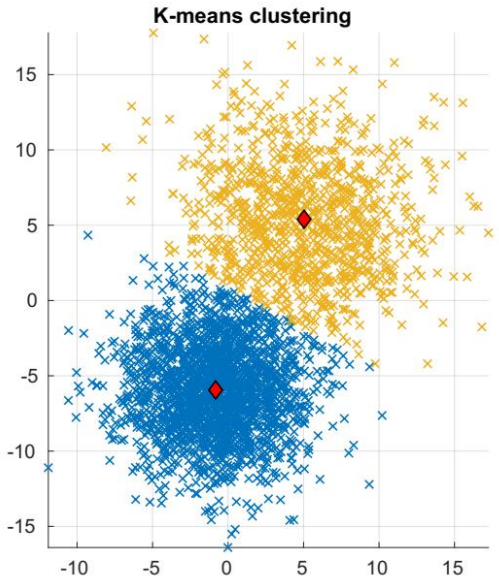
For developing your algorithm, you will use the 2D data provided in the file `DataForKmeans.mat`

"mykmeans" will:

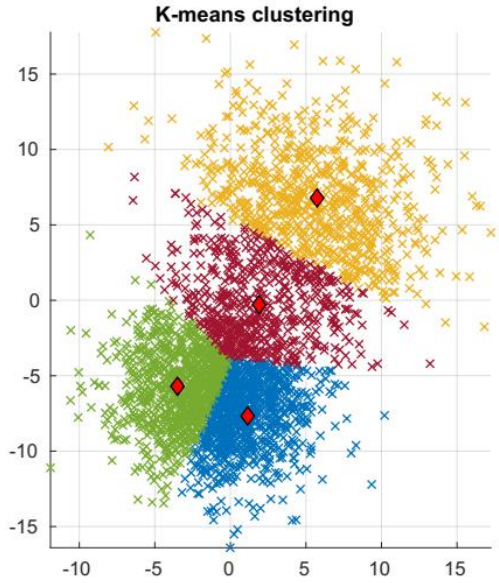
1) Plot the value of the function that is being minimized by k-means as a function of the iterations.
2) Return a matrix `C` of size `k x 2`, containing the final position of the centroids, one centroid per row.
3) Return a vector `V` specifying the cluster to which each point belongs. This vector will have `n` elements, and the i^{th} element will contain an integer in the range `[1, k]` indicating the cluster to which the i^{th} element belongs (note that `V(i)` specifies the cluster having `C(V(i),:)` as its centroid).
4) Plot a figure showing the different `k` clusters obtained, in different colours, together with their centroids – make sure the centroids are clearly visible in the figure. For example, using the data provided in the file `DataForKmeans.mat`, the call to the function:

```
[C, V] = mykmeans(Data, 2);
```

produced the following figure (here the centroids are drawn as large red diamonds).



The call to the function: `[C, V] = mykmeans(Data, 4)` produced the following figure:



[**Warning**: the above figures should only be taken as references since every run of `k`-means is likely to produce different clusterings…].

Finally, note that this exercise is about implementing k-means yourself, so you cannot use the `kmeans` function available in Matlab.

Marking Criteria

In order to obtain full marks for each question, you must answer it correctly and completely.
Marks will be given for writing compact, vectorised code and avoiding the use of "loops" (*for* or *while* loops) for carrying out operation on matrix and vector elements.