7COM1033 Semester B 2019/20
Jan T Kim, `j.t.kim@herts.ac.uk` 8 March 2021

Machine Learning: Worksheet 2

Aim

In this lab you will explore the use of Matlab facilities for numerical processing and their use for implementing Machine Learning algorithms. You will also install the Netlab toolbox and get started with using it. After completing this sheet successfully, you will

- know how to write Matlab scripts and functions, and how to use them,
- have a working installation of the Netlab Toolbox,
- have carried out K-means clustering and graphically displayed its results
- have experience of implementing some essential elements of unsupervised vector quantisation in code.

Scripts and functions are important devices to organise your programming, including your work in the practical sessions. For getting the most out of the labs, make sure that you understand the syntax and language principles, and that you develop your code on this basis. Please see Appendix A for some further guidance and advice.

This worksheet is not marked and no submission is required.

Tasks

Get yourself an overview of the the "Programming" section of the Matlab online documentation, and work through the "Scripts" and "Functions" subsections in detail. Make sure that you understand the relevant Matlab syntax as specified in the "Create Functions in Files" part of the "Functions" subsection. Also, revise the Introduction to Matlab by David F. Griffiths, available at

`http://www.maths.dundee.ac.uk/ftp/na-reports/MatlabNotes.pdf`.

Refer to these materials as you complete this worksheet.

1. **Install Netlab.** Visit the website

   `https://www2.aston.ac.uk/eas/research/groups/ncrg/resources/netlab/downloads`

   and download from there

   - the Toolbox 3.3 file (`netlab3.zip`),
   - the accompanying Help Files (`nethelp3.zip`),
   - the file `foptions.m`.

   In case of any issues, the files are also available from the 7COM1033 Canvas site. Install these Netlab using the instructions on the Netlab website. Place the `foptions.m` file in the same directory where you stored the files extracted from `netlab3.zip`.

2. **K-Means.** Further guidance for this task is provided in Appendix B for this task. You may read that before trying the task, or try it first and use the Appendix if you get stuck, or to compare your solution with the suggested approach.

   (a) Create a matrix of data points $D = \{(1,1), (2,1), (3,1), (5,5), (5,6), (6,5), (6,6)\}$ in which each point occupies one row in your matrix. Also create a matrix of initial codevectors $C = \{(2,2), (3,2)\}$.

(b) Show the data and codevectors in one scatterplot, using different colours and symbols for the data points and the codevectors.

Can you estimate from the plot what the initial clusters will be? And on that basis, where would you expect the codevectors to be after the first iteration of the K-means algorithm?

(c) Study the documentation of Netlab's `kmeans` function and use that to run K-means clustering on the data, starting with the initial codevectors. Show the codevectors produced by `kmeans` in a plot, along with the data. You may need to pass in options to the `kmeans` function, which you can obtain from `foptions`. A minimal example of running `kmeans` is

```
% create matrix of data points as variable d
% create matrix of code vectors as variable c
options = foptions();
kmeans(c, d, options);
```

(d) Consult the documentation on configuring `kmeans` via `options`. Set `options(14)` to `1` to have each call to `kmeans` run one single iteration only. Use this to graphically display the data and the codevectors for each iteration, starting with the initial codevectors (see above) and ending when the algorithm has converged (i.e. when the codevectors do not change in an iteration).

(e) Use your code to `kmeans` to cluster the data provided in the `data.mat` file.

(f) Can you construct a simple data set in which multiple results of running K-means are possible, i.e. where the objective function S has more than one minimum?

You are welcome and encouraged to discuss this question in small groups. However, in the interest of avoiding spoilers for lab classes later in the week, please do not post your example data sets on Canvas or other generally accessible places before 15 March.

3. **Simple Competitive Learning.** Write a function that performs one step of Simple Competitive Learning. The function should take a data matrix, a matrix of initial codevectors, and a learning rate as parameters. It should choose a data point at random, find the winning codevector, move it towards the data point, return the codevector matrix updated accordingly. Use the following function as a starting point:

```
function codevectorsUpdated = scmStep(data, codevectors, learningRate)
% choose a data point randomly
% compute distances of all codevectors to the data point
% determine winning codevector
% compute changes to winner's components
% assign codevectors with the winner updated to codevectorsUpdated
end
```

Add your code to actually implement the functionality outlined in the five comments.

Use the function to compute a step of Simple Competitive Learning, starting from the data points D and codevectors C you used for K-means above. Display the data, the initial codevectors, and the codevectors after an update in a scatterplot. Experiment with different learning rate settings, and verify that the winning codevector gets closer to the data point as the learning rate increases towards 1. Can you demonstrate how learning "overshoots" when you use a learning rate greater than 1?

Compute a second step by using the updated codevectors produced in the first step as the starting codevectors for the next step. Develop a loop that performs 10 steps of learning, and finally write a function to carry out n steps, given a data matrix and a learning rate.

Appendix

A Organising Code

We very strongly recommend that you save all substantial work you do during practical labs in files. This will allow you to revise what you have done later. Furthermore, keeping records of your work in files is a key part of professional practice in all computer based areas of work.

The suggestions below are provided to help you organise your work. Engaging with these will also prepare you to work in roles that require clear provenance of results, auditability, and collaborative software development. The guidance for the K-means task (Appendix B) illustrates some of the concepts outlined below.

A.1 Comments

Comments are lines in the code which are ignored by the computer but provide information to human readers. A key benefit of keeping code in files is that you can write comments on your code. The "acid test" question to ask when writing code and making decisions in the process is: "Will I benefit from this information at some time in the future?" If the answer is "yes", it's worth a comment.

In Matlab, the percent character (`%`) is used for comments. For example:

```
% somehow this 3x2 matrix has a problem
m = [1, 2; 3 4, 5 6]
```

A.2 Scripts

Running a script is essentially the same as typing the contents of the script into the Command Window. In Matlab, a script is a file with a name ending with "`.m`". In order to use a script, it has to be in a directory that Matlab has been pointed to using the `addpath` command, then the script can be run by typing the file name (without the "`.m`" extension) in the Command Window.

Finding yourself typing the same sequence of commands several times repetitively, e.g. as you work towards completing a worksheet task, is a strong indication that the code is worthwhile to be put in a script. It is a good idea to write one script per worksheet task, and to organise your scripts into a consistent directory structure (e.g. one directory per module, and within these one subdirectory for each worksheet). And consider putting the module and worksheet into a comment as well.

A.3 Functions

Functions allow you to write code to carry out a process with defined types for inputs and the resulting output. Input is passed into functions using its parameters, and the function hands back the output via its return value. In Matlab, functions are stored in `.m` files (like scripts). There should be only one function in a file, and the name of the file (without the `.m` extension) should be the same as the name of the function.

The key advantage of functions over scripts is that parameters are variable, and so the same function can be used to process many different data items. This greatly enhances code re-use. If you think the code you work on has potential use beyond your current specific task, consider writing a function. And if you find code that you wrote in a previous script has use for your current work, definitely put it into a function.

A.4 Discussing and Working Together

Working with fellow students is generally highly encouraged. Seeing the work of others, comparing it to your own, and discussing differences from various perspectives is a great way to deepen, firm up and extend understanding of the subject matter.

When working in groups, you will achieve the best learning results if all members take turns writing code and developing ideas, approaches and solutions. A great way to check results of group work and further firm up your understanding is to solve a task again, on your own, without looking at notes or code from the group session.

B Further Guidance on Getting Started with K-Means

- Recall Matlab's notation for matrices which uses commas to separate values, semicolons to separate rows, and square brackets to enclose the entire matrix. Your matrix should look like this when displayed in the Command Window:

  ```
  d =

     1   1
     2   1
     ...
  ```

  Create the matrix of initial codevectors using the same method.

- For the scatterplot, consult the documentation of the `scatter` function and notice the optional parameters to control the size of the markers (`sz`), the type of the marker (`mkr`), colour and filling of markers, and more. The `hold` function is useful to add further points to a scatterplot, e.g. to add the codevectors to a plot of the data points.

- Use the `kmeans` function of Netlab to perform K-means clustering on the data points, starting with the initial codevectors. Display the codevectors you obtained in the Command Window. Are they where you would expect them to be? Show the codevectors computed by `kmeans` along with the data, building on the plotting code you developed previously.

- The task requires you to display the same type of plot for multiple codevector sets (resulting from multiple iterations of the K-means algorithm). Noticing the pattern of carrying out *the same* operation on *different, variable data* allows you do deduce, as explained in Appendix A.3, that a function is suitable here, and that the data matrix and the codevector matrix are the parameters this function needs. Let's choose the name `datacodeplot` for this function. Now you have all the information to write the "skeleton" of the function as follows:

  ```
  function datacodeplot(data, codevectors)
  % displaying code here
  end
  ```

  Save and run this function as shown here. As the function's body is empty, nothing will appear on the screen. As a quick way of verifying that the mechanics are working, you could insert a statement like `disp('datacodeplot function running');` into the function and run it again to see that the "datacodeplot function running" message appears. Now you are ready too populate the body with `scatter` and `hold` statements. Use the statements you developed to show the data and the initial codevectors as a point of departure. You will likely need to modify these statements. However, do not change the skeleton code. Assuming that you created variables `d` for the data and `c0` for the initial codevectors, calling `datacodeplot(d, c0);` should display these. Notice how you can now bring up a plot that is specifically tailored to show K-means results with just one line in the Command Window or in a script.

  Additional bonus: If you code the `datacodeplot` function as outlined here, you will find that you can re-use it for the Simple Competitive Learning task.

- Matlab's `pause` function to stop execution of further statements until you (i.e. the user) presses a key. Therefore, a basic piece of code step through a run of the K-means algorithm and to display each step is:

```
% configure kmeans to do one iteration per call
options = foptions;
options(14) = 1;
c = c0;
for i in 1:10
    datacodeplot(d, c);
    pause;
    % call kmeans and store the new codevectors in cNew
    % check whether cNew is different from c
    c = cNew;
end
```

Start by developing the code to call `kmeans` and to assign the new codevectors to the variable `cNew`. Then develop an `if` statement to determine the codevectors have changed, and initially use that to print a message like "K-means has converged" in this case. You will likely find that you see this message displayed multiple times, as the algorithm takes fewer than 10 iterations to converge.

Once you have noticed the iteration at which convergence occurs you can manually adjust the number of iterations of the `for` loop accordingly. But with another dataset, that may cause the loop to finish before convergence. Therefore, replace the `for` loop with a `while` loop and adapt the `if` condition you developed earlier to decide whether to stop iterating. This should ensure that all steps up to convergence are displayed, regardless of when convergence is reached.