PRA1004 Scientific Computing - Lab Assignments

Frans Oliehoek
<frans.oliehoek@maastrichtuniversity.nl>

Week 4

Overview Lab 4

In this lab, you will gain experience in a number of 'tools' that are important in nearly every discipline of science:

- least squares regression,
- k-means clustering, and
- PCA

Also, along the way, you will learn some useful Matlab commands.

At the end of this lab, you will not have to hand in the report yet: you should hand in a single report for this week and week 5.

1 General Curve Fitting & Least Squares Solutions

This assignment uses the lab kit of week 3.

As might be clear by now, it is not always the best choice to use an *interpolent* (i.e., a function that passes exactly through the data points). The reason is that this may give you a very complex function, especially when your measurements have some noise, while to underlying relation in fact is much simpler.

1.1 More Noisy Data

1. Redo the plot of the noisy data of assignment 2. Open up the basic fitting tool. Try some of the different fits.
   NOTE: the basic fitting tool is Matlab only. (However, you can also try and fit polynomials of order 1–4 in Octave.)

   (a) What seems to be the best model?

   (b) What is its sum of squared errors (SSE)?

2. The data was in fact generated from a (fictitious) series of noisy measurements. Run `GenerateNoisyMeasurements` to get a new data set. Plot it together with your previous best fit. Is it still a good fit?

3. Repeat the procedure a number of times, what do you now think is the best model? That is, make a prediction on the value for some x (say, $x = 100$), that I will generate (using the same `GenerateNoisyMeasurements` function) when correcting your report.

4. What (theoretical) assumption of least-squares fitting is violated by the measurements generated by `GenerateNoisyMeasurements`?

Notes: the problem of selecting what fitting function to use is called "model selection" and has a rich literature in statistics.

1.2 Solving a Least-Squares Problem

In this assignment we will use a different data generating function `GenerateNoisyMeasurements2`.

1. Generate and plot N=5 data points using a call to `GenerateNoisyMeasurements2(5)`. (tip: reuse code from `linSysPolyFit.m`).

2. The old code (i.e., your implementation of `linSysPolyFit.m`) tries to fit a 2nd order polynomial. What is the problem you encounter when use this script for the newly generated data set? (How many equations with how many unknowns are there?)

3. Fortunately, there is a solution: 'solving' the system of equations using ' \' (also called left division). Implement this. Plot the resulting fitted polynomial.

4. The matrix C is also called the *design matrix* and it contains the so-called basis functions. What basis functions did you use here?

2 K-Means Clustering: Random Data

In this assignment you will implement k-Means and run it on randomly generated data. The basic goal is to get the script `script2.m` to execute.

1. Try running `script2` what error message do you get? What needs to be fixed? Finish the implementation in `NearestCentroids.m` by replacing 'TODO' by working code. Note that you can check your result by executing `script2`.

2. Also finish the implementation in `UpdateCentroids.m`

3. Now we arrive at 'part 3'. Look at the data imported from `3randomclusters`, pick a suitable value for `num_clusters`. If all is well, you parts 1,2 basically resulted in a working implementation of k-Means. However, an important part of k-Means, the random initialization, is not implemented yet.

   (a) Implement random initialization by finishing the implementation of `kMeansInitialization.m`.

   (b) Run `script2` a couple of times. Is the result as expected?

3 K-Means Clustering: Image Compression

In this exercise, we apply k-means to a real-life application: the compression of an image. There are multiple ways to compress images, one of which is to reduce the number of colors in the image. That is rather than storing for each pixel a value (between 0 and 256) for the amount of red, green and blue, we can create a color map that contains a fixed, but small number of colors and use only those colors for the image. That is, for each pixel we now only need to store an index (to a color in the colormap).

1. How do you think k-means can be used to compute this color map? (what is k?)
2. Open `script3.m` again finish the implementation by replacing the 'TODO's.
3. What are the smallest values for `K` and `max_iterations` that seem to give nice results (in your opinion).
4. How much space does the compressed image need to be stored? How much the normal one? What is the compression ratio achieved?
5. BONUS (only if you have the time!): adapt your code such that it runs fast also on `cityhall-large.jpg`. What are the crucial modifications to achieve efficiency.

4 PCA: Handwritten Digit Compression

In this assignment you will use PCA to perform compression of handwritten digits. You will do this by completing `script4.m` It consists of four parts. The corresponding parts in the code are indicated. You should answer all questions in your report and include all the generated figures. Also you should explain for each of the figures what it shows.

1. First, get familiar with the data set by showing some numbers. In particular varify that the first 100 datapoint are 0's, the second 100 are 1's etc. Next, finish the implementation of `DisplayDigitArray.m`. It should do the following:
   - show on each row the first 7 examplars of each digits.
   - to do this it uses '`subplot`'. Have a look at help to understand what the command does.
   - finish the missing part using `DisplayI(i)` to show the i-th digit in the data set.
2. At this point the $k = 50$ first direction with highest variance are stored in U (the directions $u^{(j)}$ is the j-th column).
   (a) Use your new knowledge of subplot to plot in a new figure the 12 first directions. (Hint: the 'directions' have the same size as the digits, so you can use `DisplayDigit` to show them).
   (b) Compute the $k \times N$ matrix Z of compressed coordinates.
3. This section of the code investigates the principle components themselves.

(a) Assign `Z_1` and `Z_2` with the first two principal components of all data points.

(b) The script makes a scatter plot: it shows the first and second principal component of each data point. Also, it draws a random data point (with index `index`) locates its coordinates, `z_index`, in the (`Z_1`, `Z_2`)-plot and plots this point. Now, we also want to plot the nearest point: Finish the implementation such that you find the data point (i.e., digit) that is closest to `z_index` in the (`Z_1`, `Z_2`)-plot.

(c) Run this section a couple of time. What do you notice? How do neighbors relate to each other? Can you connect this to the results of figure 2? Put one or two figures in the report to support your findings.

4. Here we use the found $k = 50$ first direction with highest variance together with the compressed coordinates (stored in Z) to plot an image side by side by its PCA reconstruction.

   (a) Finish the implementation such that you find the PCA reconstruction of the i-th image and store it in `reconstruct_i`.

   (b) Suppose that you find the quality provided by this reconstruction adequate, what is the least number of images that you have to store for the PCA-encoded version to be more compact than the uncompressed images?

5 PCA: ECG Analysis

Moves to next week.