1.3 2019/03/16 20:52:40

General questions

Q: How can I display the images of characters in the data set?

A: There is a Matlab sample function:

> /afs/inf.ed.ac.uk/group/teaching/inf2b/cwk2/matlab_codes/disp_one.m

Q: Are there any templates for the functions I should use?

Yes, please see:

> /afs/inf.ed.ac.uk/group/teaching/inf2b/cwk2/matlab_codes (for **Matlab**)
> /afs/inf.ed.ac.uk/group/teaching/inf2b/cwk2/python_codes (for **Python**)

Q: How should I measure the user time taken?

A: Ideally it should be measured with a function equivalent to the UNIX/Linux `time` command **(/usr/bin/time)**, which gives three measurements - 'real time', 'user CPU time', and 'system time', where '*user CPU time*' is the one requested in the coursework. However, none of Matlab time measurement functions seem to be equivalent to the time command.

Thus, please use `tic` and `toc` for Matlab.

For Python, please use `time.clock()`.

Please note that user time should be measured on a DICE machine. If you have measured the time on a non-DICE machine, e.g. your laptop, and you have no time to try a DICE machine, it is fine that report the time in your report, but please indicate so, and describes the computing environment you used.

Q: I get a (slightly) different user time every time I measure. Should I take the average?

A: No, it is not necessary for this coursework - it's fine that you measure the time only once, and report it.

Submissions

Q: Are we restricted to submitting only the files described in the assignment or can we make separate functions and turn them all in?

A: The latter. Please submit all pieces of your code including those for helper functions, so that markers can run your code.

Q: What should I submit?

Read the coursework specifications first, and the checklist:

> http://www.inf.ed.ac.uk/teaching/courses/inf2b/coursework/inf2b_cwk2_2019_checklist.txt

Task 1

Q: My k-means clustering is very slow (e.g. takes more than a few minutes).

A: See the vectorisation technique for distance calculation described in the following section for Task 2.

Q: Task **1.7** - I don't understand what you mean by the visualisation of cluster regions on the 2D-PCA plane whose position is specified by a point vector.

A: Sorry about the unclearness. The task asks you to show the cross section of cluster regions, where the plane that gives the section is defined by the first two principal component axes and the position vector. It's not a projection of data on to the plane.

The position vector specifies the origin of the plane in the original coordinate system, so that the following equality holds:

$$vector_x = vector_p + vector_r$$

where $vector_x$ is a point (vector) in the original coordinate system, $vector_r$ points $vector_p$ from the origin of the plane whose origin is given as the position vector $vector_p$. Note that I assume all the vectors have the same dimension (D). Letting $vector_y$ being the corresponding point on the plane, $vector_y$ is given as $(vector_{v1}, vector_{v2})^T \cdot vector_r$, where $vector_{v1}$ and $vector_{v2}$ are the eigenvectors corresponding to the first and second principal components, respectively.

If the $vector_y$ is a point on a 2D-plane whose dimension is smaller than D, you can extend $vector_y$ to the one in a D-dimensional space, by padding $(D-2)$ zeros to it, so that the original (y_1, y_2) becomes $(y_1, y_2, 0, ..., 0)$.

Regarding the plotting of cluster regions, this task asks you consider a grid of nbins-by-nbins on the plane, where each grid point holds the cluster number (index) that the point belongs to. An example of showing regions can be found in Lab 4 (k-NN classification).

Task 2

Q: My k-NN code takes hours to finish... How k-NN classification can be implemented efficiently?

A: K-NN requires the calculation of Euclidean distances between each pair of data points in a test set and those in a training set, which is the main bottleneck when data sets are large.

The key technique for accelerating the run time of your Matlab or Python code is vectorisation. Generally speaking, the more number of data points you put into a vector or matrix and the more you reduce for-loops, the faster your code runs (as long as sufficient amount of memory is available).

Q: Details:

We assume that a test data set X is a N-by-D matrix, and a training data set Y is a M-by-D matrix, where N is the number of test instances, M is the number of training instances, and D is the dimension of the feature vector space.

$$X = \begin{pmatrix} X_1 \\ . \\ . \\ X_N \end{pmatrix} = \begin{pmatrix} x_{11} & ... & x_{1D} \\ . & & . \\ . & & . \\ x_{N1} & ... & x_{ND} \end{pmatrix} \qquad \text{Test data set}$$

$$Y = \begin{pmatrix} Y_1 \\ . \\ . \\ Y_M \end{pmatrix} = \begin{pmatrix} y_{11} & ... & y_{1D} \\ . & & . \\ . & & . \\ y_{M1} & ... & y_{MD} \end{pmatrix} \qquad \text{Training data set}$$

where

- $X_i = (x_{i1}, ..., x_{iD})$ denotes the i-th test data, and
- $Y_j = (y_{j1}, ..., y_{jD})$ denotes the j-th training data.

(**NB:** Here, we use a row vector to denote the feature vector of a data, as opposed to a column vector in Inf2b lectures and notes)

The (squared) Euclidean distance d_{ij} is given as

$$d_{ij} = ||X_i - Y_j||^2 = (X_i - Y_j) \cdot (X_i - Y_j)^T$$

In k-NN, for each test data instance X_i, we need to calculate $d_{i1}, ..., d_{iM}$, to find the k closest data points in Y, so that, for all test instances, we calculate distance matrix DI of N-by-M:

$$D = \begin{pmatrix} D_1 \\ . \\ . \\ D_N \end{pmatrix} = \begin{pmatrix} d_{11} & ... & d_{1M} \\ . & & . \\ . & & . \\ d_{N1} & ... & d_{NM} \end{pmatrix}$$

In Matlab, DI can be efficiently computed with `pdist2()` function, e.g.

```
DI = pdist2(X, Y, 'squaredeuclidean');
```

in Python,

```
DI = scipy.spatial.distance.cdist(X, Y, 'sqeuclidean')
```

However, this coursework ***does not allow*** you to use those functions, but ask you to implement similar functions of your own. The following outlines how it can be done.

At first, note that d_{ij} can be decomposed in the following manner.

$$d_{ij} = (X_i - Y_j) \cdot (X_i - Y_j)^T = X_i \cdot X_i^T - 2X_i \cdot Y_j^T + Y_j \cdot Y_j^T$$

Let

$$XX = \begin{pmatrix} X_1 \cdot X_1^T \\ \cdot \\ \cdot \\ X_N \cdot X_N^T \end{pmatrix}, \quad YY = \begin{pmatrix} Y_1 \cdot Y_1^T \\ \cdot \\ \cdot \\ Y_M \cdot Y_M^T \end{pmatrix}$$

so that XX and YY are N-by-1 and M-by-1 matrices, respectively. Then DI can be obtained with standard matrix operations:

$$DI = (XX, ..., XX) - 2 \cdot X \cdot Y^T + (YY, ..., YY)^T$$
$$\longleftarrow M \longrightarrow \qquad\qquad \longleftarrow N \longrightarrow$$

In Matlab, `repmat()` can be used to repeat copies of matrix.

If the size of DI is too large for your computer to handle, and causing a lot of memory swapping in your machine, consider defining the matrix as a single precision variable rather than double.

Further speed-up is possible if you discard terms that are not relevant to classification.

Q: Debugging tips:

Coding for vectorisation is sometimes tricky and potentially buggy. Make sure, using a small number of samples, that your vectorised code gives the same distance as the one without vectorisation.

Q: How can I calculate covariance matrices using vectorisation?

A: See the lecture slides for Lecture 8.

Q: What should I do with the zero determinant of a covariance matrix?

A: Please try the `logdet()` function provided in the course directory.

Q: Which type of covariance matrix should I use - the one normalised by N or $N-1$?

A: Please use the one normalised by N, because MLE is assumed.