Inf2B Coursework 2

(Ver. 1.1)

Submission due: 4pm, Wednesday 5th April 2017

Hiroshi Shimodaira and Chenyang Zhao

1 Outline

The coursework consists of three tasks, Task 1 – k-means clustering, Task 2 – PCA and Gaussian classifiers, and Task 3 – neural networks.

You are required to submit (i) three reports, one for each task, (ii) code, and (iii) results of experiments if specified, whose details are given in the corresponding task sections below. Some of the code and results of experiments submitted will be checked using an automated marking system, so that it is essential that you follow the syntax of function or file format specified. No marks will be given if it does meet the specifications. Efficiency of code will not be assessed, but the code should be well written / formatted so that other people can understand.
This coursework is out of 100 marks and forms 12.5% of your final Inf2b grade. For higher marks, not only the programming and experiments specified, but also good investigations and discussions are required.

This coursework is individual coursework - group work is forbidden. You should work alone to complete the coursework. You are not allowed to show any written materials, data provided to you, results of your experiments, or code to anyone else. Never copy-and-paste material into your coursework and edit it. You can, however, use the code provided in the lecture notes, slides, and labs of this course. High-level discussion that is not directly related to this coursework is fine.

Please note that assessed work is subject to University regulations on academic conduct:
`http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

Programming: Write code in Matlab/Octave or python+numpy+scipy+matplotlib. Your code should run on DICE machines with the software installed. There are some functions that you should write the code by yourself rather than using those of standard libraries available. See section 4 for details.

This document assumes Matlab programming. If you use Python, replace the Matlab filename extension (`.m`) with the one for Python (`.py`) for the file names shown in the following sections.

Extensions: The School does not normally allow late submissions. See: `http://www.inf.ed.ac.uk/student-services/teaching-organisation/for-taught-students/coursework-and-projects/late-coursework-submission` for exceptions to this rule, e.g. in case of serious medical illness or serious personal problems. Any extension request must be made to the ITO, not the lecturer.

2 Data

2.1 Data for Task 1 and Task 2

The first two tasks employs a subset of the CIFAR-100 image data set `http://www.cs.utoronto.ca/~kriz/cifar.html`. Each image is represented as 32-by-32 pixels in RGB, so that the dimension of feature vector is $32 \times 32 \times 3 = 1024 \times 3 = 3072$.

Your data set is stored in a Matlab file named '`train_data.mat`' and '`test_data.mat`' located in your coursework-data directory (denoted as *YourDataDir* hereafter) :

`/afs/inf.ed.ac.uk/group/teaching/inf2b/cwk2/d/`*UUN*

where *UUN* denotes your UUN (DICE login name).

There are 5 classes in total, and there are 2000 training samples and 200 testing samples for each class. You can load the data set file in Matlab with the 'load' function, e.g. loading the training set by:

`load('`*YourDataDir*`/train_data.mat');`

which contains the training data, `train_x`, and corresponding labels, `train_y`.
Note that the data sets require pre-processing before you carry out experiments in Tasks 1 and 2. Please follow the instruction in the coursework web page.

2.2 Data for Task 3

The data for Task 3 is stored in a plain-text file named '`task3_data.txt`' in *YourDataDir*. See Task 3 in section 3 for details.

3 Task specifications

Task1 – K-means clustering [30 marks]

In Task-1, we use the training data, `train_x`, for experiments with k-means clustering.

1.1 Write a Matlab function for K-means clustering and save it as '`Task1/MyKmeans.m`'. The syntax of the function should be as follows.

```
[C, idx, SSE] = MyKmeans(X, k, initialCentres, maxIter)
```

where

X	N-by-D data matrix (of floating-point numbers in double-precision format, which is the default in Matlab), where N is the number of data points, and D is the dimension of each data point. Note that each sample is represented as a row vector rather than a column vector.
k	The target number of clusters
initialCentres	Initial cluster centres in the k-by-D matrix, where each row represents a cluster centre.
maxIter	Maximum number of iterations (optional)
C	Cluster centres in the k-by-D matrix.
idx	Cluster indices in the N-by-1 vector, where idx[i] indicates the cluster number (starting from 1) of i-th data point.
SSE	The mean squared error (the one defined in Equation 3.1 in Lecture note 3) for each iteration in the $(L+1)$-by-1 vector, where the first element is the SSE for the initial cluster centres, and L is the number of iterations done.

[10 marks]

1.2 Write a Matlab script that runs k-means clustering on the training set, and save the script as '`Task1/kmc8.m`'. The specifications of the script are as follows.

- Run k-means clustering for $k = 8$ with the initial cluster centres being the first k data points, i.e. `X(1:k,:)`.
- Save the cluster centres obtained (i.e. `C`) in the MAT-file named '`Task1/result_C.mat`'.
- Calculate a confusion matrix in which the original labels are used as the true class, and save it as the MAT-file named '`Task1/result_confusion_matrix.mat`'. Each element of the matrix should represent the number of corresponding samples rather than the normalised ratio. Display the confusion matrix, which you should show in your report as well.
- Obtain the final SSE (the mean squared error after the last iteration), which you report in your report.

After you wrote the script, run the script so that it saves the output to the Task1 directory.

[10 marks]

1.3 Investigate the effect of initial values, i.e. how different initial cluster centres result in different cluster centres, for which employ SSE to measure the clustering performance. Write a script that runs k-means clustering with $k = 8$ for at least 30 times with different initial cluster centres that are chosen randomly, and save the script as '`Task1/ival_test.m`'.

Run the script, measure SSE for each trial, and summarise the result using a graph, and gives discussions in the report. Give your ideas to improve the clustering performance.

[10 marks]

Task 2 – Image classification [30 marks]

In this task, we carry out image classification experiments with Gaussian classifiers. Due to the fact that the number of training samples is not large enough to estimate covariance matrices, we pre-process the data to reduce the dimensionality with the principal component analysis (PCA), before we run experiments with Gaussian classifiers.

In the following classification experiments, assume a uniform prior distribution over classes, and **use the maximum likelihood estimation (MLE) to estimate model parameters**.

2.1 Write a Matlab function file '`Task2/compute_pca.m`', to compute the principal components of a data set. The syntax of the function is as follows:

```
[EVecs, EVals] = compute_pca(X)
```

`X` is the same format as the one for `MyKmeans()` in Task 1. `EVecs` are the eigenvectors (stored in a D-by-D matrix `EVecs`), and `EVals` are the corresponding eigen values $\{\lambda_i\}_{i=1}^{D}$ (stored in a D-by-1 vector). The eigenvalues should be sorted in descending order, so that λ_1 is the largest and λ_D is the smallest, and i'th column of `EVecs` should hold the eigenvector that corresponds to λ_i.

Eigenvectors are not unique by definition in terms of scale (length) and sign, but we make them unique in this coursework by putting the following additional constraints, which your `compute_pca()` should satisfy.

- The first element of each eigenvector is non-negative. If it is not the case, i.e. if the first element is negative, multiply -1 to the eigenvector (i.e. $\mathbf{v} \leftarrow -\mathbf{v}$) so that it gets the opposite direction.
- Each eigenvector is a unit vector, i.e. $\|\mathbf{v}\| = 1$, where $\mathbf{v}$ denotes an eigenvector. As far as you use Matlab's eig() or Python's numpy.linalg.eig(), you do not need to care about this, since either function ensures unit vectors.

Hint: you will probably find the Matlab function eig helpful. In Python, you can use numpy.linalg.eig

[5 marks]

2.2 Apply `compute_pca` to the training data, and save the largest ten eigenvalues in the MAT-file named '`Task2/evals.mat`', and save the corresponding eigenvectors in the MAT-file named '`Task2/evecs.mat`'. Save the script for this as '`Task2/result_pca.m`'.

[5 marks]

2.3 After reducing the dimensionality of data to 100 with PCA, fit a Gaussian model with a full covariance matrix for each class and use them to classify the test data. Note that the same transformation for dimensionality reduction as the one for the training data should be applied to the test data.

Report the determinant of the covariance matrix of each class, and the confusion matrix and correct classification rate for the test set.

Save the code for this as '`Task2/gaussian_d100_full.m`'. Save the confusion matrix in the MAT-file named '`Task2/confmat_d100.mat`'.

[10 marks]

2.4 Carry out an investigation into the effect of the data dimensionality on classification performance (i.e. correct classification rate), report the result, and give discussions. If you use any scripts for this, save them in 'Task2' directory, and indicate the file names in your report.

[10 marks]

Task 3 – Neural networks [40 marks]

In this task, you will be asked to implement neural networks for binary classification problems, in which input feature is represented as a two-dimensional vector $(x_1, x_2)^T$. We assume that decision regions are defined with polygon(s), whose specifications are given in the polygon specification file 'task3_data.txt' [1] in *YourDataDir*. The file is a plain-text file, in which each line specifies the name of the polygon and the coordinates of its vertices $\{(x_{p1}, x_{p2})\}_{p=1}^{P}$, where P is the number of vertices. The following is an example of the file.

Polygon_A: -1 -0.5 6 1.25 6 6.25 1 6
Polygon_B: 2.5 3 3.5 3 3.5 3.5 2.5 3.5

where two polygons, Polygon_A and Polygon_B, are defined. In each line, the first two numbers (e.g. -1 and -0.5 for Polygon_A) from the left specify the coordinate (x_{11}, x_{12}) of the first vertex, followed by the coordinate (x_{21}, x_{22}) for the second vertex, and so one. You will see that each polygon has four vertices, meaning a quadrangle in this case.

3.1 Consider a single neuron with a unit function, whose output is defined as $y(\mathbf{x}) = h(\mathbf{w}^T\mathbf{x})$, where $h(a)$ is a step function such that $h(a) = 1$ if $a > 0$, and $h(a) = 0$ otherwise [2]. Implement this neuron as a Matlab function:

```
[Y] = hNeuron(W, X)
```

where `X` is a N x D data matrix, `W` is a $(D + 1)$ x 1 weight matrix, `Y` is a N x 1 output matrix. Save the function as 'Task3/hNeuron.m'.

Note that this function can take more than one input vector stored in a matrix `X`, where each input vector is represented as a row vector rather than a column one, and gives corresponding output as a vector `Y`.

[2 marks]

3.2 Similar to `hNeuron()` above, but consider another neuron which employs the logistic sigmoid function $g(a) = \frac{1}{1+\exp(-a)}$. Implement this neuron as a Matlab function:

```
[Y] = sNeuron(W, X)
```

and save it as 'Task3/sNeuron.m'.

[2 marks]

3.3 Find the structure (i.e. connection of neurons) and weights of the neural network that classifies the inside and periphery of Polygon_A as Class 1 (i.e. $y(\mathbf{x}) = 1$), and the outside as Class 0 (i.e. $y(\mathbf{x}) = 0$), where each neuron is modelled with `hNeuron()`.

This task is meant for you to work using pen and paper (and calculator), but it is also fine that you write a piece of code to find the weights. If it is the case, save the script or function as 'Task3/find_hNN_A_weights.m'.

Write the weights in a plain text file 'Task3/hNN_A_weights.txt' in the following format. Let w_{ji}^{ℓ} denote the weight of neuron j at layer ℓ from neuron i at layer $\ell - 1$ [3]. You write each w_{ji}^{ℓ} in a separate line, for $\ell = 1, ...$, $j = 1, ...$, and $i = 0, 1, ...$, so that the first line contains w_{10}^{1} followed by w_{11}^{1} and w_{12}^{1} in the second line and the third line, respectively. The format of each line should be as follows:

[1] You are not allowed to show this file of yours to anyone else.
[2] NB: The step function defined here is slightly different from the one in the lectures.
[3] The input layer where input date are fed is regarded as layer 0 (zero). The output node of a single-layer neural network is in layer 1.

W(ℓ,j,i) : <*the value of* w_{ji}^{ℓ}>

where "<*the value of* w_{ji}^{ℓ}>" is the actual value of the weight. For example, if $w_{10}^{1} = 2.35$, the first line should look like this:

```
W(1,1,0) :  2.35
```

Spaces are only allowed just before and after ":", and none in other places.

In your report, show the structure of the network and explain how you found the weights.

[7 marks]

3.4 Implement the neural network above as a function:

```
[Y] = hNN_A(X)
```

and save it as '`Task3/hNN_A.m`', where X and Y follow the same format as was shown in Task 3.1.

[4 marks]

3.5 Using `hNN_A()`, write a script that plots the decision regions in a 2D space, and save the code as '`Task3/plot_regions_hNN_A.m`'. Save the graph as a PDF file named '`Task3/regions_hNN_A.pdf`'.

[3 marks]

3.6 We now consider the decision regions formed with Polygon_A and Polygon_B, whose classification rule is shown below:

Class 1 : $A \cap \bar{B}$
Class 0 : $\bar{A} \cup B$

where A and B denote the inside and periphery of the corresponding polygon, $\bar{B}$ denotes the complement of B.

Implement the corresponding neural network as a function:

```
[Y] = hNN_AB(X)
```

and save it as '`Task3/hNN_AB.m`'. Note that each neuron should be modelled with `hNeuron()`.

[5 marks]

3.7 Using `hNN_AB()`, write a script that plots the decision regions in a 2D space, and save the code as '`Task3/plot_regions_hNN_AB.m`'. Save the graph as a PDF file named '`Task3/regions_hNN_AB.pdf`'.

[3 marks]

3.8 We now consider another network `sNN_AB()` obtained by replacing all nodes of `hNeuron()` with those of `sNeuron()` in `hNN_AB()`, so that each neuron is now modelled with `sNeuron()`. Implement the neural network as a function:

```
[Y] = sNN_AB(X)
```

and save it as '`Task3/sNN_AB.m`'. Note that you will need to modify the weights to approximate the decision regions properly.

[4 marks]

3.9 Using `sNN_AB()`, write a script that plots the decision regions in a 2D space, and save the code as '`Task3/plot_regions_sNN_AB.m`'. Save the graph as a PDF file named '`Task3/regions_sNN_AB.pdf`'.

[3 marks]

3.10 Investigate and discuss the decision regions for `sNN_AB()`, clarifying how and why they are different from those for `hNN_AB()`.

[7 marks]

4 Functions that are not allowed to use

Since one of the objectives of this coursework is to implement basic algorithms for machine learning effectively, you are not allowed to use those functions in standard libraries listed below. You should write the code by yourself using the basic operations of arithmetic for scalars, vectors, and matrices. If it is the case, use a different function name from the original one in standard libraries (e.g. MyCov() for cov() as shown in the table below). You may, however, use them for comparison purposes, i.e. to check your code.

Description of function	Typical names	Suggested name to implement
compute the mean	mean()	MyMean()
compute the covariance matrix	cov()	MyCov()
compute Gaussian probability densities	mvnpdf()	
k-means clustering	kmeans()	
compute confusion matrix	confusion()	
other utilities for classification		

You may use those functions or operations:

Description	Typical names
sum function	sum()
cumulative sum	cumsum()
square root function	sqrt()
exponential function	e, exp()
logarithmic function	log(), ln()
matrix transpose	transpose(), '
matrix inverse	inv()
determinant	det()
log determinant	logdet() ··· available in Inf2b cwk2 directory
eigen values/vectors	eig()
sort	sort()
sample mode	mode()

(NB: the list is not exhaustive)

5 Submission

You should submit your work electronically via the DICE submit command by the deadline. No submission of printed document is required.

Since marking for each task will be done separately, you should prepare *separate reports* for the three tasks, and save your report files in PDF format and name them '`report_task1.pdf`', '`report_task2.pdf`', and '`report_task3.pdf`'. Remember to place your student number and the task name prominently at the top of each report. Do not indicate your name anywhere. Your report should be concise and brief, 1 or 2 pages long, for each task.

Create a directory named `LearnCW`, copy the PDF files of your reports in it. Create sub directories, `Task1`, `Task2`, and `Task3`, under `LearnCW`, and copy all of your code for each task to the corresponding sub directory.

Make sure that each task directory contains necessary pieces of code so that coursework markers can run your code properly in DICE. There is a check list in the coursework web page, which shows a list of files to submit and the contents of reports. Submit your coursework from a DICE machine using:

```
submit inf2b 2 LearnCW
```