Inf2B Coursework

(Ver. 0.9)

Submission due: 4pm, Friday 3rd April 2020

Hiroshi Shimodaira

1 Outline

The coursework consists of two tasks, Task 1 – data analysis and classification with multivariate Gaussian classifiers, Task 2 – neural networks.

You are required to submit (i) two reports, one for each Task, (ii) code, and (iii) results of experiments if specified, using the electronic submission command. Details are given in the corresponding task sections below. Some of the code and results of experiments submitted will be checked with an automated marking system in the DICE computing environment, so that it is essential that you follow the syntax of function and file format specified. No marks will be given if it does not meet the specifications. Some helper tools to check your files and function template files will be provided. Please check the following coursework web-page frequently to see any updates.

`https://www.inf.ed.ac.uk/teaching/courses/inf2b/courseworkSchedule.html`

Efficiency of code and programming style (e.g. comments, indentation, and variable names) count. Those pieces of code that do not run or that do not finish in approximately five minutes on a standard DICE machine will not be marked. This coursework is out of 100 marks and forms 25% of your final Inf2b grade.

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, the data provided to you, results of your experiments, or code to anyone else. This includes posting your coursework to the internet and making it accessible to other people not only during the coursework period, but also after that. Never copy-and-paste material of other people (including those available on the internet) into your coursework and edit it. You can, however, use the code provided in the lecture notes, slides, and labs of this course, excluding some functions described later. 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 misconduct:

`http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

For late coursework and extension requests, see the page: `http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests`

Note that any extension request must be made to the ITO, and not to the lecturer.

Programming: Write code in Matlab(R2018a)/Octave or Python(version 3.6)+Numpy+Scipy+Matplolib. Your code should run on standard DICE machines without the need of any additional software. 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 programming in Matlab. For Python, put all the specified functions into a single file for each Task, so that `task1.py` for Task 1, and `task2.py` for Task 2. Output data should be stored in Matlab's MAT binary format.

2 Data

2.1 Data for Task 1

The coursework employs the Anuran Calls (MFCCs) Data Set introduced by J. Colonna *etal.*.[1]

You data set file, `'dset.mat'`, which is a subset of the original data set, should be found in your coursework data directory (denoted as *YourDataDir* hereafter) :

[1] `https://doi.org/10.1007/978-3-319-46307-0_13`

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

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

You can use Matlab's `load()` function to load the data set in the following manner:

`load(`*pathname*`);`

where *pathname* denotes the absolute pathname of your data set file. Once you load the data set, you will find the following variables.

Matlab variable (Class)	Description
`X`[N, D] (double))	feature vectors
`Y_family`[N,1] (int32)	family class labels
`Y_genus`[N,1] (int32)	genus class labels
`Y_species`[N,1] (int32)	species class labels
`list_family`[4,1] (cell)	family class names
`list_genus`[8,1] (cell)	genus class names
`list_species`[10,1] (cell)	species class names

where N and D denotes the total number of samples and the dimension of feature vector (D=24), respectively.

Among the three different levels of taxonomic rank provided in the original data set, we use 'species' in the coursework. There are ten different species, so that the number of classes for classification is ten, i.e., $C = 10$. The variable, `Y_species(i)`, contains the integer number that corresponds to the species of i-th sample, whose feature vector is `X(i,:)`. Hereafter, `Y` denotes `Y_species`.
The variable, `list_species`, holds the list of species names.

The following table shows the number of samples for each species in the original data set, which may be different from the number samples in your data set.

Family	Genus	Species	# of samples
Leptodactylidae	Leptodyctylus	Leptodactylus fuscus	222
	Adenomera	Adenomera andreae	496
		Adenomera hylaedactyla	3049
Hylidae	Dendropsophus	Hyla minuta	229
	Scinax	Scinax ruber	96
	Osteocephalus	Osteocephalus oophagus	96
	Hypsiboas	Hypsiboas cinerascens	429
		Hypsiboas cordobae	702
Bufonidae	Rhinella	Rhinella granulosa	135
Dendrobatidae	Ameerega	Ameerega trivittata	544

The data set has not been split in two sets for training and testing. You need to split the data set according to the instructions described later.

2.2 Data for Task 2

The data for Task 2 is stored in the plain-text file named `'task2_data.txt'` in *YourDataDir*. For details, see the Task 2 specifications.

3 Task specifications

Task1 – Anuran-Call analysis and classification [50 marks]

Task 1.1 [5 marks]

(a) Write a Matlab function `task1_1()` that

- calculates the covariance matrix, `S` and correlation matrix, `R`, for the whole data set `X`, using the maximum likelihood estimation (MLE),
- saves `S` as `'t1_S.mat'`,

- saves `R` as '`t1_R.mat`'.

Save the code as '`task1_1.m`'. Note that, hereafter, function and file names are case sensitive, and your code should save output files in the *current working directory*. The syntax of the function should be as follows.

```
function task1_1(X, Y)
```

where

`X`	N-by-D matrix of feature vector (of floating-point numbers in double-precision format, which is the default in Matlab), where N is the number of samples, and D is the the number of elements in a sample. Note that each sample is represented as a row vector rather than a column vector.
`Y`	N-by-1 label vector (of int32) for `X`. `Y(i)` is the class number of `X(i,:)`.

(b) Run the following:

```
function task1_1(X, Y)
```

Make sure that the two output files are created properly. It will be a good idea that you write a script to run the above.

Task 1.2 [5 marks]

Look into the correlation matrix, `R`, you obtained, and describe your findings in your report, using graphs.

Task 1.3 [10 marks]

(a) Write a Matlab function `task1_3()` that

- calculates the eigenvectors, `EVecs` and eigenvalues, `EVals`, of a covariance matrix, and calculates the cumulative variance, `Cumvar`,
- finds the minimum number of PCA dimensions to cover each 70%, 80%, 90%, 95% of the total variance, and store the values to a vector `MinDims`,
- saves the eigenvectors to a file named '`t1_EVecs.mat`',
- saves the eigenvalues to a file named '`t1_EVals.mat`',
- saved the cumulative variances to a file named '`t1_Cumvar.mat`',
- saves the the numbers of minimum dimensions, `MinDims`, to a fle named '`t1_MinDims.mat`',

Save the function as '`task1_3.m`'.
The syntax of the function should be as follows.

```
function task1_3(Cov)
```

where `Cov` is a D-by-D covariance matrix (double).
The specifications of the variables are as follows.

`EVecs`	D-by-D matrix (in double)
`EVals`	D-by-1 vector (in double)
`Cumvar`	D-by-1 vector (in double)
`MinDims`	4-by-1 vector (in int32)

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 program should employ.

- 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.

(b) Run the following:

```
task1_3(S);
```

In your report, show a graph of cumulative variance.

(c) Plot all data on a 2D-PCA plane, clarifying data of different classes, and show the graph in your report.

Task 1.4 [25 marks]

(a) Write a Matlab function `task1_mgc_cv()` that carries out a classification experiment with multivariate Gaussian classifiers, using k-fold cross validation, and save the code as '`task1_mgc_cv.m`'. The syntax of the function is as follows

```
function task1_mgc_cv(X, Y, CovKind, epsilon, Kfolds)
```

where `CovKind` is the type of covariance matrix - 1 for full covariance matrix, 2 for diagonal covariance matrix, and 3 for shared covariance matrix, `epsilon` is a scalar (double) for the regularisation of covariance matrix described in Lecture 8, in which we add a small positive number (ϵ) to the diagonal elements of covariance matrix, i.e. $\mathbf{\Sigma} \leftarrow \mathbf{\Sigma} + \epsilon I$, where I is the identity matrix, `Kfolds` is the number of folds (partitions) in k-fold cross validation. Assume a uniform prior distribution over class, and use MLE for the estimation of model parameters.

At first, the function should split the data set in `Kfolds` partitions for cross validation, whose information is stored in a N-by-1 vector, `PMap`, where `PMap(i)` holds the partition number that i-th sample is assigned to, and save it to a file named '`t1_mgc_<Kfolds>cv_PMap.mat`', where `<Kfolds>` is the number of folds.

For each fold, `p`, the function should

- estimate the mean vector and covariance matrix for each class from the samples that do not belong to partition `p`.
- save the mean vectors ((`Ms`)to '`t1_mgc_<Kfolds>cv<p>_Ms.mat`',
- save the covariance matrices (`Covs`) to '`t1_mgc_<Kfolds>cv<p>_ck<CovKind>_Covs.mat`',
- carry out a classification experiment using the samples of partition `p`, and save the confusion matrix (`CM`) to '`t1_mgc_<Kfolds>cv<p>_ck<CovKind>_CM.mat`',
- calculate the final confusion matrix (where each element is a relative frequency) and save it to '`t1_mgc_<Kfolds>cv<L>_ck<CovKind>_CM.mat`', where `L = Kfolds + 1`.

In the above, replace `<p>`, `<Kfolds>`, `<CovKind>`, and `<L>` with the actual values.

Details of partitioning algorithm for k-fold cross validation and the variables to save will be specified in a separate sheet.

(b) Run the function with `epsilon=0.01` and `Kfolds=5` for each `CovKind=1,2,3`, and report the accuracy (correct classification rate) in your report.

Task 1.5 [5 marks]

Using `CovKind`=1 (i.e. full covariance), investigate how the classification accuracy changes with respect to the regularisation parameter, `epsilon`. Plot a graph and describe your findings in your report.

Task 2 – Neural networks [50 marks]

In this task, you 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 '`task2_data.txt`' [2] 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.

[2] You are not allowed to show this file of yours to anyone else.

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 coordinates (x_{11}, x_{12}) of the first vertex, followed by the coordinates (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.

Task 2.1 [3 marks]

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 [3]. Implement this neuron as a Matlab function:

```
function [Y] = task2_hNeuron(W, X)
```

where `X` is a N-by-D data matrix (double), `W` is a (D+1)-by-1 weight matrix (double), `Y` is a N-by-1 output vector (double). Save the function as '`task2_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`.

Task 2.2 [3 marks]

Similar to `task2_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:

```
function [Y] = task2_sNeuron(W, X)
```

and save it as '`task2_sNeuron.m`'.

Task 2.3 [8 marks]

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 `task2_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 '`task2_find_hNN_A_weights.m`'.

Let w_{ji}^{ℓ} denote the weight of neuron j in layer ℓ from neuron i in layer $\ell-1$ [4]. Normalise your weights in such a way that $\max_i |w_{ji}^{\ell}| = 1$. Write the weights in a plain text file '`task2_hNN_A_weights.txt`' in the following format.

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:

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 $w_{10}^{1} = 0.35$, the first line should look like this:

```
W(1,1,0) :  0.35
```

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.

Task 2.4 [5 marks]

Implement the neural network above as a function:

[3] NB: The step function defined here is slightly different from the one in the lectures.

[4] 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.

```
function [Y] = task2_hNN_A(X)
```

and save it as '`task2_hNN_A.m`', where X and Y follow the same format as was shown in Task 3.1.

Task 2.5 [4 marks]

Using `task2_hNN_A()`, write a script that plots the decision regions in a 2D space, and save the code as '`task2_plot_regions_hNN_A.m`'. Save the graph as a PDF file named '`t2_regions_hNN_A.pdf`'.

Task 2.6 [6 marks]

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

$$\begin{aligned} &\text{Class 1}: \quad A \cap \bar{B} \\ &\text{Class 0}: \quad \bar{A} \cup B \end{aligned}$$

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:

```
function [Y] = task2_hNN_AB(X)
```

and save it as '`task2_hNN_AB.m`'. Note that each neuron should be modelled with `task2_hNeuron()`.

Task 2.7 [4 marks]

Using `task2_hNN_AB()`, write a script that plots the decision regions in a 2D space, and save the code as '`t2_plot_regions_hNN_AB.m`'. Save the graph as a PDF file named '`t2_regions_hNN_AB.pdf`'.

Task 2.8 [5 marks]

We now consider another network `task2_sNN_AB()` obtained by replacing all nodes of `task2_hNeuron()` with those of `task2_sNeuron()` in `task2_hNN_AB()`, so that each neuron is now modelled with `task2_sNeuron()`. Implement the neural network as a function:

```
function [Y] = task2_sNN_AB(X)
```

and save it as '`task2_sNN_AB.m`'. Note that you will need to modify the weights to approximate the decision regions properly.

Task 2.9 [4 marks]

Using `task2_sNN_AB()`, write a script that plots the decision regions in a 2D space, and save the code as '`task2_plot_regions_sNN_AB.m`'. Save the graph as a PDF file named '`t2_regions_sNN_AB.pdf`'.

Task 2.10 [8 marks]

Investigate and discuss the decision regions for `task2_sNN_AB()`, clarifying how and why they are different from those for `task2_hNN_AB()`.

4 Functions that are not allowed to use

Since one of the objectives of this coursework is to understand and implement basic algorithms for machine learning, 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
Pairwise (squared) Euclidean distance	pdist2()	MySqDist()
Compute the mean	mean()	MyMean()
Compute the covariance matrix	cov()	MyCov()
Compute Gaussian probability densities	mvnpdf()	
K-NN classification	fitcknn()	run_knn_classifier()
K-means clustering	kmeans()	my_kMeansClustering()
Compute confusion matrix	confusion()	comp_confmat()
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 cwk directory
Eigen values/vectors	eig()
Sort	sort()
Sample mode	mode()
Vectorisation helpers	bsxfun(), arrayfun()

(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 two tasks, and save your report files in PDF format and name them '`report_task1.pdf`' and '`report_task2.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 for each task.

Create a directory named `LearnCW`, copy all of the requested files to the directory, but *do NOT put the data set files in it.*

A checklist will be available from the coursework web page. Submit your coursework from a DICE machine using:

```
submit inf2b cw1 LearnCW
```