THE UNIVERSITY *of York*

BSc, BEng, MEng and MMath Examinations, 2013-2014

COMPUTER SCIENCE, COMPUTER SYSTEMS AND SOFTWARE ENGINEERING, COMPUTER SCIENCE AND MATHEMATICS, COMPUTER SCIENCE WITH ARTIFICIAL INTELLIGENCE
Part B

MACHINE LEARNING AND APPLICATIONS (MLAP)
Open Examination

Issued at:
12 February 2014

Submission due:
14 May 2014

Your attention is drawn to the Guidelines on Mutual Assistance and Collaboration in the Student's Handbook and the Department's 'Guide to Assessment Policies and Procedures' (http://www.cs.york.ac.uk/exams/statementonassessment/).

All queries on this assessment should be addressed to **James Cussens & Suresh Manandhar**.

Your examination number must be written on the front of your submission. You must not identify yourself in any other way.

Should you wish to request an extension see the appropriate section of the Department's 'Guide to Assessment Policies and Procedures'.

- All tasks should be attempted.
- A submission after the deadline will be marked initially as if it had been handed in on time, but the Board of Examiners will normally apply a lateness penalty.
- In addition to the submission, you should also provide a write-up of your methods and discussion as detailed below.
- The programs you write and the write-up should be submitted electronically through the web page http://www.cs.york.ac.uk/submit.
- The code must be written in the Matlab, Octave or Python programming languages.
- All extra materials for this assessment can be found at http://www-module.cs.york.ac.uk/mlap/assessment/13/, henceforth know as the assessment directory.

- All solutions must be justified, all programs fully documented. Inefficient solutions will be penalised.
- Programs which correctly implement the required algorithms get credit for doing so. To get further marks you need to write programs that are:
  1. efficient in both time and space;
  2. well documented; and
  3. use the most appropriate methods for the given problems.

1 Linear regression and Logistic regression

Data

The data for this part of the assessment is provided in the assessment directory in one CSV (comma separated values) file `stock_price.csv`.

Data format

The data consists of the stock volume and stock price information for a single company. Each row consists of two columns:

- Stock volume (in column 1) : total number of stock sold in a single day
- Stock price (in column 2) : averaged stock price in a single day

Each row provides stock volume and stock price information for a single day and successive rows provide information for consecutive days. Thus, the *first* row in the file provides stock volume and stock price for the *first* day. The *second* row in the file provides stock volume and stock price for the *second* day etc.

Task 1 (30 marks)

Implement a *linear regression* algorithm in either Octave (preferred) or Matlab or Python that accurately predicts the *stock price* for the **current day** given:

- stock price and stock volume of the **past 10 days**

You must submit your own implementation without using existing implementations (e.g. mnrfit, mnrval or glmfit). You may use built-in optimisation routines such as fminunc but nothing else. If using Python only builtin functions and modules from the standard library may be used.

- Calculate and report the *mean squared error* (MSE) using 2-fold cross-validation.

In your report, you are required to experiment with different choices of features and produce a bar chart of MSE values for different choices of features. The aim is to discover via experimentation the features (or variable combinations) that are good predictors of stock price. For example, you may want to use the rate of change (i.e. difference in stock price one day apart) as an indicator of the upward or downward trend. In addition to inventing new features, you are required to experiment with a number of different types of functions e.g. different order polynomials and other suitable functions of your choice.

Which set of features perform well? Explain using the experimental results.

Code format

For Tasks 1, 2 and 3, you should provide a single Matlab or Octave (preferred) or Python file. For Task 1, you will implement single function `linear` with the header:

```
function MSE = linear(InputFileName)% MSE is mean squared error
```
(or Python equivalent).

Hint: This task can be broken down into several sub-parts:

1. Firstly, you need to expand each row by adding information from the previous 10 rows. Thus only the 11th entry onwards from the training data will have all the information needed and only these will form part of the actual training data.
2. Next, you will need to do *basis expansion* to incorporate the different features you wish to incorporate. In the beginning, you can ignore this step until you have a working system that includes cross-validation and evaluation.
3. You will split the training data into 2 parts and complete the cross validation code. This should include code for computing the MSE (mean squared error).
4. You will implement the linear regression code being careful not to use existing regression routines.

Task 2 (15 marks)

Modify the code from Task 1 to implement a multi-class logistic regression classifier for 5 classes - *nochange, up, down, sharp_up, sharp_down* with their intended meaning as:

- *nochange* (class 0) – the stock price is within $\pm 5\%$ of the previous day
- *up* (class 1) – the stock price is within $+10\%$ (and $> 5\%$) of the previous day
- *down* (class 2) – the stock price is within -10% (and $< -5\%$) of the previous day

- *sharp_up* (class 3) – the stock price is *greater than* 10% of the previous day
- *sharp_down* (class 4) – the stock price is *less than* 10% of the previous day

For this purpose, you will add one extra column to each row. You will need to add code to compute the class.

As in Task 1, you must submit your own implementation without using existing implementations (e.g. mnrfit, mnrval or glmfit). You may use built-in optimisation routines such as fminunc but nothing else. You *must* implement a **multi-class** logistic regression classifier. You will **not** receive any marks for this part if you implement a *binary* (i.e. two class) classifier.

Using the same choices of features as in Task 1, report the *hard classification accuracy* using cross-validation as in Task 1. Describe your findings using a bar chart for different choices of features.

Which set of features perform well? How do these compare to those for Task 1? Explain using the experimental results.

Code format

You will implement a single function `logistic` for this task with the header:
```
function Accuracy = logistic(InputFileName)
```
(or Python equivalent).

Task 3 (15 marks)

In this task, the aim is to augment both the linear and logistic regression models from Task 1 and Task 2 with regularisation.

Augment the linear regression model from Task 1 with **either** L1 (as in lasso) **or** L2 (as in ridge) regularisation. Find the optimal λ regularisation parameter using cross-validation.

Re-run the evaluation from Task 1 and describe the results you obtain. How does the regularised linear regression perform in comparison?

Similarly, augment the logistic regression model from Task 2 with **either** L1 **or** L2 regularisation. Find the optimal λ regularisation parameter using cross-validation.

Re-run the evaluation from Task 2 and describe the results you obtain. How does the regularised logistic regression perform in comparison?

Code format

You will implement two functions `reglinear` and `reglogistic` for this task with the headers:

```
function MSE = reglinear(InputFileName)

function Accuracy = reglogistic(InputFileName)
```

(or Python equivalent).

2 Bayesian networks

The data for this part of the assessment is provided in the assessment directory in one CSV (comma separated values) file `bndata.csv`. In addition, there is a file `bnstruct.csv` which defines the structure of a Bayesian network.

`bnstruct.bn` contains the *adjacency matrix* **A** for a directed acyclic graph (DAG) G where $A_{ij} = 1$ if there is an edge from node i to node j in G and $A_{ij} = 0$ if there is no edge from node i to node j in G. Since `bnstruct.bn` contains 8 rows (and 8 columns) it follows that G has 8 nodes. You can name your nodes 0–7 or 1–8, whichever is easier for you, but in what follows it is assumed that the nodes are called 0–7.

`bnstruct.bn` represents **A** such that the first row in `bnstruct.bn` contains information on all edges from node 0 in G. You can see that there is only one such edge going from node 0 to node 2. The graph G given by `bnstruct.bn` is acyclic and we will consider it to be the structure of a Bayesian network where each node represents a binary variable taking the values 0 or 1.

Each row in `bndata.csv` represents a datapoint. Each datapoint represents an observation of the 8 random variables. There are 10000 datapoints.

Task 4 (20 marks)

Use the data in `bndata.csv` to estimate the parameters of the Bayesian network whose structure is the DAG G. Use a Bayesian approach where you put a uniform prior on each parameter, and where each parameter is estimated independently. For each parameter, report the mean of the resulting posterior distribution as the estimate. You may use Python (preferred), Matlab or Octave to do this.

You will implement a function `bnbayesfit` for this task with the header:

```
function fittedbn = bnbayesfit(StructureFileName,DataFileName)
```

(or Python equivalent).

```
['1', '0', '0', '0', '0', '0', '1', '1']
['1', '0', '1', '0', '0', '0', '1', '1']
['1', '0', '0', '0', '0', '0', '1', '0']
['0', '0', '1', '0', '0', '0', '1', '1']
['1', '0', '0', '0', '0', '0', '1', '0']
['0', '0', '1', '0', '0', '0', '1', '1']
['1', '0', '1', '0', '0', '0', '1', '1']
['1', '0', '1', '1', '0', '0', '1', '1']
['0', '0', '0', '0', '0', '0', '1', '0']
['1', '0', '1', '0', '0', '0', '1', '1']
```

Figure 1: Joint instantiations sampled from an 8 variable Bayesian network.

The choice of datatype for `fittedbn` is up to you, as long as it contains the required parameter estimates (the values of which you must include in your report). However, since you will use `fittedbn` in Task 5 it makes sense for you to choose a datatype that makes it easier to do that task.

Task 5 (20 marks)

In this task you write code to sample full joint instantiations from the probability distribution defined by `fittedbn`. You may use Python (preferred), Matlab or Octave to do this.

Implement a function `bnsample` for this task with the header:

```
function sample = bnsample(fittedbn,nsamples)
```

(or Python equivalent). `nsamples` is the number of samples to generate. Your function `bnsample` should work for any first argument (of the right type), not just the particular Bayesian network produced by doing Task 4.

To give you and idea of what `bnsample` should compute, Fig 1 shows you what my model answer computed when `nsamples` was 10.

1. Explain, using examples, how you could use your sampler to produce estimates for any conditional probability defined by the Bayesian network.
2. Discuss (in at most 40 words) when this can be done efficiently and when this is inefficient.
3. Show, using at least 4 examples, how the accuracy of your estimates increases as the number of samples increase.