Thessaloniki, 11/04/2024

Tel. : +30 231 039 8384
Fax: +30 231 039 8389
E-mail: astaras@act.edu

FINAL COURSE PROJECT DESCRIPTION

COMPUTER SCIENCE 340

INTRODUCTION TO ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING USING PYTHON

A Project on Data Processing/Visualization and Artificial Neural Networks Programming

This document contains the description for the CS340 Python coding project, which has 3 parts (two coding sections and a report). The combined 3-part project is worth 30% of the total CS340 course grade.

You are encouraged to work in study groups and exchange opinions, however each student must submit their own code and project report, which must be written from scratch by the student themselves (no similarities with the deliverables of other students will be tolerated). This is a code red assignment: no direct AI assistance is allowed in code generation, text authoring or syntax, only in spell-checking.

In **part A (40%)** you get to demonstrate your Python skills, importing **data from a text file**, processing it, calculating some relevant statistics, exporting the data to another text data file and visualizing results through graphs.

In **part B (40%)** you get to implement, train and experiment with an **artificial neural network** (ANN), have your code read data from a text file, process it to spot patterns, provide training output (pattern classification) and visualize results using graphs.

In **part C (20%)** you get to produce a concise 1200-2000 word **project report** in which you are requested to submit an MS Word or PDF file containing the following:

- an introduction (common to both parts of the project)
- a brief description of the design requirements (for part A, and for part B)
- brief commentary on coding development (extent to which design targets were met, etc.)
- 2x top-level flowchart diagrams (for part A and for part B)
- 2x function interdependency diagrams (for part A and for part B)
- brief commentary on machine learning experimentation and results obtained (part A, part B)
- sample screenshots of GUI graphs (for part A and for part B)
- a conclusion about skills required/developed/improved (common to both parts of the project)

Part A: F1 racing results (40%)

Write Python code which reads data from the 2023 Formula 1 racing season, processes it and displays the results. The data is available to view online at the following link: https://www.formula1.com/en/results.html/2023/races.html. Alternatively, your Python code may access it more conveniently from the comma-delimited text data file ***partA_input_data.txt*** (supplied). You will need to look up and utilize Python GUI libraries for this part of the project.

The code should implement a user text interface in the form of a menu:

- Presents an explanatory greeting and 5 option menu, which traps of erratic user input. Once each task related to an option is completed, the program should re-display the menu options.
- *Menu option 1*: Reads the 6 columns of data from file ***partA_input_data.txt*** and neatly displays it on screen (tab-delimited column alignment or equivalent).

- *Menu option 2*: Asks the user for a limit of laps to search by, then displays only the race results which involve that number of home laps or greater, sorted alphabetically by Grand Prix name.
- *Menu option 3*: Calculates the average lap time per race then saves this new information as a 7th column in file ***partA_output_data.txt***. After saving into the file, it should also read back and display all 7 columns of data on the screen (the 6 original columns + the new one based on the calculations).
- *Menu option 4*: Asks the user for a field to sort by, then whether the order should be ascending or descending. Displays on screen all data contained in the file sorted according to the user's instructions. This option refers to the 7-column file generated in option 3 and assumes it exists, otherwise the program should inform the user to execute option 3 first and then get back to option 4.
- *Menu option 5*: Calculates the total average lap time per driver -across all Grand Prix races- and presents it as a GUI column graph in a pop-up window (x-axis: driver names. y-axis: average lap time in minutes).[1]
- *Menu option 6*: Exit the program.

Upload all of your Python code files, your data input and output text files to the relevant node on the CS340 Moodle pages.

Here is a model menu description, prompting for the user for input:

```
=================================================================
F1 GRAND PRIX RACING DATA & STATISTICS FOR THE 2023 RACING SEASON
=================================================================
1. Read and display the F1 Grand Prix data for the 2023 racing season
2. Filter and sort race data based on a minimum threshold of laps
3. Calculate average lap time per race, save, retrieve, display
4. Sort and display the data based on user parameters
5. Calculate and graph total lap time per driver
6. Exit the program
```

Your program should ensure that every time the user is asked to input their choice they have the data available to see on screen, with clear, aligned labeling at the top of each column.

Part A: Deliverables overview

- **sectionA.py** : the Python text source code file
- **partA_input_data.txt** : a copy of the comma-delimited text data file supplied via Moodle
- ***partA_output_data.txt*** : a sample output file (executing your code should overwrite this)

[1] Of course each race track has a different length, topology and difficulty, while the data only mentions the winner driver's total time for each race. Consequently this comparative column graph does not really help us draw any useful conclusion about driver ability: simply regard it as a chance to demonstrate your Python coding skills.

Part B: Training an Artificial Neural network for pattern classification (40%)

Write Python code which implements a small **deterministic** multi-layer perceptron (MLP) artificial neural network (ANN), consisting of **binary state neurons**, **analogue state weights** and a **variable topology** (the latter partly based on user input). Your code should also implement a supervised training algorithm, back-propagation or similar, reading training data from a file which contains both the input binary vectors as well as the desired classification labels. Validate your MLP implementation using one of the data sets mentioned below.

You may use the following libraries: `matplotlib`, `seaborn`, `numpy`, `scipy`, `pandas`, `tensorflow` and `scikit-learn` (take your pick, you are unlikely to need all of them). You may also choose an MLP implementation from scratch, however `scikit-learn` is the library recommended by the course instructor.

The data sets for this section of the project can be found at the UCI repository for Machine Learning and are the following:

1. Wine Dataset (https://archive.ics.uci.edu/ml/datasets/Wine)

2. Car Evaluation Dataset (https://archive.ics.uci.edu/ml/datasets/Car+Evaluation)

3. Glass Identification Dataset (https://archive.ics.uci.edu/ml/datasets/Glass+Identification)

4. Bank Note Identification Dataset (https://archive.ics.uci.edu/ml/datasets/banknote+authentication)

5. Optical Recognition of Handwritten Digits Dataset. This is a tough classification challenge using a plain MLP & backpropagation, you may use other ANN architectures for this one if you wish.) (https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits)

For this section of the project, **pick one** of the data sets in the list above. Download the relevant data file and save it on your PC as *original_labeled_data.txt* (you *must* use the downloaded file, rather than receive the data directly via the internet every time your code executes).

The aim is for you to study the contents and description of the dataset you selected, understand the challenge, then code a Multi-Layer Perceptron to solve the classification problem posed by the dataset, perform a comparative analysis among multiple training/classification runs, and finally visualize the data metrics which demonstrate the classification performance of the ANN (under different sets of parameter settings).

The comparative analysis described in the project report should involve a discussion of 6 training/testing experiments, varying the following parameters:

- **the structure of the ANN**: 3 different topologies of your choice, possibly involving deep learning (i.e. 2 or more hidden layers). For full marks, let the user choose the topology (for simplicity you may wish to restrict the size of the input and output layers to match the challenge, letting them choose the number and width of the hidden layers instead).

- **the synaptic weight changing step**: at least 3 options, of which 1 fixed small step, 1 fixed larger step and 1 adaptable step.

- **the training/testing split of the data and the randomization of the data set**: try 2 options, a 50-50 split without randomization of the lines of the data set vs an 80-20 split with the dataset lines randomized.

You should implement the aforementioned training variations, tweaking your MLP to ensure that it converges to a solution within a reasonable number of epochs. Your Python program should input the text data set directly from the text data file which you downloaded from the UCI repository: all pre-processing of the data set should be done by your Python program, as opposed to other tools (i.e. no spreadsheet pre-processing, etc). All synaptic weights should be randomized at the beginning with a zero mean. Your code should output 3 x 3 x 2 = **18 training error graphs** (as images displayed and also saved in the same folder as the Python code file), varying the parameters as described above. Your code should also produce **2 weight changing graphs** for the best and fastest training combinations, as well as a **collective error graph** depicting all 18 error curves. All graphs should contain a legend clearly mentioning the comparative training parameters that they represent.

Control of the software by the user should come through a 7-choice text menu user interface, such as the following:

```
=====================================================================
USER MENU: MLP CLASSIFICATION OF THE ###### DATA SET (UCI REPOSITORY)
=====================================================================
1. Read the labelled text data file, display the first 5 lines
2. Choose the size of the hidden layers of the MLP topology (e.g. 6-?-?-2)
3. Choose the size of the training step (0.001 - 0.5, [ENTER] for adaptable)
4. Train on 80% of labeled data, display progress graph
5. Classify the unlabeled data, output training report and confusion matrix
6. Exit
```

You are encouraged to work in study groups and exchange opinions, however each student must submit their own code and project report. The source code and the project report must be written from scratch by each student: excessive similarities with the work of other students, AI chatbot-generated code and text will not be tolerated.

Part B: Deliverables overview

The full list of deliverables to be uploaded to the relevant nodes on the CS340 course Moodle page:

- **sectionB.py** : the Python text source code file
- **original_labeled_data.txt** : the original labelled text data file from the repository
- **training_data.txt** : a delimited text input data file, labeled, subset of the original
- **testing_data_unlabeled.txt** : a delimited text input data file, unlabeled, subset of the original, for testing (as a sample, the code should generate it at the end of training)
- **testing_data_labeled.txt** : a delimited text input labeled data file, subset of the original, for testing (as a sample, the code should generate it at the end of training)
- **output_data.txt** : a sample output text data file with the attempted classification labels (as a sample, the code should generate it at the end of testing)
- **sectionB_graphs.zip** : a compressed folder (*.zip) containing 18 training error graphs, grouped in 6 sub-folders (as samples, since the code should generate 3 such graphs at the end of each training/testing session). These same 18 graphs should appear in your report.

Part C: Project report (20%)

A 1200-2000 word **project report** is also required as part of the project deliverables. Students are requested to submit it in the form of an MS Word or PDF file containing the following for each of the 2 project parts:

- the title of your report, your name, the course name and a date
- a concise **introduction** (common to both parts of the project)
- a laconic **description** of design requirements and datasets for each part
- brief **commentary on coding development** (which design targets were met, etc.)
- top-level **flowchart diagrams** (2x)
- **function-interdependency diagrams** (2x)
- **samples of** input and output **data** so that the examiner can test your code (2x)
- an indicative **ANN diagram** showing neuron interconnectivity (part B only)
- brief **commentary on experimentation** with machine learning and the results which were obtained (part B only)
    - o comment on factors affecting the speed and quality of your ANN learning process, as well as how you would optimize it if you had more time and resources available: training step optimization? Training data and input data randomization? Ideas and suggestions on techniques to minimize or avoid over-training? (part B only)
- **graphs** displaying processed data results (GUI screenshots, for both sections)
- a **conclusion** about skills required/developed/improved
- a concise **bibliography** (see description below)

The project report should also contain a brief bibliography of 4-8 credible sources relevant to the tasks at hand, ideally cited in the text. Generally speaking, books and peer-reviewed academic papers (e.g. academic publications listed on scholar.google.com) are considered to be more valuable bibliography sources than loose web pages and videos encountered online. Well-cited papers tend to have more impact than less cited ones. A couple of relevant web page references are OK. Use this rule of thumb to build a credible, high impact bibliography for your report.

Notes:

1. All text and schematics should be included in a neat, carefully edited project report. Loose diagrams, charts and graphs will not be accepted.
2. The flowchart should be high level and not excessively detailed: auto-generated, unedited flowcharts or flowcharts with excessive detail will be marked down.
3. The report should contain a bibliography section with a few entries, containing the relevant course text books and any other sources of information you found useful while working on this project.
4. To maximize your assessment grade your Python source code should:
    - Execute free of bugs and according to design specifications.
    - Be well commented, clear, well-spaced and easy to read.
    - A comment header at the top of each code file should mention the program name and version number, a brief description of its functionality, the date of creation, date of last modification, your name, email address and a brief copyright statement.
    - Employ intuitive variable identifiers (e.g. *myFavColour* as opposed to *mfc*).
    - Avoid misuse or waste of programming resources.

- Clearly prompt the user for I/O, explain to them what the output means.
- Predict incorrect or erratic user input, trap it and respond appropriately.
- Demonstrate reasonable and intuitive code modularity using functions and/or methods.