American College of Thessaloniki

Computer Science 340: Artificial Intelligence

Instructor: Dr. Alexander Astaras

Student: Dusan Boljevic

Date: 18/05/2025

# Introduction

The CS340 term project was conceived to provide a comprehensive demonstration of artificial intelligence and machine learning concepts, as explored throughout the semester, by implementing them using the Python programming language. The main goal was to apply theoretical knowledge to a practical application using a real-world dataset to showcase proficiency in data preparation (cleaning), model training and model evaluation. The project is divided into two sections with **section A** being focused on basic data manipulation using a dataset of Major League Baseball (MLB) players from the Boston Red Sox team, while **section B** is dedicated to applying the machine learning concepts studied in class to a dataset of our choosing (for which I chose the ‘Predicting Academic Outcomes’ dataset) using a Multi-Layer Perceptron (MLP) neural network.

# Section A: Boston Red Sox Data Analysis

## Overview

Section A of the CS340 term project is designed to provide hands-on experience with data manipulation and analysis using pandas. The dataset consists of statistics for the Boston Red Sox players and tasks include extracting, filtering and computing new metrics from this data.

## Core Functionalities

The main features implemented in the section A code are:

* **Displaying Top Home Run Hitters** - The program reads a .txt file containing the player statistics and sorts the players by number of home runs in descending order. It then displays the top 12 players.
* **Filtering by Home Run Threshold** - The user specifies a home run threshold (number of home runs) and the program lists all the players with their number of home runs being equal or higher than the user’s threshold.
* **Calculating Derived Statistics** - The program computes two new metrics, appends them to the dataset and saves it to a new file before displaying the results to the user. The two new metrics being home runs per game (HR/G) and strikeouts per game (SO/G).
* **Custom Sorting** - The user can sort the dataset by any of the primary fields (Name, Games, Home Runs, Strikes).

## Implementation Details

* **Data Loading** - The code uses the Python library **pandas** to read the file to the dataset, specifying the correct delimiter and encoding.
* **Data Transformation** - New columns are calculated using vectorized operations for efficiency.
* **Output** - Results are displayed in the console and saved to output files as specified in the project description.
* **User Interaction** - The program runs in a loop, presenting the menu until the user chooses to exit.

## Commentary on Development

Section A of the CS340 term project was not particularly challenging in comparison with section B, at least when a useful library such as **pandas** is introduced. The code for this section had to do with very simple data manipulation with the data coming from a .csv file containing the stats of the top Boston Red Sox players. Pandas made working with this data very simple and approachable allowing me, for example, to access variables like a player’s number of home runs by simply stating the index and column name. With this in mind, calculating new variables, filtering and custom sorting could all be done in very few lines of code. This section also didn’t call for any machine learning experimentation, although it can be implemented. If I were to implement an AI/ML model to this section it would probably be one that predicts the number of home runs each player will score next season according to historical data. Of course for such a model the dataset would also need to be larger than the one used, with more variables and more data.

# Section B: Predicting Student Outcomes with MLP

## Overview

Section B addresses a more complex problem, in my case: predicting whether a student will graduate or drop out based on related data. This section demonstrates the process of preparing data, configuring and training an ANN (Artificial Neural Network), evaluating its performance and interpreting the results.

## Data Preparation and Cleaning

A critical aspect of section B, as any other data science related project, is the preparation of the dataset. The original data contains several challenges including malformed float values and missing entries. I implemented the following steps to ensure data quality:

* **Fixing Malformed Floats** - The fix\_broken\_floats function uses regular expressions (regex) to extract the first valid float from any string, correcting entries with multiple decimal points or other formatting issues.
* **Filtering and Encoding** - Only values where the target variable is equal to “Graduate” or “Dropout” were kept. The target variable is then encoded as binary (1 for Graduate and 0 for Dropout).
* **Handling Missing Values** - Rows with missing data are dropped to avoid any errors during the training of the model.
* **Feature and Target Separation** - The cleaned dataset is split into features (X) and the target (y) for use in model training and evaluation.

## Model Configuration and Training

The purpose of section B is systematic experimentation with different ANN configurations. With that in mind the program supports:

* **Customizable Topologies** - Users can define the number and size of hidden layers that will be added to a list of predefined configurations for experimentational use.
* **Learning Rate Selection** - Users can define the learning rate that will be added to a list of predefined configurations for experimentational use.
* **Train-Test Split** - The models are trained on both a 50/50 split and a 80/20 split.

For each combination of topology, learning rate and train-test split the program trains an MLP model, records performance metrics and visualizes the training process. The best-performing (most accurate) model and the fastest model (shortest training time) are tracked for further analysis.

## Visualization and Output

Visualization is important for interpreting the model’s behaviour and results:

* **Loss Curves** - The program plots the loss curve for each experiment showing the reduction in error over training iterations.
* **Collective Graphs** - The program combines all the loss curves from the experiments into a single plot for comparative analysis
* **Weight Change Plot** - Direct access to model weights is not possible with the MLPClassifier so the program approximates weight changes by calculating the absolute value of the difference between consecutive loss values.
* **Output Files** - The program generates several output files as outlined in the description:
  + Training and testing datasets (both unlabeled and labeled)
  + Model predictions with actual outcomes (output\_data)
  + Classification reports and confusion matrices
  + Zip directory containing all generated graphs

## Commentary on Development

Section B of the CS340 term project was definitely more challenging than section A since it required extra steps such as preparing the dataset and setting up a base for the models which will later be trained and compared. My take on the project uses the “Predict Students' Dropout and Academic Success” dataset in order to train the models and predict if a student will graduate or drop out of school. This section required use of Python libraries such as scikit-learn for preparing and training the models, pandas for data cleaning and manipulation, and matplotlib for displaying the **training loss curves, weight changing graphs and the collective graph comparing all the models** in order to satisfy the project requirements. The approach to a successful AI/ML project is always to first prepare and clean the data before building the models. My first challenge arose in this stage, when I realised the dataset I am working with contained what I dubbed “broken floats” (ex. 13.428.571). Having these “broken floats’ meant that I won’t be able to successfully train my model, leading me to make a function called **fix\_broken\_floats** that:

1. Looks for values that are stored as strings, because that's where the broken float values might be hiding,
2. Defines a pattern in order to extract a valid float value using a regular expression,
3. Fixes the “broken floats’ wherever it spots them by converting the string into a float with two decimals,
4. Returns a dataframe with the fixed values.

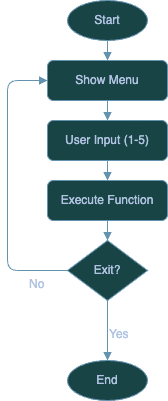
Instead of manually changing the values in order to obtain the weight changing graph, collective graph and 18 graphs for the different model configurations, I decided to make the program run different experiments for each model configuration, save the fastest and most accurate configurations (for the weight changing graphs) then create a directory to store all the generated graphs and compress it into a zip file holding the name sectionB\_graphs.zip along with generating the output files as outlined in the project description. This is achieved by the **choice4** function (the training function) running a single experiment for each possible model configuration and calling the functions: **create\_collective\_graph, create\_weight\_change\_graphs, generate\_output\_files, create\_graph\_zip**. The program contains 3 default values for topologies, 3 default values for learning rates and 2 default train-test splits in global lists which accounts for 18 different ways a model can be defined. Options 2 and 3 allow the user to add more topologies and learning rates to the global lists which increases the number of different ways a model can be trained. Another challenge I encountered developing this section of the project was getting the weight change values in order to plot them, since the MLPClassifier does not have a built-in function for getting these values. I solved the problem by using the **numpy** library to apply the following mathematical formula: |loss(t+1) - loss(t)| where t is the current iteration number.

## Flowcharts, Interdependency Diagrams and UI Screenshots

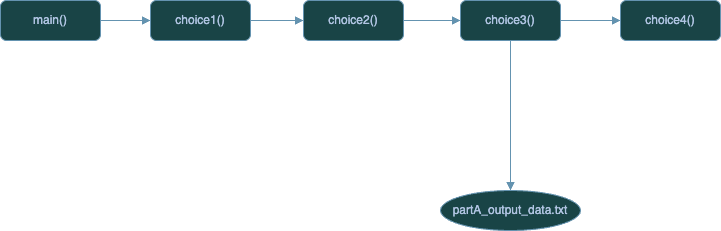
Following are the flowcharts, interdependency diagrams and ui screenshots for both sections of the CS340 Term Project.

### Section A:

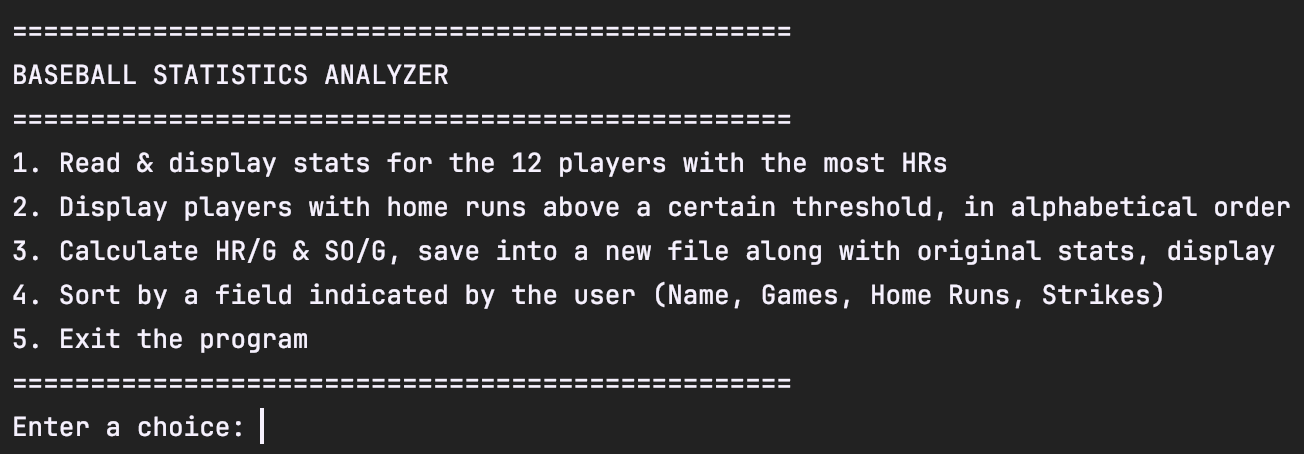
*Top-Level Flowchart for Section A*

**

*Interdependency Diagram for Section A*

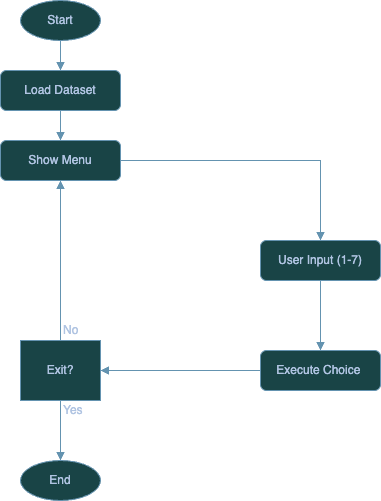
**

*UI Screenshot for Section A*

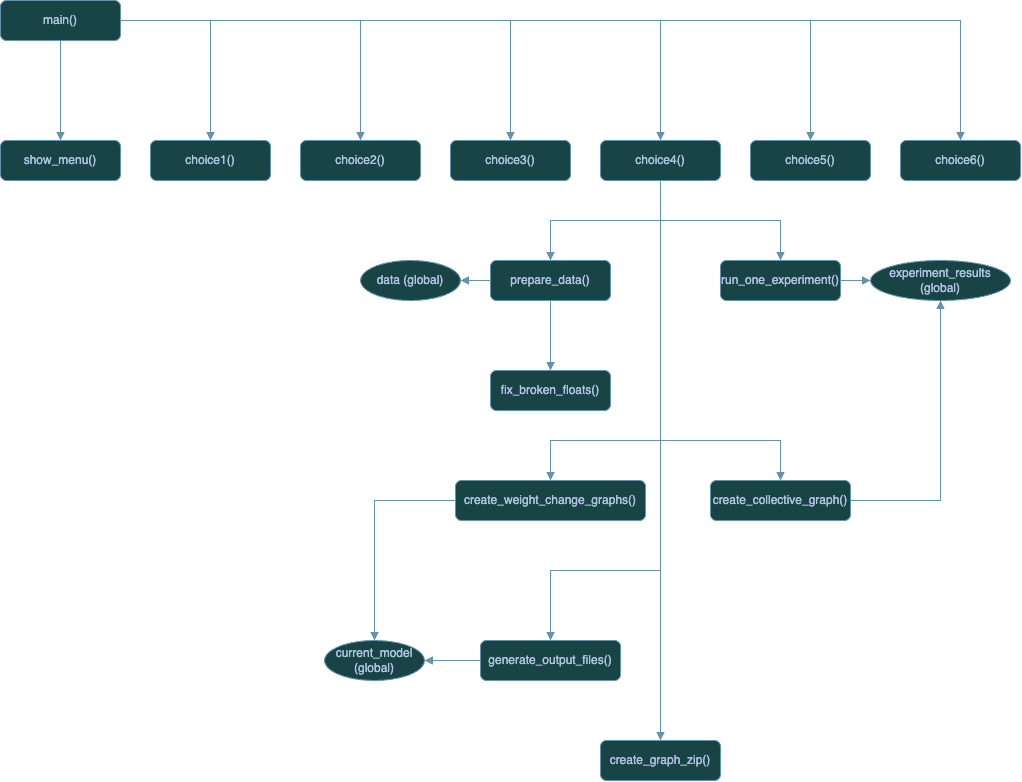
**

### Section B:

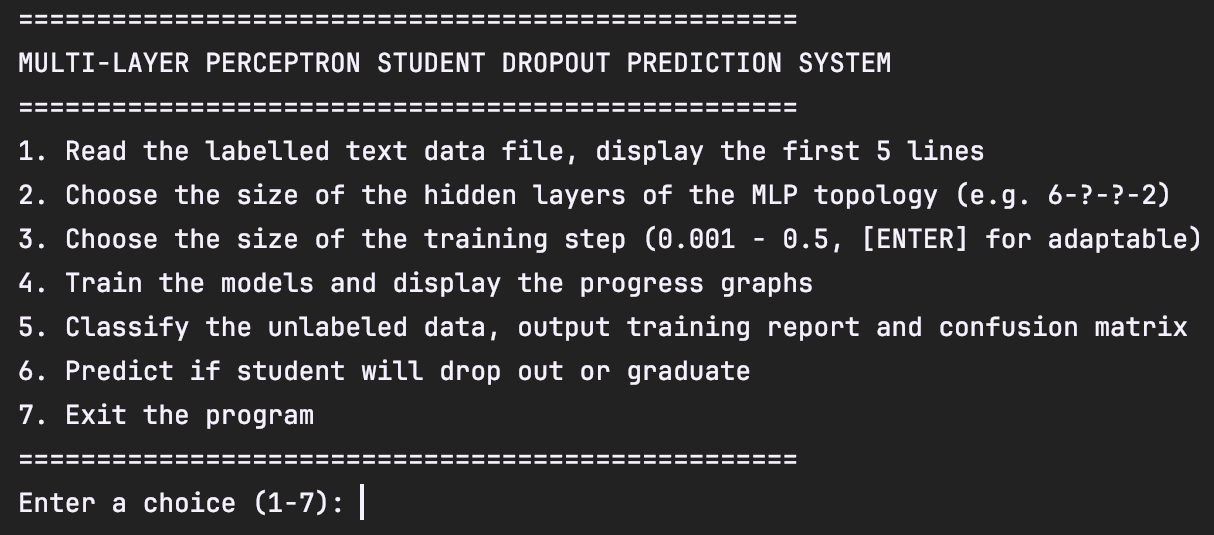
*Top-Level Flowchart for Section B*



*Interdependency Diagram for Section B:*

**

*UI Screenshot for Section B*

**

# Conclusion

This project, especially section B, required a fair amount of knowledge of software engineering, data science, Python and its libraries of which there are many. It provided an opportunity to bridge theoretical knowledge with practical application of AI/ML concepts. The project could not be done without a deeper understanding of the dataset and machine learning principles. While Python libraries such as scikit-learn and pandas offer solutions that speed up the process of data preparation and training a model, those solutions don’t help much without understanding the theory behind them. Tackling problems such as “bad” data (broken floats) and automating model evaluation across multiple MLP configurations helped me familiarize myself with the more abstract uses of the scikit-learn and matplotlib libraries. Even though I have worked with Python in the machine learning context before, mainly LSTM (Long Short Term Memory) networks, this was my first time using an MLP (Multi-Layer Perceptron) model. I also gained more experience in data preparation since the project required the use of a .txt file which brings more problems than simply importing the dataset using Python. Finally, I believe my approach is highly modular and ready for any future improvement thanks to the way I designed the program.