# CMP304 Unit 2 Report

Harry Findlay

2102552

Contents

[CMP304 Unit 2 Report 1](#_Toc166536115)

[Introduction & Background 3](#_Toc166536116)

[Data Specification 4](#_Toc166536117)

[Methodology 4](#_Toc166536118)

[Results & Conclusions 11](#_Toc166536119)

[Results for faction: Horde: 11](#_Toc166536120)

[Results for faction: Alliance: 13](#_Toc166536121)

# Introduction & Background

The AI which I decided to develop was an AI which ‘scans’ through player data and classifies players into play styles depending on their in-game statistics. The AI would also make predictions from the data provided. The data in which the AI will be using as its data set is a csv file containing World of Warcraft (WoW) battlegrounds data, where players from the two factions go head-to-head in a player versus player (PvP) environment. I wrote my AI and algorithms manually using Visual Studio Code (VS Code) where I use different classes to process different stages of the machine learning process;

DataPreperation.py

DataAnalysis.py

UserTesting.py

Main.py

The data set was sourced from Kaggle. I chose to write my AI from scratch as this allowed me to have as much control and flexibility as possible, allowing me to push and twist my AI and its learning process in any ways which I desired. This also enhances the reasoning behind why I use separate classes for different stages of learning apposed to one massive Collab sheet. The main directory is split into the following:

Main folder containing all ML classes, which is where the functionality of the MLAI is contained.

Sub folder within main folder, containing all data sets (though, only one CSV file was used).

# Data Specification

The data used within this project contains many different rows and columns of data. Despite this, only 4 columns were used in the end.

Throughout the AI’s ML process; DataPreperation, DataAnalysis and UserTesting, the three columns which were used consisted of “KB”, “HK” and “D”. These translate to player kill count, player assist count and player death count. For each of these columns, there totalled 3727 rows, meaning that this data was present for 3727 players. This is a good size as it allowed for consistent and reliable training process, a fair and numerous train/test split and still remained quick and efficient in the learning process, averaging <10 seconds to train, classify and predict.

The fourth column used was the player's chosen faction, either 'Horde' or 'Alliance'. This faction data was used by the AI to group the players into two classes, also splitting the CSV file’s data. This allows for quicker and more specific data analysis based on the chosen faction (Horde or Alliance).

Link to dataset: <https://www.kaggle.com/datasets/cblesa/world-of-warcraft-battlegrounds>

# Methodology

I opted to use VS Code to plan and develop this project. This was partially due to the multi-class or object orientated (OOP) approach to my machine learning AI (MLAI). This allowed for full the development process of this MLAI to be contained completely outside of web-based compilers and virtual machines such as Google Collabs or Jupyter Notebook.

Beginning the process was the DataPreperation class. This class was solely responsible for the importing and preparation of all data to be passed on to the later stages. This stage contains various functions and importing of libraries. Starting with the libraries that were used, we’re looking at pandas, matplotlib.pyplot (matplot) and sklearn.cluster’s KMeans (sklearn) functionality. Pandas was used to read in the data from the CSV file from the specified directory and allowed for the data to be split into groups (KB, HK and D). Matplot was used to plot the data points on to a graph to allow for preparation of a visible representation of the data. Sklearn was utilised for the unsupervised clustering of the data and then fitting the data.

A black screen with white text

Description automatically generated

The analysis functions that were created contain the following:

ReadData():

A screenshot of a computer

Description automatically generated Used to read in the CSV file using pandas and store the data in a readable format.

FilterByFaction(faction):

A screen shot of a computer

Description automatically generated Used to filter the data depending on the desired faction (parameter), returning only the data from players within the chosen faction (‘Horde’ or Alliance’)

ElbowPlot():

A computer screen shot of a program

Description automatically generated This calculates the optimal number of clusters (k) in the KMeans clustering process.

DataAnalysis is the next class used in the MLAI’s learning process. This is the class which contains all of the ‘learning’ functionality. This class is started with various sklearn libraries being imported to support with the learning processes. Following these, some lists are then initialised for uses later on in the class. Classes designed to hold the player classifications, the AI’s take on the player classifications and the total kills/assists/deaths of the player.

Following this were the functions, leading with PlayerClassification(players). This function is responsible for iterating through each and all of the players passed in via the DataPreperation phase and calculating the total number of kills, assists and deaths of each player, further adding this to the lists defined above. Following this, each and all players were then reiterated through yet again, this time filtering the data into kills, assists and deaths again but filling any void or missing data as doing so. This data, for each player, was then run through various classification rules to determine the playstyle of the player so that a tag can be applied. The tags consist of; Assault, Team Player, Defender, Scout, Casual Player and Not Classified. The appropriate tag was applied to the player depending on the sum of kills and/or assists and/or deaths of the player. A pie chart is then plotted and displayed to the user of the MLAI. In short, this is the human classification portion of the learning process.

A computer screen shot of a program code

Description automatically generatedA screen shot of a computer program

Description automatically generated

A black screen with white text

Description automatically generated

PlayerClassificationAI(players) is the next function in-line. This is similar to the non-AI version where each and every player is iterated through, sorting data, and applying the same tags where appropriate however, there are extra steps in this function that opposed to the previous. The data is then split into training and testing data, where I chose a split ratio of 60/40. Utilising this data, the MLAI then uses a ‘Decision Three Classifier’ (DTC) to compare the X training data and labels (tags) to the Y training data and classification rules. This information is then used by the MLAI to predict and sort the players into classifications itself. Overall, the MLAI learns the classification rules set by the developer, sorts the relevant player data from within the dataset and applies the tags/labels depending on what it believes the player(s) should be classified as. Polishing off this function is some simple code to plot and display a pie chart and an accuracy report where the MLAI generally has a 100% prediction accuracy.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

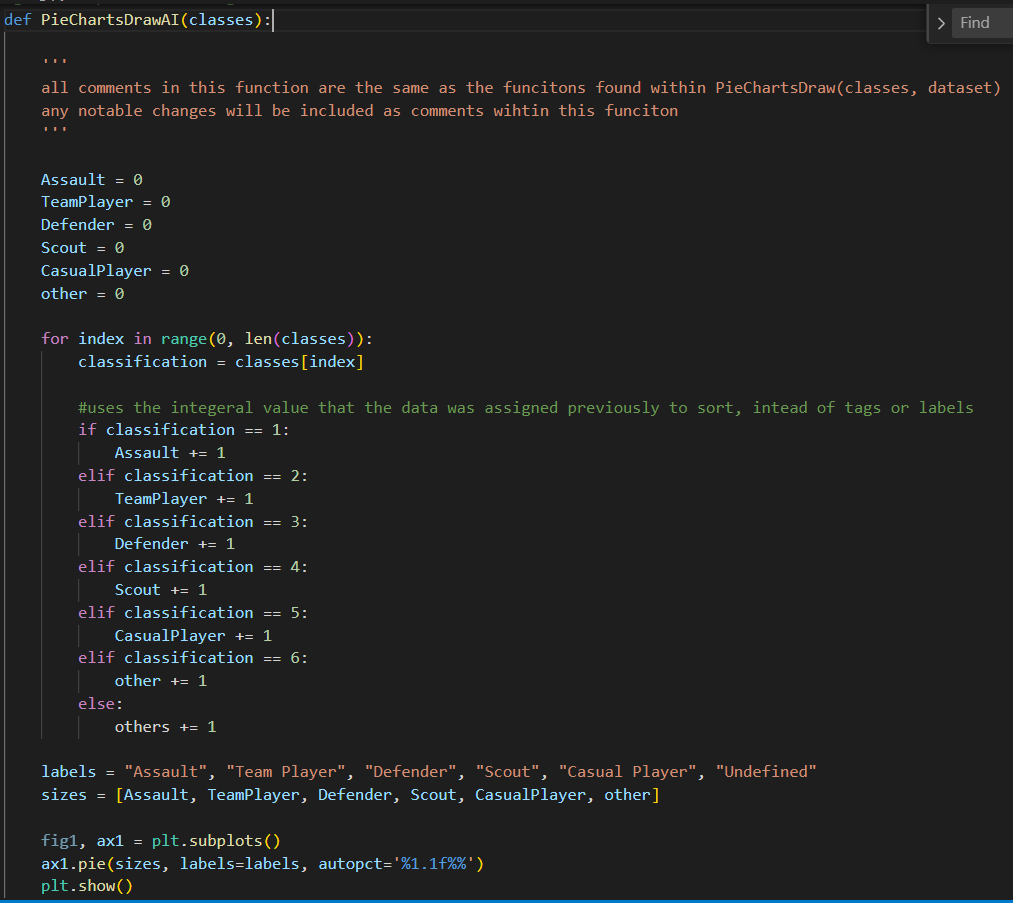
The final stage of the MLAI’s learning phase is the UserTesting class. This isn’t fully a testing stage in its own right but is arguably important as this is where I display the final results to the user. The UserTesting contains a couple of functions; PieChartsDraw(classes, dataset), and PieChartsDrawAI(classes, dataset).

PieChartsDraw(classes, dataset) starts but initiaising the names of each of the tags/labels as integer variables. Following, each row (player) in the data set is then looped through, setting up the classifications and assigning string values to allow the class to ‘read’ and assign the labels appropriately. Whist iterating through the rows, it searches for the labels and increments the appropriate class variable. Finally, the function defines the labels again and creates a list containing all of the class variables that were defined above. Once this data has been set up accordingly, the function plots and displays the pie chart with the finalised classifications.

A screen shot of a computer

Description automatically generated

The parallel function, PieChartsDrawAI(classes), does the exact same thing, just in a slightly different manner. Instead of using string checks within the classification rules/sorting, because the AI version does not use a dataset and uses predefined data, it simply scans through the data for integers (from the previous enumerate) and sorts with that instead.



The main function handles the overall application loop. Handling the calling of each class where appropriate as well as managing user inputs and overall maintenance.

# Results & Conclusions

In conclusion, the MLAI was a success. I feel that it has been structured well and is fairly well optimised. The time taken for the MLAI is, on average, less than 10 seconds in duration. Furthermore, the accuracy consistently rests at 100%. The pie charts (plotting and generation) are included in this time, excluding application pause time where it waits for user input (for example, inputs or closing of pie charts when theyre no longer needed.) The pie charts are always accurate and easy to read too. Labels are all correct and consistent with data accuracy being flawless. Overall, the model shows a successful MLAI model that can successfully manage, split, prepare, analyse, and test data of 3732 rows. This MLAI would be easy to adapt for differing data sets also.

## Results for faction: Horde:

A screenshot of a computer screen

Description automatically generatedManual plot:

A screenshot of a computer screen

Description automatically generatedAI plot:

A screenshot of a computer screen

Description automatically generatedAccuracy report:

## Results for faction: Alliance:

A screenshot of a computer screen

Description automatically generatedManual plot:

A screenshot of a computer screen

Description automatically generatedAI plot:

A screenshot of a computer screen

Description automatically generatedAccuracy report: