COMP704 – Machine Learning Worksheet WK 2

Regression Workshop

Introduction
In the lecture this week, we looked at regression-based learning to 'teach' sine() to an algorithm. In this workshop, we will explore the sine() learning solution in more detail, experiment with regression learning for house prices and consider how regression can be applied to our Breakout example.

Build the sine() example
The lecture contains all the component parts for the sine learning example, bring these parts together into a Python project, bearing in mind that the project will need to reference pandas and sklearn in order to execute correctly.

Sine() learning
The useful learning example uses the Gradient Boosting Regressor as its learning algorithm. This model has a lot of parameters that can be tweaked, experiment with them to see what impact they have on:

1. Time taken to train
2. Set Mean Absolute error
3. Overall quality of sine values produced

In addition, think about how the size of data generated and training / testing data ratios can impact the 3 qualities above.

House Price Regression example
This zip file contains a worked example from Adam Geitgey's value estimation course. Unzip the files and get the resulting code to work. It's written with a slightly order version of pandas, so there are a couple of issues to fix.

What values does Geitgey get for the three qualities mentioned above, are these reasonable? Experiment with his training parameters to see if you have a positive or negative on his performance.

Breakout Regression
As I mentioned in the lecture, we could train breakout using regression *if* we treat the bat as the desired output (Y) as something that has a range from 0 to screen_width, rather than button presses.

To do this, we need to think about what a line of input data (x) will look like in order to generate our desired output. This may be something as simple as ball position, however, we won't know until we can capture, process and present data to a learning algorithm.

To capture enough data to train our regression algorithm, we need to automate playing the game. This leads to two questions:

1. How can we make AI play the game?
2. What data should be capture?
3. How do we capture the data?

Making AI play the game
The player has a fairly easy role in Breakout, they just need to go to where the ball is going to be when it is at the player's y position on the screen. We can solve this problem with maths, but it's far easier to use the game to solve it for us. If we refactor the game so that we can make a new copy of the game

from the current version of the game, we can use that version to forward project what will happen to the ball over the next few frames (say, 1 second) and move the player to the right X position. If the difference in height between the player and the ball is large, the player may not need to do anything.

Simply refactor the breakout code to work like this.

What data should be capture

From part 1, we could capture all of the frames the game produces, but we know from Mariflow, this is wasteful, so we can look to just capture every nth frame. However, we want to capture frames that result in success; therefore, we should look to buffer frame data until the ball hits the player y-position. If the player hits the ball, it's a successful rally and we can use that data for training, otherwise we just delete it.

How do we capture the data?

Realistically, we can use any approach to capture data, however, saving it remotely might be the most effective approach as we can then run multiple breakout AI apps to collect lots of data. The http.zip example shows how to set up an HTTP server and corresponding client and form the basis of this approach.