System Development Analysis

Rock paper Scissors

Introduction

For my NEA controlled-assessment I would like to propose a solution to a problem in rock paper scissors; to create a rock paper scissors AI. The aim of the solution is to limit the randomness of the game but also involve more skill, making rock paper scissors challenging and fun. As simple and basic the game is; incorporating artificial intelligence will be no easy task therefore I will take this opportunity to develop my Python programming skills even further.

Rather than the problem being related to a third-party, I set this problem to challenge myself and my abilities in preparation for the computer science exams in June.

The application of artificial intelligence will require the use of multiple algorithms in order to simulate machine learning. I will research machine learning algorithms to aid the prediction of data. My NEA will be entirely based on data analysis and prediction to generate realistic outcomes and actions to counter human players.

How I will achieve the completion of this project

1. Start System Development Analysis
2. Research the machine learning algorithms
3. Start System Design section
4. Create a functioning rock paper scissors game
5. Integrate an AI system that will be able to recognise patterns and select moves to counterplay patterns
6. Start System Testing section
7. Start the program evaluation
8. Add further implementations to the problem [increase aglorithm's complexity]

Constant testing and debugging

Machine Learning

As much as we humans may tend to think that we can replicate randomness, various studies[1] have shown we cannot, at least not on our own. When playing a rock paper scissors game, 'random moves' are generally strings of a set of moves that we cut into pieces and play mix and match. This can be exploited using machine learning.

Machine learning can be separated into three types:

- Supervised
- Unsupervised
- Reinforcement

Supervised Machine Learning

[1] (Rath, Gustave J. "Randomization by Humans." *The American Journal of Psychology* 79, no. 1 (1966): 97-103. doi:10.2307/1420712.)

Machine learning algorithms falling into this category have known input and output data which are used to build a *classifier*[2]. Training data is used to develop an algorithm that matches input with known output. This type of machine learning is for classifying data. The algorithm created sorts input data and arranges them to the correct output for a given set of outputs – for example classifying where a certain creature classifies as a reptile or mammal using attributes such as legs, cold/hot blooded etc.

Supervised machine learning is suitable for my project. The reason being that an algorithm and classifier is based on the input data and output data – both of which I know. I will need to create an algorithm that will classify data into sequences and predict an output using the sequence within the pattern. Many pattern finding algorithms fall into the *supervised machine learning* category.

Examples of supervised machine learning include K-Means Nearest Neighbour (K-NN), Logistic Regression and Decision Tree.

Unsupervised Machine Learning

Similar to supervised machine learning; algorithms within this category have an input but an unknown output. The objective would be to sort data into groups depending on its relation to other data points. One other difference is that training data is not used, reason being that the output is unknown. The struggle with the implementation of unsupervised machine learning algorithms into my program would to organise the data in such a way that allows for the classification of data. Algorithms such as K-Means Clustering use data plotted into a graph and segments groups of data into categories depending on their relationships.

Reinforcement Learning

Reinforcement learning approaches machine learning with a different perspective – all reinforcement learning algorithms have a reward system implemented. Algorithms within this learning system create an environment that continuously alters itself to maximise on the reward it receives; when certain objectives are meet the system rewards itself and if certain criteria are unfulfilled it punishes itself. It is this continues improvement that the algorithm trains itself to produce the best decisions possible in which case yields the highest reward score.

Reinforcement learning is also suited to my program – actually, I would think that reinforcement-learning algorithms such as Artificial Neural Networking (ANN) would be the best approach to solution. A case study[3] I have used from New York Time had implemented this to their rock paper scissors program. However, I feel as though ANN would be too complex for the time scale I have to complete this project.

Objectives

[2] How the data is grouped

[3] Rock-Paper-Scissors: You vs. the Computer, The New York Times, http://www.nytimes.com/interactive/science/rock-paper-scissors.html

- The machine learning algorithm must be able to detect patterns
- The program must be able to adapt to the user
- The patterns detected must be processed to produce a reasonable predictions
- The machine learning algorithm must be able to detect when the user is spamming the same move
- The machine-learning algorithm must be able to differentiate patterns up to four numbers.
- The program must be versatile and can adapt to different inputs
- The program must be user-friendly

My choice of machine learning algorithm

I have decided that rather than using someone else's machine-learning algorithm for sequence prediction, I could create one myself based on examples. Whether the algorithm I have created is effective or not will become a topic for my evaluation. Ideally, working with neural networks would be the best solution however, I feel as though I will not be able to create a sufficient working solution to my problem.

The machine-learning algorithm will be split into three main components:

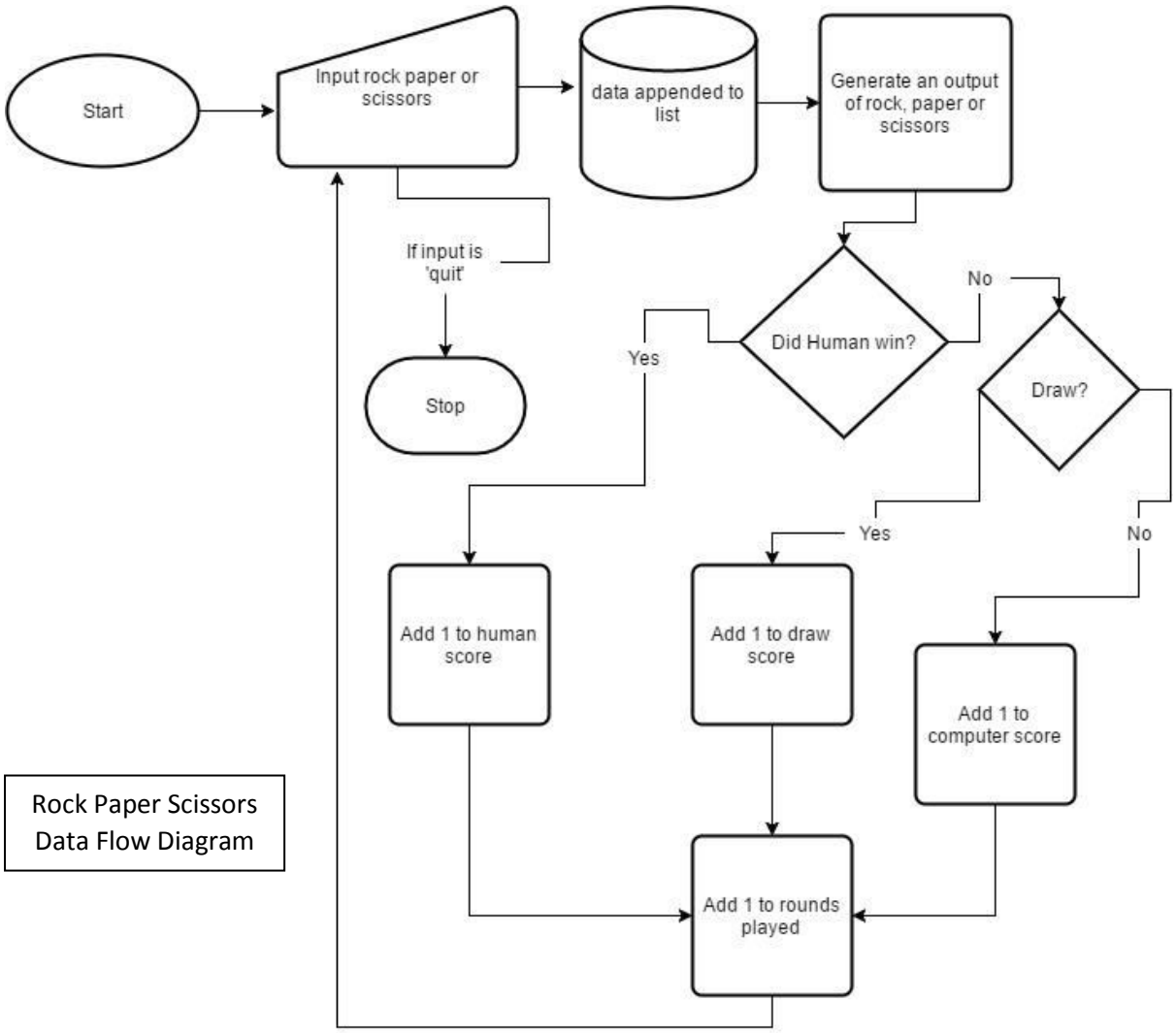
- Pattern finding
- Pattern frequency
- Prediction

The way I am thinking of designing my algorithm is that given a data set, it will first arrange the data set into patterns, and then arrange the patterns into list with the number of matches. The pattern with the highest frequency of matches will become the most likely to reoccur; so the third step in predicting the next move will finding the last data point within the pattern and using the next data point as the prediction.

Documented Design

The Game

The main function of my program is that it is a rock paper scissors game. I will need to build a simple rock paper scissors game – one with a randomly generated outcome (to simulate the computer's moves) to the user's input. This will not be hard to implement. I will be trying to use modular programming to incorporate the artificial intelligence; I will be splitting my code into two modules, one that serves as a game and the other with the sole purpose of simulating machine learning. I will also try to refrain from using libraries other than the much-needed in-built random library within python.



Rock Paper Scissors Data Flow Diagram

Data Storage

The types of data I will be using will be both integers and strings. However, the code will be working with the integer equivalent of the moves rock, paper and scissors; being 1, 2 and 3 respectively. I will use a dictionary to convert the strings into integers and vice versa. The data will be stored as integers and the type of data storage I will be using will be a lists. This is because I can extensively manipulate lists more so than for example circular queues – not only this the data set I will be working with will be relatively small. There will be little need for optimised data storage.

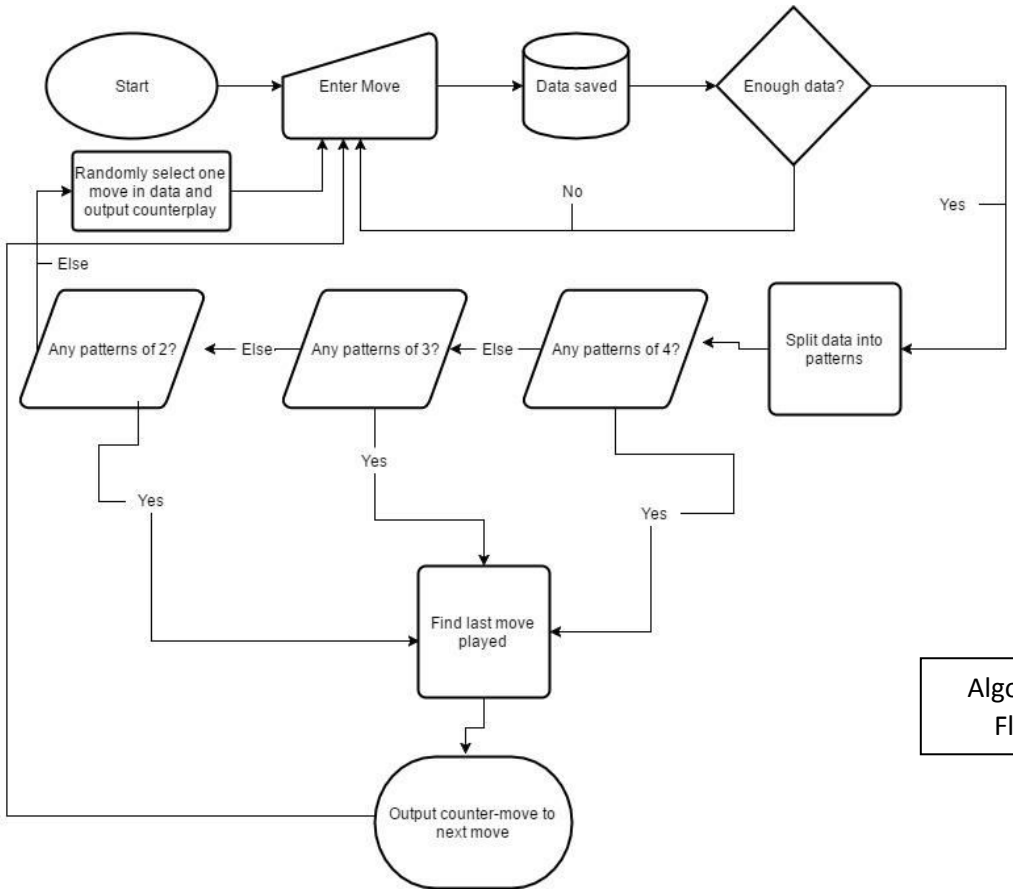
The key component within my machine-learning algorithm is the way it finds patterns. The approach I am thinking will incorporate the user of multiple pointers and the comparison between each data point in the list. I can see this being conducted with the use of for loop in python.

Creating an algorithm

I thought of challenging myself to create an algorithm that replicates machine learning. Because I know the input and out of the model, I can create an algorithm that sorts data into patterns that can also create predictions accordingly – this is a form of supervised machine learning.

Sequence sorting and prediction

A general description of the algorithm I have created would be that; data is first collected from the user and when the data is large enough, the algorithm sorts the data into a sequence of patterns. These sets of patterns are listed in priority order from their length – longest being the highest priority. The longest reoccurring pattern will become the sequence that the algorithm will use as a basis for its prediction. The last move will be located within the sequence and the next will become the prediction.



Algorithm Data Flow Chart

Why is it that data of the longest length have the highest priority?

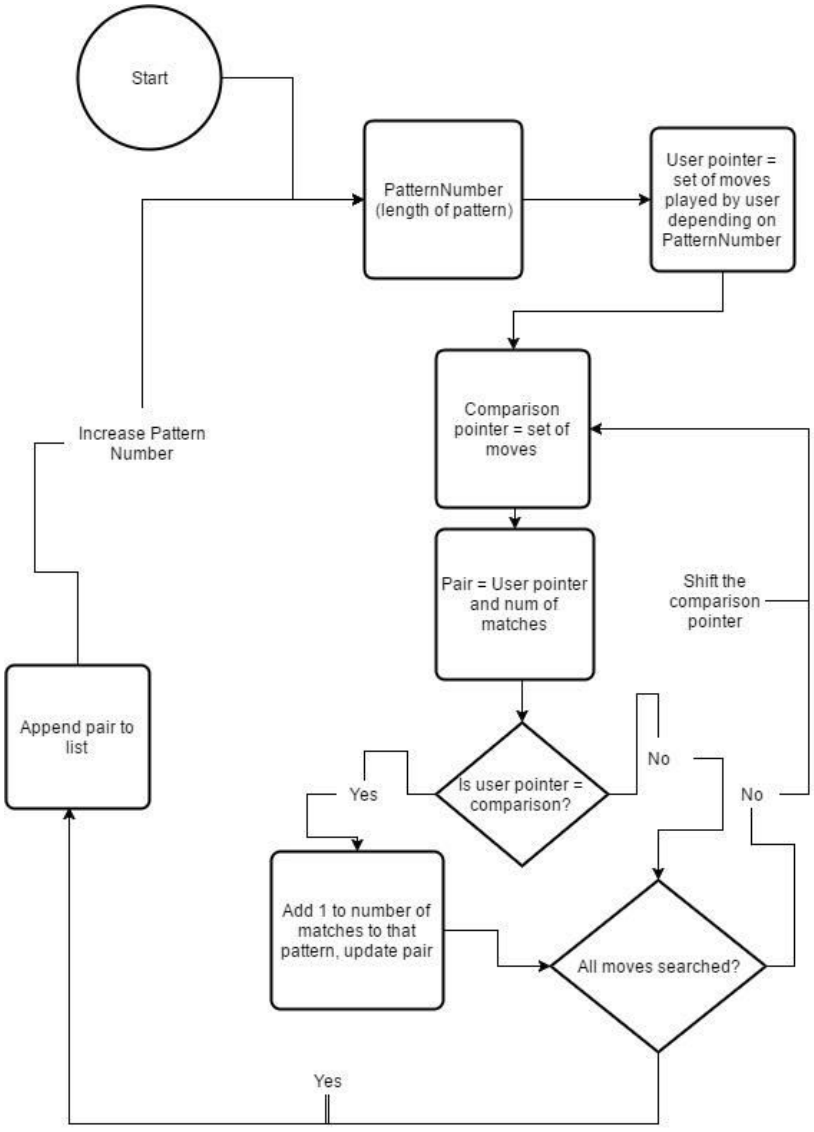
This is because patterns arranged into large sets are more predictable and accurate solutions than smaller patterns; not only this but usually larger patterns are formed from small and more frequent reoccurring sequences – this effect is more prevalent in data with little variation. For example if you have the dataset – 1,2,3,1,2,3,1,2,3 and you try to predict the next integer you would realise that although 2,3 occurs 3 times, you can predict the next value with the pattern 1,2,3,1 which only occurs twice. This is what my algorithm does; it finds the largest pattern possible and finds the last played move within that pattern, the next move being the most likely that the user will choose.

One potential problem in which could arise is that as the rounds progress; earlier rounds may negatively influence the algorithm. If the user plays a certain set of moves at the beginning, the algorithm may think that the user will play the same set of moves despite being played twice during earlier rounds – this will skew predications. Instead of finding all patterns within the data, I will try to design the algorithm to find patterns in relation to the moves recently played.

I used multiple classmates and asked them to play a game of rock paper scissors. For each classmate after ten rounds, I asked them to recall the moves they had played. Four out of the five peers were able to accurately calculate six rounds, one only managing four because he was throwing random moves.

This trial gave me into an insight to not only how much data should be sufficient for the algorithm to start with; but also what the maximum length of pattern my algorithm should find. I have decided that the algorithm will become active after five rounds of rock paper scissors and that the maximum length pattern that will be searched for will be four – this however could be subjected to change.

Here is the design to my pattern-finding algorithm.



Pattern Finding Algorithm Flow Chart

Pattern List

My algorithm will create a list with four items; patterns found using the last four, three and two of last played moves as well as the most frequent move played. For example if the list of moves played was 1,2,3,1,2,3,1,2,3 – the list of patterns will consist of the patterns 3,2,1,3 as its four pattern item and 3,2,1 as its three pattern item and so on. There will be a max number of four items in the last as my design currently only supports patterns of four combinations – this can be increased later on.

If the frequency of patterns have multiple matches, the pattern that is the longest will have the highest priority. This can be implemented using a nested if statement that checks the pattern number in descending order.

Predictions in no patterns found

When no patterns are detected from the algorithm, I will need to simulated weighted probability to guess the next move. This function is not a part python's default random library and can be achieved using a library called PyNum. It has a method called enumerate which calculates the probability of an item in a list given the number of times it has occurred. Using this library will become essential to my machine-learning algorithm.

Prediction in patterns of two or more

Patterns that occur in arrangements of two or more will have the algorithm calculate the next move according to that pattern. If the algorithm finds that the most occurring pattern is 1,2,3 then the prediction for the next move will simply be the next move after the move that has been played; in which case a prediction of 2 will be calculated if the last played move was a rock. This does pose some problems however, if the last played move is the last move within a pattern, simply choosing the move after will pose problems as it goes over the range of numbers available in the list. I will need to problem a way in resolving this by adding a statement that selects the first number in a pattern as a prediction if the last played move is the last move in the pattern.

Added human thinking

There some techniques in rock paper scissors that almost all humans use; these include bluffs, double bluffs and some logical thinking that can be hard coded to a computer given certain situations. For example, if the user inputs rock multiple times and then switches to a paper; the AI may think that the next move will be rock given the frequency of rock used as a move. A human opponent however may be more likely to throw a scissors or repeat the paper instead. These will provide fewer implications as the rounds progress because the AI should be able to detect these patterns but during early rounds, the AI is more vulnerable due to its lack of data. Also, smart thinking could also be implemented to the AI for situations where the AI will purposely throwing the losing move to the previous move in hope to reduce the probability of losing the round to 33% but drawing and winning to 66% should the player choose not to repeat the same move.

Technical Solution

Main.py

```
## random number generating library
import random
import aiaiai ## imports the AI module

          ##############################
          ## Rock Paper Scissors game ##
          ##############################
          ##  Artifical Intelligence  ##
          ##############################

moves = {1:'rock', 2:'paper', 3:'scissors'} ## dictionary to help convert numerals into moves

data = []
Round = 1
HumanScore = 0
ComputerScore = 0
DrawScore = 0

def Human(HUchoice): ## function to receieve and convert move into a numeral
  while True:
    if HUchoice == 'rock':
      Choice = 1
      return Choice
    elif HUchoice == 'paper':
      Choice = 2
      return Choice
    elif HUchoice == 'scissors':
      Choice = 3
      return Choice
    elif HUchoice == 'quit' or HUchoice == 'exit':
      return HUchoice
    else: ## error checking, loops the function to the beginning if either 'rock', 'paper' or 'scissors' is not inputted
      print ("Please enter a valid move")
      print('\n')
      HUchoice = input("Please enter rock, paper or scissors:")
      print('\n')

def Computer(Choice, Round, data): ## function to compare and produce a decisive ouput on winner
  if Round < 5:
    AIchoice = random.randint(1,3) ## generates computer's move randomly
  else:
    AIchoice = aiaiai.WinningChoice(data) ## this triggers the machine learning algorithm
  if Choice == 1  and AIchoice == 3: ## anonomlous comparison
    return ('Human', AIchoice)
  elif Choice == 3 and AIchoice == 1: ## anonomlous comparison
    return ('Computer', AIchoice)
  elif Choice > AIchoice:
    return ('Human', AIchoice)
  elif Choice < AIchoice: ## returns who won and the AI's move
    return ('Computer', AIchoice)
  elif Choice == AIchoice:
    return ('Draw', AIchoice)

## main program
print ("Welcome to the Rock Paper Scissors! If you would like to quit please enter 'quit'")
while True:
  HUchoice = input("Please enter 'rock', 'paper' or 'scissors':")
  print ("\n")
  if HUchoice == 'quit' or HUchoice == 'exit':
    print("Exiting program")
    break
else:
  Choice = Human(HUchoice) ## function returned human choice and sets as variable outside function
  if Choice == 'quit' or Choice == 'exit':
    print("Exiting program")
    break
  CompOutput = Computer(Choice, Round, data) ## variables returned as tuples from fucntion Computer()
  data.append(Choice)
  print("Round:",Round)
  print ("You threw", moves[Choice]) ## using dictionary 'moves' to convert choice into appropriate move
  print ("Computer threw", moves[CompOutput[1]]) ## Tuple in CompOutput has 2 variables, CompOutput[1] ouputs the second variable AIchoice
  if (CompOutput[0] == 'Human') or (CompOutput[0] == 'Computer'):
    print ("%s won" % (CompOutput[0]))
  else:
    print ("The outcome was a",CompOutput[0]) ## CompOutput[0] returns the first variable from the tuple
if CompOutput[0] == 'Human': ## score system
  HumanScore += 1 ## adds 1 to the initial value
elif CompOutput[0] == 'Computer':
  ComputerScore += 1
else:
  DrawScore += 1
print("Player Score: %s\nComputer Score: %s\nDraw: %s\n"  % (HumanScore, ComputerScore, DrawScore)) ## outputs the score
Round += 1
```

aiaiai.py

```
import random

patterns = [] ## sets the list
matches = 0 ## sets variable matches to 0
PatNum = 0 ## sets patnum to 0
i = None ## changes data withim variable i to none
j = None ## changes data withim variable j to none

def MaxMatches(patterns): ## this function finds the most frequent pattern
  position = 0
  Max = 0
  MaxM = 0
  TheList = [] ## list of checked patterns
  for i in range(3,-1,-1): ## decending order from
    if not TheList:
      if patterns[i][1] > Max: ## max matches found
        MaxM = patterns[i][1] ## update matches
        position = i ## update position
        TheList.append(patterns[i]) ## append pattern to list of checked patterns
  return patterns[position][0], MaxM ## once all patterns have been checked

def MultiplePattern(data, NewMatch, UserPointer):
  counter = 0 ## counter
  for i in NewMatch: ## loop desgined to find the next value within the pattern
    if i == UserPointer and NewMatch[0] != UserPointer: ## as long as the loop has not reached the last value in pattern
      guess = NewMatch[counter-1] ## the predicted move will be the value before the last thrown move
      exit ## exit loop
    else:
      guess = NewMatch[len(NewMatch)-1] ## if last thrown move is last in the pattern, next move is first in the pattern
      exit
    counter += 1
  return guess

def DoublePattern(data, NewMatch, UserPointer):
  guess = NewMatch[0] ## since values are reversed, last thrown move is second and next move is first
  return guess ## returns first value as prediction

def SinglePattern(data, UserPointer):
  guess = random.choice(data) ## weighted probability simulation
  return guess

def SearchMatching(data, NewMatch, PatNum):
  UserPointer = (data[(len(data))-1]) ## finds last thrown move
  if PatNum > 2: ## patterns of varying lengths follow different functions
    NextMove = MultiplePattern(data, NewMatch, UserPointer)
  elif PatNum  == 2:
    NextMove = DoublePattern(data, NewMatch, UserPointer)
  else:
    NextMove = SinglePattern(data, UserPointer)
  return NextMove

def NextChoice(patterns, data):
  Maximum = MaxMatches(patterns) ## receives pattern of highest frequency
  if patterns[0] == Maximum: ## finds the pattern matching with the highest frequency
    PatNum = 1
    NewMatch = (patterns[0][0]) ## selects pattern instead of pattern and matches (tuple value)
    Move = SearchMatching(data, NewMatch, PatNum) ## move calculated
  elif patterns[1] == Maximum:
    PatNum = 2
    NewMatch = (patterns[1][0][::-1]) ## reverses the order of numbers in the pattern
    Move = SearchMatching(data, NewMatch, PatNum)
```

```
  elif patterns[2] == Maximum:
    PatNum = 3
    NewMatch = (patterns[2][0][::-1])
    Move = SearchMatching(data, NewMatch, PatNum)
  elif patterns[3] == Maximum:
    PatNum = 4
    NewMatch = (patterns[3][0][::-1])
    Move = SearchMatching(data, NewMatch, PatNum)
  return Move

def SelectingPointer(data,PatNum, i): ## function that selects pointer depending on pattern number
  if PatNum == 4:
    UsePointer = data[len(data)-1],data[len(data)-2],data[len(data)-3],data[len(data)-4]
  elif PatNum == 3:
    UsePointer = data[len(data)-1],data[len(data)-2],data[len(data)-3]
  elif PatNum == 2:
    UsePointer = data[len(data)-1],data[len(data)-2]
  elif PatNum == 1:
    UsePointer = data[len(data)-1] ## different scales so -1 needed to match scales accordingly
  if i == None:
    Comparison = None
  else:
    if PatNum == 4:
      Comparison = data[i],data[i-1],data[i-2],data[i-3] ## comparison pointer changes depending on loop number
    elif PatNum == 3:
      Comparison = data[i],data[i-1],data[i-2]
    elif PatNum == 2:
      Comparison = data[i],data[i-1]
    elif PatNum == 1:
      Comparison = data[i]
  return UsePointer,Comparison

def GetPatterns(data, matches, PatNum):
  pair = 0,0 ## defines pair as a tuple
  for i in range((len(data)-2),(PatNum)-2,-1): ## loops from decending order from max length of data to zero
    UsePointer = SelectingPointer(data, PatNum, i) ## userpointer corresponding to the correct pattern selected
    pair = UsePointer[0],matches ## pair redefined
    if UsePointer[0] == UsePointer[1]: ## if match found
      matches += 1 ## adds one to matches
      pair = UsePointer[0],matches ## pair updated
  patterns.append(pair) ## once loop finishes (all data searched), pair is appended to list of patterns
  matches = 0 ## matches reset to zero

def LearnChoice(PatNum, data): ## variables PatNum and data are passed on from previous function
  for PatNum in range(1,5): ## finds patterns of 1,2,3, and 4
    GetPatterns(data,matches,PatNum) ## function finds patterns
  AIMove = NextChoice(patterns, data) ## function finds prediction from patterns found
  del patterns[:] ## deletes the list of patterns - this is done at the end of each round
  return AIMove

def WinningChoice(datalocal): ## the data is sent form the main module to machine learning module
  data = datalocal ## defines list of moves from the received data
  Move = LearnChoice(PatNum, data) ## LearnChoice triggers all the other functions
  if Move == 3: ## these are the predictions
    Choice = 1 ## beating the predictions
  elif Move == 1:
    Choice = 2
  elif Move == 2:
    Choice = 3
  return Choice ## returns the move to be played by the AI
```

aiaiai.py

The base game

You cannot compared the strings rock, paper and scissors together, especially in python. This is because they are words and there is nothing to compare. In order to simulate the rock paper scissors hierarchy, I could use multiple if statements that trigger when certain conditions are met; for example if the user plays rock and the computer, plays scissors then the outcome would be that the human wins. This process will require me to label each scenario but this is very inefficient. The solution I came with makes use of pythons ability to compare numbers; I stored the strings the user inputted as integers that indicate to each move; rock, paper and scissors being 1,2 and 3 respectively. Although I still had to use multiple if statements this still meant I achieved the same effect with fewer lines of code. My method was that three is greater than two and two is greater than one, in these situations the higher number won the round – likewise scissors beats paper and paper beats rock. I had to predefine other conditions where one and three were values being compared but it was only this and the drawing rounds.

The majority of the code in the base game is composed of entirely if, else and print statements – the game is simple and does not require an increase in any complexity. I feel as though simpler code will not only be easier to debug but tend to work more efficiently as well.

Moves played

When the moves are inputted to the game, they are converted into integers making data processing a much easier process. However when I output these values, I have to output the string equivalent to these values; this is where the dictionary I had created at the beginning of my code became effective. With the key becoming the integer counterpart of the string, and the string, the value, the dictionary allowed for easy conversion between integer and string. I could do this with the simple code moves[integer]. You could argue that a list could have served the exact same purpose and it could. However, one thing to note is that a dictionary cannot be changed and in my case it does not need to be; a list on the other hand could be prone to accidental changes. If I were to change the corresponding values of rock paper scissors to something else, the dictionary would allow for a much easier change rather than a list. For a list, I will have to create a separate list within a list to store the string equivalent of integers – then there is also the mix-up between data types, this could produce unnecessary complications within the code.

Error handling

I decided not to use python's in-built function of error handing using exceptions. As good as it is, it is limited a small range of errors like ValueError or ZeroDivsionError. My code could make use of the ValueError but I will still in the end need to validate the user input. I chose to do this using a while loop. The while loop will continually play rounds of rock paper scissors until either an 'exit' or 'quit' has been inputted – in which case will stop the program. When a supposedly valid input has been entered, the function Human attempts to convert the input into a move. If it cannot then an error message is outputted asking the user to enter a valid input. The program will not progress until it detects a valid input – this ensures that whatever the reason for the code breaking later will not be due to the input.

Functions within rock paper scissors

There are two functions within my program of rock paper scissors – One defined as Human and the other define as Computer.

The reason being why the code within these functions are put in functions in the first place is that they process data and produce an output. For example, the function Human validates the input of the user and will call upon itself in recursion until a valid input has been detected. The data the function outputs corresponds to this input. Computer on the other hand simply outputs an integer in between one to three randomly until five rounds have past, after which the machine-learning algorithm starts to analyse and interpret the data. In addition, it is within this function that the comparative statements are made; it outputs who won and the computer move in the form of a tuple.

Aiaiai module

My rock paper scissors program is split into modules – main.py and aiaiai.py. Using a modular format allows for easier debugging as errors can be located easier and problems tend to be more isolated. I also found the program to be displayed more clearly and easier to alter personally. One problem I encountered was the transferring the list of data (moves) to the machine learning module. Since global variables are not cross modular, I had to pass the data as an argument, which worked just as well.

The AI module consists of many functions. The first function called is WinningChoice. In this function not only was the list of moves set as a local variable for all modules, but also oversees the entire pattern finding and prediction process. It also produces an output corresponding to the prediction produce – meaning if LearnChoice outputs a prediction of one, the AI's choice of move returned will be two.

The function LearnChoice splits data into patterns. It splits the data into patterns of four, three, two and one accordingly. This is done using a forloop which increase an increment of one each loop, running the pattern finding algorithm GetPatterns for that pattern number. After all the patterns have been found, a separate algorithm in NextChoice is used to find the prediction. All the patterns in the pattern list are then deleted to avoid a collection of unused patterns – bearing in mind that my program finds patterns according to the last four moves played which will not necessarily be the same every time. LearnChoice then returns the predicted moved.

GetPatterns is an essential part of my pattern-finding algorithm. This where the data is analysed and process for patterns. The way in which the forloop works in GetPatterns is that it loops depending on the number of total moves played by the user and the pattern number the algorithm is searching for – it goes down in descending order from the number of moves played, not the moves themselves. Each loop has a variable i corresponding to the loop number. The pattern number affects the data searched as the whole list of moves may be searched which will result in duplicates. For example, for data of length, ten and we are searching for patterns of four, the loop will only repeat six times. It will not search the last four number themselves as they are being used to compare for other patterns. The UsePointer tuple contains the pattern to be matched – this changes with pattern number. This is achieved using another function, SelectingPointer. The function also selects the comparison pointers – these changes accordingly with the loop number. The tuple pair is updated, and if the UsePointer matches, the Comparison pointer then matches is increased by one and the

pair tuple updated again. Once the whole set of moves is processed, the pattern and its matches is appended to a separate list. After which, the matches variable reset to zero. The whole process is looped for four times.

NextChoice is the main function in which a prediction is calculated – the function receives the predicted pattern and guesses the next move dependently. The most frequent pattern chosen by the function MaxMatches. The pattern's positon is calculated within the pattern list to find its pattern number. A collection of if statements then separate the pattern into the right segment; reversing the pattern in the process. Reversing the pattern at first seemed to be a good idea – the pattern was moved left to right and was easier to read. However, I feel as though it was unnecessary and added unneeded complexity to the code and actually causing some implications. I will expand on this at a later point during the testing. The pattern is then processed by SearchMatching to predict the next move in the series, passing on the data, pattern and pattern number as arguments. The value outputted would be returned as the prediction.

SearchMatching runs the prediction function according to the pattern number. For single patterns, the guess outputted would be a random number within the moves played. This is to simulate weighted probability. Because no patterns have been detected, the chance of any move being played next may be depended on the moves already played. For example if rock has been played four times but scissors and paper, three times; the most likely scenario could be that rock will be played again. Strictly speaking this is quite untrue actually – and quite the opposite in some scenarios. For example if the user plays rock and paper, a likely scenario would be that the next option would be scissors. For patterns coming in arrangements of doubles, the guess move would simply be the move that is not the last played move – so the other move. For patterns in arrangements of three and four, I have designed an algorithm, which finds the position of the last played move. Moreover, because the pattern is reversed, the guess would therefore be the move before the last played move within the pattern. I did this by using a for loop to search every integer within the pattern. If the integer matched, the last played move and the integer is not the first number within the pattern then the guess would simply be the previous digit. I used a counter to keep track of the position of a virtual pointer – this counter will help in returning the positon of the guess since the for loop this time uses the integers in the pattern itself as its i values.

The function MaxMatches analyses the patterns in the pattern list to find the pattern with the most matches. Since there is no in-built function in python that can search a list to return an item that has met a certain criteria, I decided to simulate this using a for loop. The loop makes use of the integers three to negative one – reason being positions in lists start at zero rather than one. The function makes use of a separate list to keep track of patterns searched; multiple other variables are used, like MaxM and Max to indicate the highest number of matches and number of matches respectively. For each pattern, the loop compares the matches' number to the highest match number. If the loop finds a new highest match number, the patterns position and match, number is saved as a variable to be compared to other match numbers. Once the loop ends, the last saved pattern position and number of matches is outputted.

Testing Evidence

```
Round: 5
You threw rock
Computer threw paper
Computer won
Player Score: 1
Computer Score: 2
Draw: 2

Please enter 'rock', 'paper' or 'scissors':rock


Round: 6
You threw rock
Computer threw paper
Computer won
Player Score: 1
Computer Score: 3
Draw: 2

Please enter 'rock', 'paper' or 'scissors':rock


Round: 7
You threw rock
Computer threw paper
Computer won
Player Score: 1
Computer Score: 4
Draw: 2

Please enter 'rock', 'paper' or 'scissors':rock


Round: 8
You threw rock
Computer threw paper
Computer won
Player Score: 1
Computer Score: 5
Draw: 2

Please enter 'rock', 'paper' or 'scissors':rock


Round: 9
You threw rock
Computer threw paper
Computer won
Player Score: 1
Computer Score: 6
Draw: 2
```

```
Round: 5
You threw paper
Computer threw scissors
Computer won
Player Score: 1
Computer Score: 2
Draw: 2

Please enter 'rock', 'paper' or 'scissors':paper


Round: 6
You threw paper
Computer threw scissors
Computer won
Player Score: 1
Computer Score: 3
Draw: 2

Please enter 'rock', 'paper' or 'scissors':paper


Round: 7
You threw paper
Computer threw scissors
Computer won
Player Score: 1
Computer Score: 4
Draw: 2

Please enter 'rock', 'paper' or 'scissors':paper


Round: 8
You threw paper
Computer threw scissors
Computer won
Player Score: 1
Computer Score: 5
Draw: 2

Please enter 'rock', 'paper' or 'scissors':paper


Round: 9
You threw paper
Computer threw scissors
Computer won
Player Score: 1
Computer Score: 6
Draw: 2
```

```
Round: 5
You threw rock
Computer threw paper
Computer won
Player Score: 2
Computer Score: 1
Draw: 2

Please enter 'rock', 'paper' or 'scissors':paper


Round: 6
You threw paper
Computer threw paper
The outcome was a Draw
Player Score: 2
Computer Score: 1
Draw: 3

Please enter 'rock', 'paper' or 'scissors':rock


Round: 7
You threw rock
Computer threw paper
Computer won
Player Score: 2
Computer Score: 2
Draw: 3

Please enter 'rock', 'paper' or 'scissors':paper


Round: 8
You threw paper
Computer threw scissors
Computer won
Player Score: 2
Computer Score: 3
Draw: 3

Please enter 'rock', 'paper' or 'scissors':rock


Round: 9
You threw rock
Computer threw paper
Computer won
Player Score: 2
Computer Score: 4
Draw: 3
```

```
Round: 5
You threw paper
Computer threw paper
The outcome was a Draw
Player Score: 3
Computer Score: 1
Draw: 1

Please enter 'rock', 'paper' or 'scissors':scissors


Round: 6
You threw scissors
Computer threw paper
Human won
Player Score: 4
Computer Score: 1
Draw: 1

Please enter 'rock', 'paper' or 'scissors':rock


Round: 7
You threw rock
Computer threw paper
Computer won
Player Score: 4
Computer Score: 2
Draw: 1

Please enter 'rock', 'paper' or 'scissors':paper


Round: 8
You threw paper
Computer threw paper
The outcome was a Draw
Player Score: 4
Computer Score: 2
Draw: 2

Please enter 'rock', 'paper' or 'scissors':scissors


Round: 9
You threw scissors
Computer threw scissors
The outcome was a Draw
Player Score: 4
Computer Score: 2
Draw: 3
```

```
Round: 66
You threw paper
Computer threw paper
The outcome was a Draw
Player Score: 26
Computer Score: 24
Draw: 16

Please enter 'rock', 'paper' or 'scissors':paper


Round: 67
You threw paper
Computer threw scissors
Computer won
Player Score: 26
Computer Score: 25
Draw: 16

Please enter 'rock', 'paper' or 'scissors':scissors


Round: 68
You threw scissors
Computer threw rock
Computer won
Player Score: 26
Computer Score: 26
Draw: 16

Please enter 'rock', 'paper' or 'scissors':rock


Round: 69
You threw rock
Computer threw paper
Computer won
Player Score: 26
Computer Score: 27
Draw: 16

Please enter 'rock', 'paper' or 'scissors':paper


Round: 70
You threw paper
Computer threw scissors
Computer won
Player Score: 26
Computer Score: 28
Draw: 16
```

```
Round: 66
You threw paper
Computer threw paper
The outcome was a Draw
Player Score: 19
Computer Score: 21
Draw: 26

Please enter 'rock', 'paper' or 'scissors':scissors


Round: 67
You threw scissors
Computer threw scissors
The outcome was a Draw
Player Score: 19
Computer Score: 21
Draw: 27

Please enter 'rock', 'paper' or 'scissors':rock


Round: 68
You threw rock
Computer threw rock
The outcome was a Draw
Player Score: 19
Computer Score: 21
Draw: 28

Please enter 'rock', 'paper' or 'scissors':paper


Round: 69
You threw paper
Computer threw paper
The outcome was a Draw
Player Score: 19
Computer Score: 21
Draw: 29

Please enter 'rock', 'paper' or 'scissors':paper


Round: 70
You threw paper
Computer threw scissors
Computer won
Player Score: 19
Computer Score: 22
Draw: 29
```

```
Round: 66
You threw paper
Computer threw paper
The outcome was a Draw
Player Score: 22
Computer Score: 23
Draw: 21

Please enter 'rock', 'paper' or 'scissors':scissors


Round: 67
You threw scissors
Computer threw scissors
The outcome was a Draw
Player Score: 22
Computer Score: 23
Draw: 22

Please enter 'rock', 'paper' or 'scissors':rock


Round: 68
You threw rock
Computer threw rock
The outcome was a Draw
Player Score: 22
Computer Score: 23
Draw: 23

Please enter 'rock', 'paper' or 'scissors':paper


Round: 69
You threw paper
Computer threw paper
The outcome was a Draw
Player Score: 22
Computer Score: 23
Draw: 24

Please enter 'rock', 'paper' or 'scissors':apepr


Please enter a valid move


Please enter rock, paper or scissors:paper


Round: 70
You threw paper
Computer threw scissors
Computer won
Player Score: 22
Computer Score: 24
Draw: 24
```

```
Round: 5
You threw rock
Computer threw paper
Computer won
Player Score: 1
Computer Score: 2
Draw: 2

Please enter 'rock', 'paper' or 'scissors':rock


Round: 6
You threw rock
Computer threw paper
Computer won
Player Score: 1
Computer Score: 3
Draw: 2

Please enter 'rock', 'paper' or 'scissors':scissors


Round: 7
You threw scissors
Computer threw paper
Human won
Player Score: 2
Computer Score: 3
Draw: 2

Please enter 'rock', 'paper' or 'scissors':paper


Round: 8
You threw paper
Computer threw paper
The outcome was a Draw
Player Score: 2
Computer Score: 3
Draw: 3

Please enter 'rock', 'paper' or 'scissors':rock


Round: 9
You threw rock
Computer threw paper
Computer won
Player Score: 2
Computer Score: 4
Draw: 3
```

```
Round: 5
You threw scissors
Computer threw scissors
The outcome was a Draw
Player Score: 1
Computer Score: 1
Draw: 3

Please enter 'rock', 'paper' or 'scissors':scissors


Round: 6
You threw scissors
Computer threw paper
Human won
Player Score: 2
Computer Score: 1
Draw: 3

Please enter 'rock', 'paper' or 'scissors':rock


Round: 7
You threw rock
Computer threw scissors
Human won
Player Score: 3
Computer Score: 1
Draw: 3

Please enter 'rock', 'paper' or 'scissors':rock


Round: 8
You threw rock
Computer threw rock
The outcome was a Draw
Player Score: 3
Computer Score: 1
Draw: 4

Please enter 'rock', 'paper' or 'scissors':paper


Round: 9
You threw paper
Computer threw paper
The outcome was a Draw
Player Score: 3
Computer Score: 1
Draw: 5
```

```
Round: 5
You threw paper
Computer threw scissors
Computer won
Player Score: 2
Computer Score: 2
Draw: 1

Please enter 'rock', 'paper' or 'scissors':scisors


Please enter a valid move


Please enter rock, paper or scissors:paper


Round: 6
You threw paper
Computer threw scissors
Computer won
Player Score: 2
Computer Score: 3
Draw: 1

Please enter 'rock', 'paper' or 'scissors':rock


Round: 7
You threw rock
Computer threw scissors
Human won
Player Score: 3
Computer Score: 3
Draw: 1

Please enter 'rock', 'paper' or 'scissors':paper


Round: 8
You threw paper
Computer threw scissors
Computer won
Player Score: 3
Computer Score: 4
Draw: 1

Please enter 'rock', 'paper' or 'scissors':scissors


Round: 9
You threw scissors
Computer threw scissors
The outcome was a Draw
Player Score: 3
Computer Score: 4
Draw: 2
```

```
Welcome to the Rock Paper Scissors! If you would like to quit please enter 'quit'
Please enter 'rock', 'paper' or 'scissors':ROCK


Round: 1
You threw rock
Computer threw scissors
Human won
Player Score: 1
Computer Score: 0
Draw: 0

Please enter 'rock', 'paper' or 'scissors':PAPER


Round: 2
You threw paper
Computer threw paper
The outcome was a Draw
Player Score: 1
Computer Score: 0
Draw: 1

Please enter 'rock', 'paper' or 'scissors':SCISSORS


Round: 3
You threw scissors
Computer threw paper
Human won
Player Score: 2
Computer Score: 0
Draw: 1

Please enter 'rock', 'paper' or 'scissors':RoCk


Round: 4
You threw rock
Computer threw paper
Computer won
Player Score: 2
Computer Score: 1
Draw: 1

Please enter 'rock', 'paper' or 'scissors':quit


Exiting program
```

```
Welcome to the Rock Paper Scissors! If you would like to quit please enter 'quit'
Please enter 'rock', 'paper' or 'scissors'://sa/sf


Please enter a valid move


Please enter rock, paper or scissors:asff


Please enter a valid move


Please enter rock, paper or scissors:123


Please enter a valid move


Please enter rock, paper or scissors:12


Please enter a valid move


Please enter rock, paper or scissors:2


Please enter a valid move


Please enter rock, paper or scissors:papr


Please enter a valid move


Please enter rock, paper or scissors:rock


Round: 1
You threw rock
Computer threw scissors
Human won
Player Score: 1
Computer Score: 0
Draw: 0

Please enter 'rock', 'paper' or 'scissors':scisor


Please enter a valid move
```

Evaluation

References

Draw. n.d. *Dataflow Chart Creator.* Accessed April 12, 2017. https://www.draw.io/.

Morrison, Karl. n.d. *Supervised learning, unsupervised learning and reinforcement learning: Workflow basics.* Accessed March 17, 2017. http://stats.stackexchange.com/questions/144154/supervised-learning-unsupervised-learning-and-reinforcement-learning-workflow.

Ray, Sunil. n.d. *Machine Learning Algorithms.* Accessed March 17, 2017. https://www.analyticsvidhya.com/blog/2015/08/common-machine-learning-algorithms/.

SRIVASTAVA, TAVISH. 2014. *Introduction to k-nearest neighbors : Simplified.* October 10. Accessed March 17, 2017. https://www.analyticsvidhya.com/blog/2015/08/common-machine-learning-algorithms/.

The New York Times. n.d. *Rock-Paper-Scissors: You vs. the Computer.* Accessed March 14, 2017. http://www.nytimes.com/interactive/science/rock-paper-scissors.html?_r=0.

Wikipedia. n.d. *K-Means Clustering.* Accessed March 17, 2017. https://en.wikipedia.org/wiki/K-means_clustering.

—. n.d. *K-Nearest Neighbors.* Accessed March 17, 2017. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

—. n.d. *Pattern Recognition.* Accessed March 17, 2017. https://en.wikipedia.org/wiki/Pattern_recognition.

—. n.d. *Supervised Learning.* Accessed March 17, 2017. https://en.wikipedia.org/wiki/Supervised_learning.