THEORY OF COMPUTATION AND COMPILER DESIGN PROJECT

On

Predicting Human Move in Rock, Paper and Scissors Game using Machine Learning

Group Members

NAME	REGISTRATION NUMBER
ARPIT RATHI	16BCI0065
PARMEET SINGH	16BCE0184

Under the guidance of
SHALINI L
School of Computer Science And Engineering
VIT University, Vellore

NOVEMBER 2017

TABLE OF CONTENTS

INTRODUCTION	**3**
ABSTRACT	**3-4**
LITERATURE SURVEY	**4**
STRATEGY	**4-5**
CURRENT BOTS	**5-6**
ALGORITHM	**6**
SYSTEM DESIGN AND EXPLANATION	**7-11**
STATISTICS	**12**
CONCLUSION	**12**
REFERENCES	**12**

INTRODUCTION

Rock-paper-scissors is a hand game usually played between two people, in which each player simultaneously forms one of three shapes with an outstretched hand. These shapes are "rock" (a closed fist), "paper" (a flat hand), and "scissors" (a fist with the index and middle fingers extended, forming a V). "Scissors" is identical to the two-fingered V sign (aka "victory" or "peace sign") except that it is pointed horizontally instead of being held upright in the air.

A player who decides to play rock will beat another player who has chosen scissors ("rock crushes scissors" or sometimes "blunts scissors"), but will lose to one who has played paper ("paper covers rock"); a play of paper will lose to a play of scissors ("scissors cut[s] paper"). If both players choose the same shape, the game is tied and is usually immediately replayed to break the tie.

This being a study of the human mind and the pattern in which humans think to make their next move is an interesting area for using algorithms and machine learning concepts to predict the next move and beat humans in this game using an algorithm.

Because of this many algorithms have been made in the past to simulate this issue. The most popular of which is the Iocaine Powder, which won the first International RoShambo Programming Competition and uses a heuristically designed compilation of strategies.

ABSTRACT

This project focuses on creating a bot that can play the common game of Rock Paper Scissors (RPS) better than its human opponent. The key idea is that humans tend to play in patterns rather than completely at random. Using machine learning techniques, we will train a bot to learn how we play RPS, thus giving it an advantage over us. For example, if every time we play the sequence of ‘Rock’, ‘Rock’, ‘Paper’ and ‘Rock’ our next throw is consistently ‘Scissors’, then our bot learns that and plays ‘Rock’ in the next game to beat us.So, for this problem our input is a sequence of n human plays, which we represent as an n length vector, where each element in the vector is an integer in [1,3]. ‘Rock’ is represented by 1, ‘Paper’ is represented by 2, and ‘Scissors’ is represented by 3. Given this sequence of human plays, our bot uses various classifiers to predict our n+1 th play, which is also represented by an integer as described above.Given this play, the bot chooses its winning counterpart (i.e. predicting ‘Rock’ entails playing ‘Paper’,predicting ‘Paper’ entails playing ‘Scissors’, and predicting ‘Scissors’ entails playing ‘Rock’).

We will also create different difficulty level:

1. **Easy** – just a random guess out of the three possibilities.
2. **Medium** – based on previous inputs by the user and the maximum probabilities of the user to choose their next move.
3. **Hard** – In addition to the checking the previous inputs it will also keep a track of the current player playing style and pattern and make its next move.

LITERATURE SURVEY

1.) A Bayesian Model for Plan Recognition in RTS Games Applied to StarCraft

The task of keyhole (unobtrusive) plan recognition is central to adaptive game AI. "Tech trees" or "build trees" are the core of real-time strategy (RTS) game strategic (long term) planning. This paper presents a generic and simple Bayesian model for RTS build tree prediction from noisy observations, which parameters
are learned from replays (game logs). This unsupervised machine learning approach involves minimal work for the game developers as it leverage players' data (common in RTS). We applied it to StarCraft1 and showed that it yields high quality and robust predictions that can feed an adaptive AI.

2.) Value-function reinforcement learning in Markov games

Markov games are a model of multiagent environments that are convenient for studying multiagent reinforcement 17 learning. This paper describes a set of reinforcement-learning algorithms based on estimating value functions and resents 18 convergence theorems for these algorithms. The main contribution of this paper is that it presents the convergence theorems 19 in a way that makes it easy to reason about the behavior of simultaneous learners in a shared environment. Ó 2001 20 Published by Elsevier Science B.V.

STRATEGY

The question arises as to how the previous results and previous gestures influence the players' next gestures. We propose that there are 10 potential strategies that a player can adopt in a multi-round game of rock-paper-scissors. An example of one strategy is: a player repeats a gesture when they win, alternates gestures when they lose and chooses randomly when they draw. Thus for each of the three results, there are three potential responses (repeat, alternate,

random). There are therefore nine (three x three) potential strategies. Another example of this kind is when a player alternates their gestures when they win, chooses randomly when they lose and alternates their gestures when they draw.

Easy Mode-

In this mode, there is no as such strategy for the CPU. The CPU randomly chooses any one of the gestures i.e. rock, paper or scissors for every round.

Medium Mode-

In this mode, we calculate the probability of the gestures which user can use. The gesture with the highest probability means the user is prone to choosing it again so the CPU chooses that gesture with defeats the high probability gesture.

Hard Mode-

In this mode, we will observe the computer learning by picking a strategy and sticking to it for a while. Here are a few things to try:

- Pick Rock, then Paper, then Scissors, then Rock again and keep that pattern up. See how quickly the computer learns to beat you every time?
- Having done that a few times, change strategy and pick Paper 5 times in a row. See how the computer spots your change of strategy and alters its play?
- Pick any strategy of your own and see if the computer can spot the pattern.
- See if you can be perfectly random in your choices and beat the computer.

CURRENT BOTS

Iocaine Powder: This bot was created by Dan Egnor for a RPSbot competition in which bots played against each other. Iocaine Powder uses a combination ofthree strategies to predict its opponent's next move: guessing randomly, playing against the mostfrequently used throw, and history matching (i.e. finding patterns in the opponent's history).However, this bot's edge lies in the fact that it also detects the opponent's meta strategy (i.e. guessing one ahead, two ahead, etc…).

MegaHAL:Thisbot was created by Jason Hutchens for the same RPS competition described above. His approachwas to create a simple Markov model that stores frequency information about the opponent's playsfor all possible contexts. This allows the bot to predict the next play in the form of a probability distribution over all possible throws. The bot then picks the throw that maximizes the expected value of its score (where a win scores a 1, a tie 0, and a loss 1).It also only tracks the frequency information over a sliding window, in case the opponent changes strategies over time.

REVIEW III WORK:

ALGORITHM-

- Read the mode in which you want to play
- Easy
  - Play 10 rounds
  - The CPU chooses randomly which gesture to play
- Medium
  - Play 10 rounds
  - The CPU chooses that gesture which defeats the gesture with the highest probability
- Hard
  - Play 10 rounds
  - The CPU analyzes the pattern of gestures recorded before choosing one of them before each round
- After each mode ask the user if he/she wants to see the statistics of the gameplay
  - No
    - Exit the game
  - YES
    - It displays two bar graphs
      - Number of gestures
      - Result of the games played

SYSTEM DESIGN AND EXPLANATION

METHODS TO BE ADAPTED

Language Chosen to be used: Python

Our project shall contain three parts:

EASY – It will randomly choose one of the three choices and produce the output. So there is 50% chance of this bot winning based on sheer luck.

```
if(mode==1):
   print("...THE GAME IS ABOUT TO BEGIN...\n")
   for i in range(0,10):
      print("_____\nYou choose from r,p and s ...")
      user=input()
      cpu=random.randint(1,3)
      print(cpu)

      user=appoint_user(user)
      cpu=appoint_cpu(cpu)
      print_choice(user,cpu)
```

```
ROCK PAPER SCISSORS

Rules :
Select which mode you want to play - Easy(1) Medium(2) Hard(3)
1
...THE GAME IS ABOUT TO BEGIN...

_____________________________
You choose from r,p and s ...
r
1
USER chooses  rock
CPU chooses  rock
NOBODY WINS
_____________________________
You choose from r,p and s ...
p
2
USER chooses  paper
CPU chooses  paper
NOBODY WINS
_____________________________
You choose from r,p and s ...
s
3
USER chooses  scissors
CPU chooses  scissors
NOBODY WINS
```

```
You choose from r,p and s ...
r
3
USER chooses  rock
CPU chooses  scissors
USER WINS
_____________________________
You choose from r,p and s ...
r
1
USER chooses  rock
CPU chooses  rock
NOBODY WINS
```

MEDIUM – In this mode, the bot will keep a track of the various choices made buy all the users till date and come up with its next move based on the track record kept by it.

```
elif(mode==2):
    print("...THE GAME IS ABOUT TO BEGIN...\n")
    for i in range(0,10):
        print("_____________________________\nYou choose from r,p and s ...")
        user=input()
        f_r= open("C:/Users/PARMEET SINGH/Desktop/RPS/rps.txt","r+")
        r1=f_r.readline()
        sr=int(r1[7:])
        p1=f_r.readline()
        sp=int(p1[8:])
        s1=f_r.readline()
        ss=int(s1[11:])
        f_r.close()
        cpu=0
        if(sr>=sp and sr>=ss):
            if(cpu==0):
                cpu=2
        if(sp>=sr and sp>=ss):
            if(cpu==0):
                cpu=3
        if(ss>=sr and ss>=sp):
            if(cpu==0):
```

```
        cpu=1
    print(cpu)
```

```
ROCK PAPER SCISSORS

Rules :
Select which mode you want to play - Easy(1) Mediu
2
...THE GAME IS ABOUT TO BEGIN...


_____________________________
You choose from r,p and s ...
p
2
USER chooses  paper
CPU chooses  paper
NOBODY WINS
_____________________________
You choose from r,p and s ...
s
2
USER chooses  scissors
CPU chooses  paper
USER WINS
_____________________________
You choose from r,p and s ...
r
2
USER chooses  rock
CPU chooses  paper
CPU WINS
```

```
_____________________________
You choose from r,p and s ...
p
2
USER chooses  paper
CPU chooses  paper
NOBODY WINS
_____________________________
You choose from r,p and s ...
r
2
USER chooses  rock
CPU chooses  paper
CPU WINS
_____________________________
You choose from r,p and s ...
r
2
USER chooses  rock
CPU chooses  paper
CPU WINS
```

HARD – In this mode, the bot in addition to the medium mode will also keep a track record of the current player playing pattern and predict his next move keeping in mind the record pattern kept by it.

```
elif(mode==3):
   print("...THE GAME IS ABOUT TO BEGIN...\n")
   for i in range(0,15):
      f_r= open("C:/Users/PARMEET SINGH/Desktop/RPS/rps_hard.txt","r+")
      text=f_r.readline()
      f_r.close
      if(i==0):
         cpu=random.randint(1,3)
      elif(text.find("p")==-1 and text.find("s")==-1):
         cpu=2
      elif(text.find("r")==-1 and text.find("s")==-1):
         cpu=3
      elif(text.find("r")==-1 and text.find("p")==-1):
         cpu=1
      elif(re.match("rp",text) or re.match("ps",text) or re.match("sr",text)):
         if(text[len(text)-2:]=="rp"):
            cpu=2
```

```
    elif(text[len(text)-2:]=="pr"):
        cpu=3
    if(text[len(text)-2:]=="ps"):
        cpu=3
    elif(text[len(text)-2:]=="sp"):
        cpu=1
    if(text[len(text)-2:]=="sr"):
        cpu=1
    elif(text[len(text)-2:]=="rs"):
        cpu=2
elif(re.match("sp",text) or re.match("pr",text) or re.match("rs",text)):
    if(text[len(text)-2:]=="rs"):
        cpu=2
    elif(text[len(text)-2:]=="sr"):
        cpu=1
    if(text[len(text)-2:]=="pr"):
        cpu=3
    elif(text[len(text)-2:]=="rp"):
        cpu=2
    if(text[len(text)-2:]=="sp"):
        cpu=1
    elif(text[len(text)-2:]=="ps"):
        cpu=3
if(re.match("rps",text) or re.match("psr",text) or re.match("srp",text)):
    if(text[len(text)-3:]=="rps"):
        cpu=2
    elif(text[len(text)-3:]=="psr"):
        cpu=3
    elif(text[len(text)-3:]=="srp"):
        cpu=1
if(re.match("spr",text) or re.match("prs",text) or re.match("rsp",text)):
    if(text[len(text)-3:]=="spr"):
        cpu=1
    elif(text[len(text)-3:]=="prs"):
        cpu=3
    elif(text[len(text)-3:]=="rsp"):
        cpu=2
if(re.match("rrps",text) or re.match("rrsp",text)):
    if(text[len(text)-4:]=="rrps" or text[len(text)-4:]=="rpsr"):
        cpu=2
    if(text[len(text)-4:]=="psrr"):
        cpu=3
    if(text[len(text)-4:]=="srrp"):
        cpu=1
    if(text[len(text)-4:]=="rrsp" or text[len(text)-4:]=="rspr"):
        cpu=2
    if(text[len(text)-4:]=="sprr"):
        cpu=1
    if(text[len(text)-4:]=="prrs"):
        cpu=3
if(re.match("pprs",text) or re.match("ppsr",text)):
    if(text[len(text)-4:]=="pprs" or text[len(text)-4:]=="prsp"):
        cpu=3
    if(text[len(text)-4:]=="rspp"):
        cpu=2
```

```
        if(text[len(text)-4:]=="sppr"):
            cpu=1
        if(text[len(text)-4:]=="ppsr" or text[len(text)-4:]=="psrp"):
            cpu=3
        if(text[len(text)-4:]=="srpp"):
            cpu=1
        if(text[len(text)-4:]=="rpps"):
            cpu=2
    if(re.match("ssrp",text) or re.match("sspr",text)):
        if(text[len(text)-4:]=="ssrp" or text[len(text)-4:]=="srps"):
            cpu=1
        if(text[len(text)-4:]=="rpss"):
            cpu=2
        if(text[len(text)-4:]=="pssr"):
            cpu=3
        if(text[len(text)-4:]=="sspr" or text[len(text)-4:]=="sprs"):
            cpu=1
        if(text[len(text)-4:]=="prss"):
            cpu=3
        if(text[len(text)-4:]=="rssp"):
```

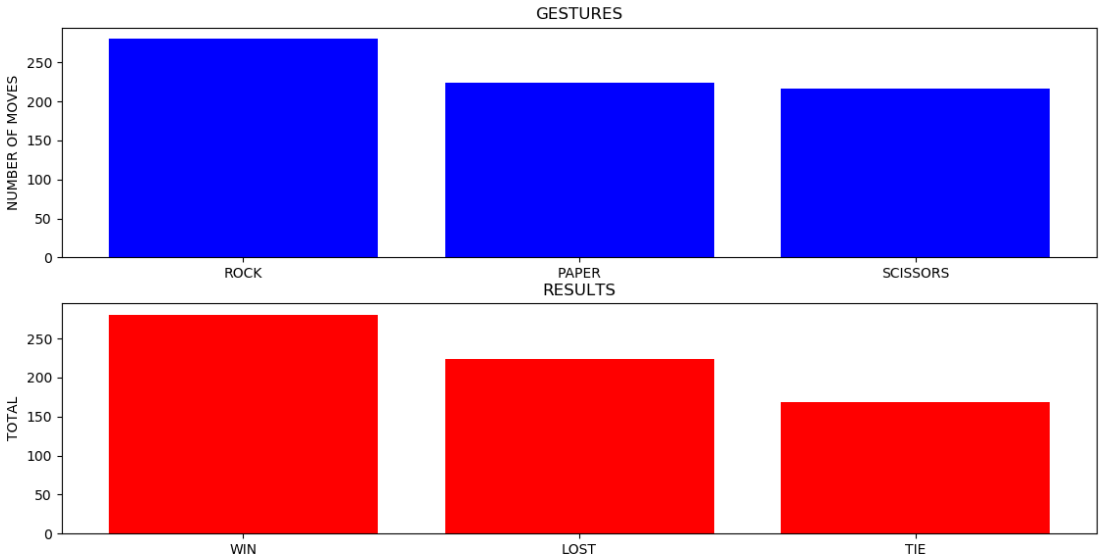
```
ROCK PAPER SCISSORS

Rules :
Select which mode you want to play - Easy(1) Medium(2) Hard(3)
3
...THE GAME IS ABOUT TO BEGIN...


_____________________________
You choose from r,p and s ...
r
2
USER chooses  rock
CPU chooses  paper
CPU WINS
_____________________________
You choose from r,p and s ...
s
2
USER chooses  scissors
CPU chooses  paper
USER WINS
_____________________________
You choose from r,p and s ...
p
2
USER chooses  paper
CPU chooses  paper
NOBODY WINS
```

```
You choose from r,p and s ...
r
2
USER chooses  rock
CPU chooses  paper
CPU WINS
_____________________________
You choose from r,p and s ...
s
1
USER chooses  scissors
CPU chooses  rock
CPU WINS
_____________________________
You choose from r,p and s ...
p
3
USER chooses  paper
CPU chooses  scissors
CPU WINS
_____________________________
You choose from r,p and s ...
r
2
USER chooses  rock
CPU chooses  paper
CPU WINS
```

STATISTICS



CONCLUSION

We have successfully implemented our game.We can play the game in three different modes i.e. Easy,Medium and Hard.
Easy mode is purely random based.
Medium mode is purely based on probability.
Hard mode is purely based on pattern recognition of the user's gestures
We have used numpy and matplotlib py to plot the number of moves of gestures and the results of the game with bar graphs

REFERENCES

1. http://ofb.net/~egnor/iocaine.html

2. http://web.archive.org/web/20020802183909/http://www.amristar.com.au/~hutch/roshambo/

3. Jordan MB. An actuarial artificial intelligence for the game rock-paper-scissors. S Afr J Sci. 2016;112(5/6), Art. #a0154, 3 pages. http://dx.doi.org/10.17159/sajs.2016/a0154