Making a Poker Agent

Sebastian Resendiz, A01701111, Tecnológico de Monterrey Campus Querétaro

***Abstract*— This document presents the implementation and the analysis of using a decision tree for a poker agent.**

Introduction

Poker is not only an incredibly fun card game, but it also brings millions of dollars per year to the big companies that run all the popular websites. Not only that, but working with the cars is hard, as it requires object detection, and classification for the cards.
This paper is written to serve as an understanding on how this specific Agent works.
The agent uses a model-reflex architecture, with a partial environment.
The model uses the existent OCR tesseract to correctly identify the characters in the screen and some OpenCV work to identify the Suits of the cards.

Data Set

This project uses the data set from UCI Data set repository titled Poker Hand Data Set that contains over a million of samples
Encoding:
Suite: 1: Hearts, 2: Spades, 3: Diamonds, 4: Clubs
Rank: 1: Ace, 2:2, ..., 10: Ten, 11: Jack, 12: Queen, 13: King
Label: 0: Nothing, 1: Pair, 2: Two pairs, 3: Three of a kind, 4:
Straight, 5: Flush, 6: Full house, 7: Four of a kind 8: Straight
Flush 9: Royal Flush

Model

Using the previous work by Jagdeep S. Sihota titled Poker Rule induction. We can compare the usage of multiple models such as Decision Trees, Neural Networks and Support Vector machines. Based on the paper we can conclude that using the support vector machine is the best option available as it presents >90% accuracy compared to the rest of the models. However for the simplicity of this agent, we will be using Decision Trees as they are the easiest to work with.

Implementation

For the first part we need a way to instantiate the game, for this the framework Selenium was selected, from here we can launch a browser with 248poker.com open

```
def init_game(self):
    print("Initializing agent")
    url = 'https://www.247freepoker.com/'
    self.driver = webdriver.Chrome()
    self.driver.set_window_size(1400,800)
    self.driver.set_window_position(0,0)
    self.driver.get(url)
    self.capture = Capture()
    self.suit = Suits()
    print("Page loaded")
    print("Starting game")
```

After this we can begin the game by simply clicking on the play button and then selecting the medium difficulty

```
def init_game(self):
    print("Initializing agent")
    url = 'https://www.247freepoker.com/'
    self.driver = webdriver.Chrome()
    self.driver.set_window_size(1400,800)
    self.driver.set_window_position(0,0)
    self.driver.get(url)
    self.capture = Capture()
    self.suit = Suits()
    print("Page loaded")
    print("Starting game")
```

After this we use some functions that constantly read the cars and pass them to an array

```
def getHand1(self):
    # get the two images
    img1 = self.capture.click(488,260,18,18)
    img2 = self.capture.click(528,355,30,30)

    # preprocess
    rank = self.capture.prelude(img1)
    suit = self.capture.prelude(img2,80,80)

    rankStr = self.capture.imageToString(rank)
    suitStr = self.suit.card_suit(suit)

    return [self.rankToInt(rankStr), suitStr]
```

We then can pass the list to a function that returns if we have any playable hand

```
def predict(self, cardsList):
   # the data should be encoded already
   # do a switch case of the list len 5,6,7
    listLen = len(cardsList)//2
    print(listLen)

    if listLen == 5:
        print("list of 5 cards")
        ls = np.array(cardsList)
        ls = ls.reshape(1, -1)
        pred = self.d_tree.predict(ls)
        print(f'the prediction for this hand is {pred}')
        return np.min(pred[np.nonzero(pred)])
    elif listLen == 6:
        print("list of 6 cards")
        arr = self.permamut_6(cardsList)
        pred = self.d_tree.predict(arr)
        return np.min(pred[np.nonzero(pred)])
    return 0
```

With that information we then can choose if we want to fold, check, or bet more money and then continue playing depending on the text on the screen

```
while flag<1000:
    # check if the menu is active
    text = agent.get_menu()
    state = agent.get_state()
    deal = agent.get_deal()
    if text == "Call":
        print("here")
        agent.make_decision()
    if text == "Check":
        aux = agent.getHand1()
        agent.make_decision()
        print(aux)
    if state in ["You Win!","Big Win!","Huge Win!"]:
        agent.continue_playing()
    if deal == "Deal Again":
        agent.continue_playing()
    print(flag, sep=" ")
    time.sleep(1)
    flag +=1
```

At the end this agent can very well run forever however we stop it after 1000 iterations.

Conclusions

I used a decision tree to implement a model-reflex agent, and show its viability. However it requires much more setup than previously expected

References

- Sihota, J. (2022). Poker Rule Induction. Retrieved 17 November 2022, from http://rstudio-pubs-static.s3.amazonaws.com/59520_f25e40618a3a442ca45b110b57a24a6f.html
- Walinton Cambronero, Poker Hand Dataset: A Machine Learning Analysis and a Practical Linear Transformation (2022). Retrieved 17 November 2022, from https://walintonc.github.io/papers/ml_pokerhand.pdf
- How Much Money Does Pokerstars Make Per Day?. (2022). Retrieved 17 November 2022, from https://www.poker-king.com/answers/how-much-money-does-pokerstars-make/