Tic-Tac-Toe

Hugo Veríssimo

January 29, 2024

COMMENT FUNCTIONS + INITIAL “ “ ” “ “ ” GET THEM BETTER AND SAY THERE WICH IMPORT WE USING

tic tac toe - estatistica de true games

tic tac toe - variation and symetry and stuff using pandas

. . . .

Introduction

1	2	3
4	5	6
7	8	9

```
import numpy as np
import pandas as pd
```

How many games

using NUMPY to get more efficient since all lists are only int numbers

```
def ALL_tic_tac_toe():
    """
    generate all tic-tac-toe games but,
        doesn't care who starts (X or O)
        doesn't care if theres a win, just fills all the nine squares
        impar numbers stand for the first player
        it's justs all the 9! games
    """
    move = 0
    ith_move = np.array([[0,0,0 , 0,0,0 , 0,0,0], [0,0,0 , 0,0,0 , 0,0,0]])

    move = 1
    while move < 9:
        all_games = np.array([0,0,0 , 0,0,0 , 0,0,0])
        list_all_games = [all_games]
        for j in range(1, len(ith_move)):
            for i in range(9):
                if ith_move[j][i] != 0:
                    pass
                else:
                    ith_move[j][i] = move
                    list_all_games.append(ith_move[j].copy())
                    ith_move[j][i] = 0
        ith_move = np.vstack(list_all_games)
        #print("Move " + str(move) + " - OK")
        move += 1

    move = 9
    last_move = np.where(ith_move[1:] == 0, 9, ith_move[1:])
    #print("Move " + str(move) + " - OK")

    np.savetxt("attachment(1)_alltictactoe.csv", last_move, delimiter=",", fmt="%d")

    return "File saved.", len(last_move)

ALL_tic_tac_toe()
```

```
## ('File saved.', 362880)
```

lets use pandas to preview da csv

```
pd.read_csv("attachment(1)_alltictactoe.csv", header = None)
```

```
##         0  1  2  3  4  5  6  7  8
## 0       1  2  3  4  5  6  7  8  9
## 1       1  2  3  4  5  6  7  9  8
## 2       1  2  3  4  5  6  8  7  9
## 3       1  2  3  4  5  6  9  7  8
## 4       1  2  3  4  5  6  8  9  7
## ...    .. .. .. .. .. .. .. .. ..
## 362875  7  9  8  6  5  4  3  2  1
## 362876  8  7  9  6  5  4  3  2  1
```

```
## 362877  9  7  8  6  5  4  3  2  1
## 362878  8  9  7  6  5  4  3  2  1
## 362879  9  8  7  6  5  4  3  2  1
##
## [362880 rows x 9 columns]
```

we can see we have the 9! ....

since we now have all the possible games lets see the ones according to the possible win using only NUMPY

```
def TRUE_tic_tac_toe():
    """
    remove the games from the set of 9! games where there are wins
    """
    def is_there_a_win(ttt):
        # convert 1D array into 2D array (3 by 3)
        TTT = ttt.reshape(3,3)

        # lines to verify for wins (lines, columns, diagonals)
        lines = np.vstack([TTT, np.transpose(TTT), np.diag(TTT), np.diag(np.fliplr(TTT))])

        # check sums to see if there is a winner
        X_vs_O = np.where(lines % 2 == 1, 1, -1)
        sums = np.sum(X_vs_O, axis=1)

        # someone won
        if 3 in np.abs(sums):
            # find the winning move
            last_move = np.max(lines, axis=1)
            last_move[np.abs(sums) != 3] = 10
            return True, min(last_move)

        # draw game
        else:
            return False, 0


    with open("attachment(1)_alltictactoe.csv", "r") as csvfile:
        rows = [line.strip().split(',') for line in csvfile]

    new_games = []
    for row in rows:
        ttt = np.array(row).astype(float)
        win = is_there_a_win(ttt)
        if win[0]:
            ttt[ttt > win[1]] = 0
            new_games.append(ttt)
        else:
            new_games.append(ttt)

    processed_games = np.vstack(new_games)
    unique_games = np.unique(processed_games, axis=0)

    np.savetxt("attachment(2)_truetictactoe.csv", unique_games, delimiter=",", fmt="%d")
```

```
    return "File saved.", len(unique_games)

TRUE_tic_tac_toe()
```

```
## ('File saved.', 255168)
```

as we can see we got the possiblities down to ... (% decrease from original)

lets see the csv using pandas

```
pd.read_csv("attachment(2)_truetictactoe.csv", header = None)
```

```
##         0  1  2  3  4  5  6  7  8
## 0       0  0  0  0  2  4  1  3  5
## 1       0  0  0  0  2  4  1  5  3
## 2       0  0  0  0  2  4  3  1  5
## 3       0  0  0  0  2  4  3  5  1
## 4       0  0  0  0  2  4  5  1  3
## ...    .. .. .. .. .. .. .. .. ..
## 255163  9  8  7  6  5  2  4  3  1
## 255164  9  8  7  6  5  3  2  1  4
## 255165  9  8  7  6  5  3  4  1  2
## 255166  9  8  7  6  5  4  2  1  3
## 255167  9  8  7  6  5  4  2  3  1
##
## [255168 rows x 9 columns]
```