Uniwersytet Warszawski
Wydział Nauk Ekonomicznych

Marcin Basiuk
Nr albumu: mbp-18217

Przegląd wybranych algorytmów uczenia maszynowego i sztucznej inteligencji na przykładzie gry w kółko i krzyżyk

Praca dyplomowa
Data Science w zastosowaniach biznesowych. Warsztaty z wykorzystaniem programu R.

Praca wykonana pod kierunkiem
dr. Piotr Wójcik
Zakład Finansów Ilościowych

Czerwiec 2018

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora pracy

Streszczenie

W pracy zaimplementowano trzy różne algorytmy uczące komputer gry w kółko i krzyżyk. Pierwszym algorytmem jest Minimax połączony z przycinaniem α - β. Następnie wykorzystano algorytm sieci neuronowych oraz algorytm genetyczny. Praca zawiera, opis porównanie działania algorytmów oraz aplikację do gry z komputerem wykorzystującym wybrany algorytm.

Słowa kluczowe

mimimax,teoria gier, uczenie maszynowe,sieci neuronowe, algorytmy genetyczne

Spis treści

Wprowadzenie 5

1. Minimax 7
1.1. Rys historyczny i trochę teorii 7
1.2. Opis algorytmu 7

2. Sieć neuronowa 9
2.1. Rys historyczny 9
2.2. Opis algorytmu 9

3. Algorytm genetyczny 11
3.1. Rys historyczny 11
3.2. Opis algorytmu 11

4. Podsumowanie 13

A. Kody 15
A.1. Klasa gry 15

B. Aplikacja 19

Bibliografia 21

Wprowadzenie

Celem pracy jest wykorzystanie trzech różnych algorytmów uczenia maszynowego w programie do gry w kółko i krzyżyk.

Pierwszy rozdział poświęcony będzie algorytmowi minimax. Jest to najbardziej klasyczne podejście do uczenia komputera grania w gry z dwoma graczami, gdzie wynikiem może być przegrania, wygrana lub remis. Ten algorytm jest od 40 lat wykorzystywany np. do grania w szachy.

W drugim rozdziale opisuję algorytm wykorzystujący sieć neuronową do nauki gry w kółko i krzyżyk. Sieć będzie się uczyła za pomocą klasycznego algorytmu 'stochastic gradient descent' grając z "graczem"posługującym się minimaxem. W rozdziale opiszę proces uczenia i strojenia sieci oraz końcowe rezultaty - czyli jak często algorytm oparty na sieci neuronowej wygrywa.

Na końcu chciałbym jeszcze spróbować zaimplementować algorytm genetyczny, który z grubsza polegać będzie na tym, że będę definiował profile/strategie graczy i sprawdzę jak dobrze grają przeciwko graczowi opartemu na minimax'ie. Wybiorę 10% najlepszych profili, część z nich zmutuję losowo i znowu będą grali przeciwko minimaxowi. Idea jest taka, że z każdego pokolenia wybieram najlepszych i część z nich mutuję. Po wielu pokoleniach, powinienem otrzymać skutecznie "wyhodowowanych genetycznie"graczy.

W ostatnim rozdziale, zamieszczę krótkie podsumowanie, a bardziej precyzyjnie moje refleksje dotyczące samych algorytmów jak.i również implementacji.

Do pracy chciałbym dodać dwa dodatki:

1. **DODATEK A** - napisane przeze mnie kody. Wygląda na to, że kody będą w Pythonie. Myślałem o przepisaniu na R, ale na tym etapie trochę wątpię w moje siły pod tym względem. Zobaczymy. Zależy mi też na tym, żeby większość algorytmów była napisana bez wykorzystania wysoko poziomowych pakietów. Wierzę, że takie podejście pomoże mi organicznie zrozumieć co się dzieje w tych algorytmach. A do takich źabaw"wydaje mi się, że Python jest lepszy. W R, z pewnością dałoby się to zrobić równie dobrze używając C++, ale tego języka nigdy nie używałem.
2. **DODATEK B** - aplikacja do gry w kółko i krzyżyk z przełączaniem z jakim graczem chce się grać (z jakim algorytmem)

Rozdział 1

Minimax

1.1. Rys historyczny i trochę teorii

Przed samym opisem algorytmu, napiszę tutaj parę słów minamxie i o teorii gier i jw szczególności jak minimax się w tą teorię wpisuje. Dodam trochę historii, między innymi o tym jak w latach 90-tych oparty na minimax'ie algorytm Deep Blue wygrał z uwczesnym mistrzem szachowym - Kaparowem.

1.2. Opis algorytmu

Potem przejdę do opisu samego algorytmu - czyli analizowane jest drzewa wszystkich możliwych rozwojów gry. W przypadku kółka i krzyżyk, to drzewo gry jest na tyle małe, że można analizować drzewo do samego dołu, ale w takich szachach, to już nie jest możliwe i trzeba ograniczyć się do analizowania tylko kilku kroków na przód. Do tego potrzebna jest jakaś funkcja heurystycznie oceniająca korzystność planszy - inaczej nie uda się wybrać ńajlepszegożuchu patrząc kilka ruchów w przód. Dodatkowo warto zastosować α - β przycinanie (ang. pruning), co znakomicie przyspiesza algorytm.

Rozdział 2

Sieć neuronowa

2.1. Rys historyczny

Tak jak poprzednio, w tej sekcji chciałbym trochę opisać koncepcję , rozwój i zastosowania sieci neuronowych. Jeszcze do uzupełnienia.

2.2. Opis algorytmu

Tutaj natomiast opis algorytmu i imlementacji. Jestem jeszcze przed napisanie kodu. Celuję w sieć z jedną warstwą ukrytą, 9 elementów wejściowych i 9 wyjściowych.

Sieci neuronowe często stosuje się do problemów klasyfikacji (np. rozpoznanie obrazów). W takiej sytuacji, podczas uczenia sieci, od razu dostępna jest informacja, czy sieć poprawnie zaklasyfikowała czy też nie. W przypadku gry w kółko i krzyżyk, sieć będzie musiał podjąć kilka decyzji i dopiero po zakończeniu partii będzie wiadomo, czy te ruchy były korzystne, czy też nie. Z mojej perspektywy jest to ciekawe utrudnienie. Jeszcze nie wiem, jak sobie z tym poradzić, ale jestem dobrej myśli.

Rozdział 3

Algorytm genetyczny

3.1. Rys historyczny

Jak poprzednio - krótki opis.

3.2. Opis algorytmu

W tej materii jeszcze niczego nie zrobiłem, ale będę walczył :-)

Rozdział 4

Podsumowanie

Tak jak zaznaczyłem we wprowadzeniu, tutaj opiszę - bardzo skrótowo - wrażenie z implementacji oraz podam jakieś informacje, który z algorytmów jest skuteczniejszy, szybszy i dlaczego. Na razie nic specjalnie mądrego nie przewiduję w tym rozdziale, ale wypadałoby, żeby jakieś podsumowanie było. Może po ukończeniu i przetestowaniu kodu, przyjdzie mi do głowy coś wartego napisania.

Dodatek A

Kody

Pytanie, czy warto tu umieszczać wszystkie, kody czy też nie ma sensu i może wystarczy link do gitHub'a.

A.1. Klasa gry

```
from Player import *

class Game(object):
    def __init__(self,Size=3):
        self.Size=Size
        board=[]
        empty_player=Player('-')
        for i in range(Size):
            board.append(['-'])
            for j in range(Size-1,):
                board[i].append('-')
        self.Board=board

    def newBoard(self,board):
        self.Board=board

    def getSize(self): return self.Size
    def getBoard(self): return self.Board
    def setBoard(self,n,m,Player):
        try:
            assert n in range(self.getSize()) and m in range(self.getSize())
            self.Board[n][m]=Player
        except AssertionError:
            print "Invalid move"

    def getBoardPosition(self,n,m):
        assert n<=self.getSize() and m<=self.getSize()
        return self.getBoard()[n][m]

    def move(self,n,m,Player):
        if self.getBoardPosition(n,m)!='-':
            return None
        try:
            self.setBoard(n,m,Player)
```

```
            return True
        except:
            print 'Invalid move'
            return None


    def check(self):
        ''' Checks if one of the players won the game. If so, returns a list with
            first argument binef True or False, and the second argument is the name
            of the player that won. If no player one, False is returned'''
        size=self.getSize()
        checklist=[0]*(2*size+2)

        for i in range(size):
            for j in range(size-1):
                if str(self.getBoardPosition(i,j))==str(self.getBoardPosition(i,j+1))
                     and str(self.getBoardPosition(i,j))!='-':
                    checklist[i]+=1
                if str(self.getBoardPosition(j,i))==str(self.getBoardPosition(j+1,i))
                     and str(self.getBoardPosition(j,i))!='-':
                    checklist[i+size]+=1

        for i in range(size-1):
            if str(self.getBoardPosition(i,i))==str(self.getBoardPosition(i+1,i+1))
                and str(self.getBoardPosition(i,i))!='-':
                checklist[2*size]+=1
            if str(self.getBoardPosition(i,size-i-1))==str(self.getBoardPosition(i+1,
                size-i-2)) and str(self.getBoardPosition(i,size-i-1))!='-':
                checklist[2*size+1]+=1

        for k in range(len(checklist)):
            if checklist[k]==size-1:
                if k<size:
                    return [True, str(self.getBoardPosition(k,0))]
                elif k<2*size:
                    return [True, str(self.getBoardPosition(0,k-size))]
                elif k==2*size:
                    return [True,str(self.getBoardPosition(0,0))]
                else:
                    return [True,str(self.getBoardPosition(0,size-1))]

        return False

    def checkTie(self):
        empty_slots=False
        for i in range(self.getSize()):
            if '-' in self.getBoard()[i]:
                empty_slots=True
        if self.check()==False and empty_slots==False:
            return True
        else:
            return False

    def lastMove(self):
        ''' returnt TRUE if there is only one move to go, and FALSE otherwise'''
```

```
    last=0
    for i in range(self.getSize()):
        for j in range(self.getSize()):
            if self.getBoardPosition(i,j)=='-': last+=1
    if last==1:
        return True
    else:
        return False


def __str__(self):
    line=''
    for i in range(self.getSize()):
        #line=line+'------\n'
        for j in range(self.getSize()):
            line=line + ' ' + str(self.getBoardPosition(i,j)) +' '
        line=line+'\n'
    return line
```

Dodatek B

Aplikacja

Tutaj zamieszczę krótki opis i ścreen shotź aplikacji do gry w kółko i krzyżyk.

Bibliografia

[1] Michael Nielsen, *Neural Networks and Deep Learning* Determination Press, 2015

[2] James Gareth, *Introduction to Statistical Learning* Springer, 2013