AI Report

Four in a Row (=Connect Four)

: Rule / Heuristic / NN (+ SVM) Approach

제출일: 2018.05.07

과목명: 인공지능

담당교수: 김현철 교수님

학과: 컴퓨터학과

학번: 2016320(137 / 198 / 144)

이름: 라건우 / 백진헌 / 태영준

목차

1. 개요

1) 과제 개요

Connect Four 는 2명이 번갈아 플레이하는 게임입니다. 이 게임은 7개의 column 과 6개의 row 로 구성된 board 내에서 매 턴 마다 각 player 가 한 column 을 선택하여 말을 놓습니다. 이 때 연속된 말의 개수가 4개 이상인 경우 해당 플레이어가 승리하게 됩니다.

본 과제는 이러한 Connect Four 게임을 구현하고, 해당 게임을 Play 하는 인공지능을 수업에서 학습한 내용을 바탕으로 만들어보는 것입니다. 기본적으로는 Rule 과 Search Algorithm 을 활용하여 대전 가능한 형식으로 코드를 작성하였으며, 추가적으로 Connect Four 승리 / 비김 / 패배 데이터셋을 모아 Machine Learning 중 Neural Network 와 SVM 을 바탕으로 학습시킨 모델 역시 사용하였습니다. 자세한 코드 설명은 하단에서 소개해 드리겠습니다.

2) Baseline

Search Algorithm 을 기반으로 한 Heuristic 코드를 작성할 때 Github 에 탑재된 https://github.com/ajendrosch/connect-4 코드를 활용하였습니다. 해당 Code 에서 활용한 파일은 "connect4.py" 하나이며, 활용한 부분에 대해 서술하자면 다음과 같습니다.

- Board (Board 와 관련된 함수들을 모두 Class 화 시켰습니다.)

  display_board: Board 를 화면에 출력하는 함수

  check: 연속된 말의 개수가 몇 개인지 확인하는 함수 (4개면 승리조건)

  insert: board 에서 말을 놓는 하는 함수

  uninsert: board 에서 말을 제거하는 함수

  poss_steps: 놓을 수 있는 column 을 return 하는 함수

- Heuristic (Heuristic 과 관련된 함수들은 모두 Class 화 시켰습니다.)

  minmax: minmax 를 alpha – beta pruning 과 함께 활용하는 함수

  evaluate: 현재 상태에 대해 score 를 해주는 함수 (내부 로직은 다릅니다.)

- Main

  main: 게임을 플레이하는 기본 함수 (틀만 이용했습니다.)

  winner: 승리자가 누구인지 알려주는 함수

해당 Github Repository 에서 활용한 부분은 위가 전부이며, 그 외에 해당 저장소에 탑재된 Neural Network 와 관련된 부분은 모두 해당 코드를 이용하지 않고 sklearn 에 있는 라이브러리를 활용하였습니다. 구체적인 사항에 대해서는 하단에서 소개해 드리겠습니다.

또한 저희 팀은 Neural Network 와 SVM 을 이용하여 Connect Four 현재 상태를 evaluation 할 수 있는 모델을 만들었으며, 해당 dataset 은 UCI connect-4을 전처리 하여 이용하였습니다. 이 역시 하단에서 구체적으로 소개해 드리겠습니다.
(http://archive.ics.uci.edu/ml/datasets/connect-4)

3) 파일 구조

저희 팀이 작성한 코드는 Github (https://github.com/JinheonBaek/AI-ConnectFour) 에 탑재되어 있습니다.

저희 팀이 작성한 Connect four 파일에 대한 구조를 설명 해드리면 다음과 같습니다.

- Data Folder

  UCI connect-4 dataset 이 들어있는 폴더 입니다.

- Model Folder

  학습시킨 NN (Neural Network – MLP) 및 SVM (Support Vector Machine) 이 들어있는 폴더 입니다. 두 모델은 모두 sklearn 라이브러리를 활용하여 간단하게 학습 시켰습니다.

- Learning.ipynb

  Jupyter Notebook 위에서 구동되는 파일이며, Github 웹페이지 위에서 (https://github.com/JinheonBaek/AI-ConnectFour/blob/master/Learning.ipynb) 코드와 결과들은 확인할 수 있습니다. Dataset 을 Load 하고 Pre-processing 하는 과정 및 MLP / SVM 학습 결과들에 대해 각각 Accuracy 와 같은 평가 함수를 활용한 내용이 작성되어 있으며, 모델을 저장하는 코드 역시 기록되어 있습니다.

- board.py

  Connect Four 보드에 대한 Class 파일 입니다. Class 내부 각 함수에 대해서는 하단 코드 설명에서 더 자세하게 말씀 드리겠습니다.

- heuristic.py

  Connect Four Heuristic Logic 에 대한 Class 파일 입니다. 해당 파일 내부에는 저희가 직접 만든 Evaluation Function 으로 구동되는 Heuristic 에 대한 Class 와 Data-driven 하게 학습시킨 모델의 Evaluation Function 을 바탕으로 구동되는 Heuristic 에 대한 Class 2 개로 구성되어 있습니다. 해당 Class 는 모두 Minmax & Alpha-beta Pruning 을 기반으로 실행되며, 파일 내부에 있는 각 클래스와 함수에 대해서는 하단 코드 설명에서 더 자세하게 말씀 드리겠습니다.

- main.py

  해당 Connect Four 게임이 돌아가는 main 함수가 기록되어 있는 파일입니다. 파일 내부에 있는 각 함수에 대해서는 하단 코드 설명에서 더 자세하게 말씀 드리겠습니다.

- rule.py

  Connect Four Rule 에 대한 Class 파일입니다. Rule 에 대한 설명과 클래스 내부 함수에 대해서는 하단 코드 설명에서 더 자세하게 말씀 드리겠습니다.

4) 기여도

- 백진헌

  전체적인 Class 구조 설계 및 코드 작성

  Heuristic Algorithm 개발과 Evaluation Function 생각 및 개발

  Data-driven 한 Evaluation Model (NN / SVM) 학습 및 적용

- 라건우

  Heuristic Class 구조 설계 및 코드 작성

  선배들이 활용했던 Rule / Heuristic Algorithm 에 대해 분석

  Heuristic 중 Evaluation Function 아이디어 제공

- 태영준

  Rule Class 구조 설계 및 코드 작성

  게임을 다수 Play 해보며 적용 가능한 Rule 에 대해 개발

  Rule / Heuristic Function 예외처리 (불가능한 상황 확인에 큰 기여)

2. 코드 설명 및 시행착오 기술

1) board.py (코드 설명)

Connect Four 를 플레이하기 위한 board Class 입니다.

- __init__

  board 객체가 생성되었을 때 board 를 list 형태로 만듭니다.

- get

  현재 board 상황을 list 형태로 return 합니다.

- getData

  현재 board 상황을 pandas 의 DataFrame 라이브러리를 활용하여 return 합니다. 학습에 이용하기 전 list 로 작성된 현재 board data를 학습 모델에서 바로 이용할 수 있게끔 전처리 하는 과정이 포함되어 있습니다.

- display

  현재 board 상황을 print 해서 보여주는 함수입니다.

- check

  현재 board 상황에서 연속된 말의 개수를 가로 / 세로 / 대각선 방향으로 모두 확인한 다음 return 합니다. 만약 connectNum 매개변수를 4로 설정해주었다면 연속된 4개의 말이 있는지 확인하여, 승리자가 누구인지 알 수 있게끔 역할을 다합니다.

- insert

  매개변수로 받은 column 에 말을 놓는 함수입니다.

-

- uninsert

  매개변수로 받은 column 에 해당 말을 제거하는 함수입니다.

- poss_steps

  현재 board 상황에서 놓을 수 있는 column 을 map 함수로 감싸 return 합니다.

2) main.py (코드 설명)

Connect Four 가 실행되는 main 파일 입니다.

board / rule / heuristic Class 에 대한 코드를 모두 import 해서 관리합니다.

- main

  Connect four 게임을 실행시키는 함수 입니다.

  Board / rule / heuristic Class 에 대한 객체를 만든 후, 승부가 정해질 때까지 게임을 반복 합니다.

- make_user_move

  유저(Player)의 턴이 되면 해당 유저가 말을 놓을 수 있는 함수입니다.

- winner

  Connect Four 게임이 끝난 후 해당 판의 형세를 바탕으로 누가 승리자인지 return 하는 함수입니다.

- firstMove

  Rule / Heuristic 알고리즘 에서 첫 번째 move 를 가운데로 둘 수 없게끔 제한한 조건을 해결하기 위해 만든 함수 입니다. (첫 번째 수는 column 3에 위치)

- getMode

  AI 의 턴이 되면 모드를 선택할 수 있게끔 만든 함수입니다.

  (1: Rule / 2: Heuristic / 3: Neural Network)

3) heuristic.py (코드 및 시행착오 설명)

Connect Four 의 AI – Heuristic Algorithm 에 대한 Class 입니다.

두 개의 Class로 구성되어 있으며, 첫 번째 Class 는 지식 기반의 Heuristic Algorithm 이고, 두 번째 Class 는 Data-driven 하게 작성한 Search 기반의 NN 입니다.

- Heuristic Class: __init__

  Search 깊이를 설정하고, 해당 board의 착수 지점에서 이길 수 있는 line 의 개수에 대한 Table 을 미리 할당해 놓습니다.

- Heuristic Class: make_ab_move

  Heuristic 을 이용하고 싶을 때 해당 함수를 호출하게 되며, 해당 함수 내에서 호출하여 반환된 Minmax 결과를 바탕으로 board 에 말을 놓게 됩니다.

- Heuristic Class: minmax

  make_ab_move 에서 호출하는 함수로, Minmax 와 Alpha-Beta Pruning 을 바탕으로 해당 플레이어에게 가장 유리한 착수 지점을 탐색하게 합니다.

  다만 말을 놓았을 때 4개의 말이 연속될 경우 해당 state의 값을 최대(최소)로 하게끔 처리하였습니다. 또한 4개의 말이 연속된 경우가 아니라면 해당 state 의 값을 evaluate 함수 내에서 처리하게 됩니다.

- Heuristic Class: evaluate

  현재 state 에 대한 score 를 매기는 함수 입니다. 만약 4개의 말이 연속된 경우가 존재할 경우 바로 최대 / 최소 값을 return 하게 되며, 그렇지 않을 경우 하단 Heuristic 을 사용하게 됩니다.

  우선 __init__ 에서 할당한 Evaluation Table 의 값을 기반으로 해당 착수점에 내 말이 있을 경우 + / 해당 착수점에 상대방의 말이 있을 경우 – 를 하여 sum 값을 계산합니다.

  그 다음 내 돌이 3개 연속인 경우에 대해 * 3 의 Weight 을 주고, 내 돌이 2개 연속인 경우에 대해 *1 의 Weight 을 주게 됩니다. 또한 상대방의 돌이 2개 연속인 경우에 대해 *-2 의 Weight 을 주게 됩니다.

- **Heuristic Class 에 대한 시행착오**

  처음에는 착수점에 대해 winning line 을 만들 수 있는 Evaluation Table 만을 만들어 사용하였습니다. 하지만 winning line 을 만들 수 있는 착수점이 가운데에 몰려 있었고, 이러한 특성 때문에 돌을 가운데에만 두려는 특징이 다수 나타났습니다.

  물론 이러한 특징 역시 winning line 만을 만들어 갈 때 유리하게 작용할 수 있지만, line 위 내 착수점이 가운데에 있다고 한들 상대방의 돌에 가로막히게 되면 큰 의미가 사라지게 됩니다.

  따라서 내 돌의 연결도와 상대방의 돌의 연결도를 기반으로 평가할 수 있는 Heuristic 알고리즘을 추가 했습니다. 이 알고리즘은 해당 state 에서 내 돌이 3개 연속되어 있을 경우와 2개 연속되어 있을 경우에 대한 가중치를 각각 부여하였고, 상대방의 돌이 2개 연속되어 있을 경우에 대한 음의 가중치 역시 부여하였습니다.

  상대방의 돌이 3개 연속인 경우에 대한 가중치를 부여하지 않은 이유는 내가 돌을 놓았을 때 상대방의 돌이 3개 연속인 경우 그 다음 차례에서 상대방이 이길 수 있는 가능성(돌이 막혀 있지 않다면)이 매우 높고, 한 단계를 거슬러 생각해보면 상대방의 돌이 2개 연속인 경우에 그 다음 차례에서 상대방의 돌이 3개 연속일 가능성이 많기 때문에, 3개 연속인 경우에 대해 evaluate 하지 않고 2개 연속인 경우에 대해서 높은 가중치를 주어 계산하였습니다.

  물론 상대방의 돌이 3개 연속인 경우 / 2개 연속인 경우 각각에 대해 가중치를 계산하여 evaluate 할 수 있겠지만, 실제로 돌려본 결과 3개 연속 / 2개 연속인 경우에 대한 각각의 가중치를 준 경우 보다 위 경우의 승리 확률이 저희 팀에 대해서는 더 많았습니다.

- NN_Heuristic Class: __init__

  Search 깊이를 설정하고, MLP Classifier 을 pickle 라는 라이브러리를 이용하여 load 합니다.

- NN_Heuristic Class: make_ab_move / minmax

  위 두 함수는 위 Heuristic Class 와 동일합니다.

- NN_Heuristic Class: evaluate

  이 역시 현재 state 에 대한 score 를 매기는 함수 입니다. 만약 4개의 말이 연속된 경우가 존재할 경우 바로 최대 / 최소 값을 return 하게 되며, 그렇지 않을 경우 학습한 결과를 바탕으로 점수를 매기는 모델을 사용하게 됩니다.

해당 모델은 현재 board 의 상태를 받아와 승리 / 비김 / 패배 에 대한 확률을 계산해주는 모델로, 이 모델에서 나온 확률을 바탕으로 계산한 승리 확률 – 패배 확률이 해당 state 에 대한 점수 입니다.

- **NN_Heuristic Class 에 대한 시행착오**

UCI 의 Connect four 데이터셋에는 현재 보드 상황에 대한 features 와 해당 보드 상황이 야기한 승리 / 비김 / 패배라는 세 가지 카테고리의 label 이 있습니다. 어느 순서대로 말을 두었는지에 대한 척도가 나와 있으면 강화학습 또는 RNN 과 같은 학습 방법을 이용할 수 있지만, 해당 데이터셋은 단순하게 현재 상황과 이 상황이 야기한 결과에 대해 기록되어 있었고, 또한 학습 기법이 중요한 과제는 아니기에 간단한 Machine Learning 학습 모델인 Neural Network 와 SVM 을 이용하였습니다.

처음에는 비김에 대한 데이터를 제외하고 승리 / 패배 label 을 기준으로 Classification 하려고 하였습니다. 하지만 두 데이터에 대해서만 학습을 진행할 경우 컴퓨터가 비기는 경우에 대해서는 잘못된 판단을 내릴 수 있다고 생각하여, 데이터 loss 없이 승리 / 패배 / 비김을 학습시킬 수 있는 multi label classification 을 이용하였습니다.

그 다음 학습시킨 모델을 바탕으로 현재 상황이 가지고 있는 승리 확률을 측정하였으며, 이 값을 통해 evaluation 을 진행 하였지만 생각보다 성능이 좋지 못하였습니다. 따라서 기존 search - heuristic 에서 이용한 방법처럼 minmax 와 Alpha-Beta Pruning 을 모두 사용하였으며, minmax 에서 현재 state 를 evaluate 하는 부분에서 학습한 모델을 사용하였습니다.

또한 단순하게 승리 확률 기준으로 착수 지점을 결정하지 않고, (승리 확률 – 패배 확률)을 계산하여 착수 지점을 결정하였습니다. 이 이유는 승리 확률만 고려하는 것보다 승리 확률 – 패배 확률을 고려하는 것이 나의 승리를 더욱 높일 수 있다는 생각에서 비롯된 것이며, 비김에 대해서 역시 학습했기 때문에 패배 확률이 동일한 상황에서는 비김 확률이 적은 것이 승리 확률이 높아진다는 생각에서 역시 비롯되었습니다. (승리 + 비김 + 패배 확률 = 1)

4) rule.py (코드 및 시행착오 설명)

Connect Four 의 AI – Rule 에 대한 Class 입니다.

크게 6가지의 rule 로 구성되어 있으며, 하단에서 상세하게 설명 드리겠습니다.

- make_move

  rule 을 이용하고 싶을 때 해당 함수를 호출하게 되며, 해당 함수 내에서 호출하여 반환된 rule과 column을 바탕으로 board 에 말을 놓게 됩니다. 룰이 적용되었다면 어떤 룰이 적용되었는지 print 를 통해 알려줍니다.

- **rule (시행착오와 함께 설명)**

  make_move 에서 호출하는 함수로, rule 을 바탕으로 해당 플레이어에게 가장 유리한 착수 지점을 계산합니다. 해당 rule 을 적용하기 전에는 현재 board 상황에서 돌을 놓을 수 있는 column 들을 모두 반환해 옵니다.

(1) Rule 1

- Board Class: check 함수 활용

내가 돌을 놓아 내가 승리한다면 그 지점에 나의 돌을 놓습니다.

(2) Rule 2

- Board Class: check 함수 활용

상대방이 돌을 놓아 상대방이 승리한다면 그 지점에 나의 돌을 놓습니다.

(3) Rule 3

- Rule Class: row_check 함수 활용

상대방이 돌을 놓아 Row 상에서 (가로 상에서) 연속된 돌이 3개 발생한다면 그 지점에 돌을 놓습니다.

이 Rule 을 개발한 이유는, Row 상에 상대방의 돌이 연속해서 3개 놓아져 있고, 나의 돌이 상대방의 돌을 하나라도 감싸지 않는 상황이 발생한다면, 반드시 제가 지기 때문입니다. (하단과 같은 상황을 사전에 방지)

(_ _ O O O _ X)

(4) Rule 4

- Rule Class: row_check 함수 활용

  Rule 3과 동일한 경우로, 내가 돌을 놓아 Row 상에서 (가로 상에서) 연속된 돌이 3개 발생한다면 그 지점에 돌을 놓습니다.

- Rule Class: double_check 함수 활용

  몇 번의 시행착오를 거치면서 내가 돌을 놓은 Column 위에 상대방이 돌을 놓았을 때 이기는 경우가 다수 발생한다는 사실을 알아차렸습니다. 따라서 내가 돌을 놓은 column 위에 상대방이 돌을 놓았을 때 상대방이 이기는 경우를 사전에 방지하고자 개발한 함수 입니다.

  해당 Column 에 돌을 놓았을 때 double_check condition 이 발생한다면 해당 Column 은 돌을 놓는 대상에서 제외합니다.

(5) Rule 5

- Rule Class: double_check 함수 활용

  위와 동일한 경우에 대해 확인하는 함수가 적용됩니다.

- Rule Class: evaluate

  Rule 을 다 확인한 다음 적용 가능한 Rule 이 없다면 단순한 evaluate 함수를 이용하여 각 state 에 대해 점수를 매긴 다음 가장 점수가 우수한 state 을 채택하는 함수 입니다.

  내 돌이 3개 연속인 경우에 대해 * 3 의 Weight 을 주고, 내 돌이 2개 연속인 경우에 대해 *1 의 Weight 을 주게 됩니다. 또한 상대방의 돌이 2개 연속인 경우에 대해 *-2 의 Weight 을 주게 됩니다.

(6) Rule 6

Rule 5 에서 double_check 가 적용되어 더 이상 놓을 column 이 없다고 판단되는 경우에 대한 예외처리를 위해 Rule 6을 개발하였습니다.

비록 double_check 가 발생하여 지는 상황이 발생하더라도 무한 loop 에 돌지 않게 하기 위해 놓을 수 있는 지점을 확인하여 그 지점을 반환 하는 함수입니다.

3. 기타

1) Heuristic 의 Depth

Heuristic 의 Search Depth 선정에 대해 고민이 많았습니다.

간단하게 Evaluation Table 만을 만들 경우 깊은 Depth 를 탐색할 수 있다는 장점이 있지만, 단순한 winning line 개수만으로 해당 state 의 값을 결정한다는 것이 큰 한계로 다가왔습니다.

반면에 Evaluation Table 뿐만 아니라 현재 상황에서 연결된 나의 말 개수와 상대방의 말 개수를 계산하여 Table 에서 계산된 값에 더하는 방식을 채택할 경우 깊은 Depth 를 탐색할 수 없다는 단점이 있었지만, 저희 팀에 대해서는 Heuristic 이 훨씬 더 똑똑하다는 인상을 받았습니다.

따라서 더 깊은 Depth 를 탐색하지 못하지만 탐색한 state 에 대해서는 보다 잘 판단하는 Heuristic 을 개발하여 채택하였습니다.

2) (Neural Network) vs (SVM), 그리고 Neural Network

SVM (Support Vector Machine) 대신 Neural Network 를 채택한 배경은 다음과 같습니다.

	Neural Network	SVM
Accuracy	**0.8337**	0.7689
Precision	0.8658	**0.8763**
Recall	**0.8337**	0.7869
F1 Score	**0.8473**	0.8275

또한 Neural Network 의 Hidden Layer 을 (50, 20)으로 설정하였습니다. 기존에는 (100, 50, 20)으로 설정하였는데 Overfitting 과 같은 문제로 인해 test 데이터셋에 대한 성능이 좋지 못해 Hidden Layer 규모를 낮췄습니다.