CE888 Assignment 1 - Decision Tree Classifier with MCTS for Noughts and Crosses

Chelmsford, Essex, UK
plgent@essex.ac.uk

GitHub: https://github.com/peterlloydgent/ce888labs

Dataset: https://github.com/peterlloydgent/ce888labs/blob/master/CE889ASSIGNMENT_GENTP53602/10000games.csv

Reg No: 1802559

Peter Lloyd Gent
University of Essex

***Abstract*— This study provides unique and novel approach to finding optimal moves for the game of noughts and crosses. This will be achieved by training decision trees with optimal moves being evaluated by MonteCarlo Tree Search (MCTS) based on the current state of a game. In related studies by Nunes et al the accuracy to beat from bleeding edge research is 93.8% accuracy [7]. This paper is in effect an outline of the approach proposed to be taken to create such a program that can compete with such impressive figures. The paper is also a primer to topics such as Decision Tree (DT) induction, MCTS basics and approaches to DT induction using MCTS.**

I. INTRODUCTION

The given project task is to train a decision tree classifier for the purpose of predicting the best moves to take in any given state of the game of noughts and crosses. The game is simple in its premise: each player takes it in turn to mark a nought or cross depending upon their chosen symbol until a row or diagonal of three of the same symbols are achieved or resulting in a draw if no player wins outright. Although noughts and crosses is from the outset a relatively simple game it gives each player at the start a combination of 9^9 different moves from the outset. A typical approach to solving such a problem is to use Monte Carlo Tree Search (MCTS) alone. However for the purpose of this paper the task is to augment MCTS with a decision tree classifier for the purpose of rollout (exploration of next states for MCTS). The motivation for precomputing the moves available with a decision tree is twofold. Firstly, the performance gains should be noticeable as the decisions have been precomputed so that for a game such as GO the rollout may be optimised [1]. Secondly, unlike with regular MCTS the use of decision trees results in a situation where a non random path for the decision tree can be selected. In short this means that a better selection of moves are made rather than random moves.

In the next sections a literary review of the background to MCTS and decision trees will be covered to give the background knowledge required to solve the task. The section on methodology will detail what the paper sets out to achieve and how the task is to be achieved. The section on experiments will outline the results of any other studies relevant to the task at hand. Finally the discussion and conclusion will evaluate and conclude upon the projects outcomes and the plan will outline what and when project goals will be achieved using descriptors such as Gant charts.

II. LITERARY REVIEW

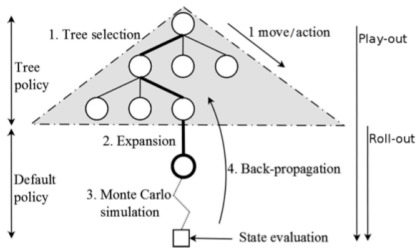

fig 1. Four phases of MCTS - selection, expansion, simulation or 'rollout' and back propagation [2]

As Thomas and Tian note MCTS is "an any-time best-first tree-search algorithm" [2]. It is considered an 'anytime algorithm' because one of the major advantages of the algorithm is that it can terminate at any number of iterations and still provide a valid next action to take, meaning that it is extremely fast to resolve tree search queries [2]. It is also a form of reinforcement learning that relies on a four stage process of: selection, expansion, simulation and back-propagation (fig 1). The algorithm starts from a root node and builds the tree iteratively [2]. The first stage in this process is tree selection, whereby an unexpanded node is selected for possible expansion. The algorithm then balances itself between exploring new nodes or taking a well travelled path by something called an upper confidence bound (UCB) [2]. This is in effect the MCTS tree policy that is at the heart of the algorithms ability to balance exploitation or exploration.

$$a^* = \arg\max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right\}$$

fig 2. UCB for a given MCTS

The UCB (fig 2.) works by using a constant C as a coefficient to weight to the choice between exploration and exploitation. The higher the value of C, the more weight will be given to exploring new nodes. The Q(s,a) on the other hand provides the highest average reward [2]. Once a node has been selected by using the UCB the node is expanded and added to the tree. A simulation or rollout then occurs where the algorithm looks ahead to a possible end state, which in the case of vanilla MCTS chooses random nodes on the way to a leaf node with the best reward. Finally the back propagation stage the reward from this exploration is

returned and the statistics in each node on the path to the most recently expanded node are updated. It then returns the next action to take depending on the statistics in the root node. If the UCB exploration value is high an entirely different branch may be taken from the root node down. While the algorithm is extremely efficient the random nature of tree exploration means that non optimal choices are possible, and this is where the augmentation with a decision tree for rollout will provide potentially better exploration or exploitation of possible future nodes.

A decision tree may perform regression over a continuous set of variables or classification in order convert a discreet value into an outcome [4]. For the purposes of this paper the focus will be on classification in order to get the best moves selected or classified depending upon game state. In general decision trees are trained either through the Gini criterion or forms of decision tree inference based on, or in actual fact ID3 itself. The originator of ID3 J Quinlan introduces [5] ID3 in his original paper as being suitable for cases with many attributes and many objects without much computation. The process of forming a tree is described [5] as being iterative with a window moving across the dataset constructing a tree for all the data within each window. At each step the decision tree is rooted to the attribute that gives the most information at any given point. The information gain is calculated through the use of Shannon's information function that gives us the information gain

$$\text{Informatio}\quad n = -\sum_{i=1}^{N} p_i \ln_2(p_i)\text{bits}$$

fig 3: Shannon's information function [4]

The function above (figure 3) gives a value between 0 and 1 for information gain. When taken with two information values that are subtracted from each other for a different classification it provides an idea of how uncertain it is for a given variable to be a member of either class. With further variables it is then possible to work out how much information is gained from these attributes and hence a decision tree may be trained by splitting on attributes that give the most information for a classification. The scikit learn package uses a heavily modified version of C4.5 (the successor to ID3 and ID4) called CART [6]. However the basis for these algorithms i.e. information theory has been covered, as has the basis of MCTS. The synergy of both of these will now be covered as a novel means of creating an expert tic tac toe player.

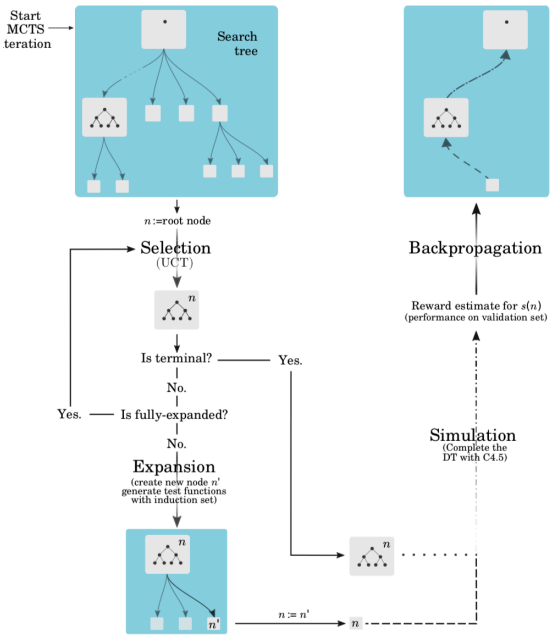

fig 4. [7]

The novel part of the proposed solution is to marry both the strengths of decision trees with that of MCTS to produce an expert tic tac toe player. This is an area with little or no prior research. The only relevant research that could be found [7] for this implements decision tree learning in each of the MCTS processes of selection, expansion, simulation and back propagation (above). For simulation the algorithm uses c4.5 to complete a given decision tree, and instead of discarding the tree after finding the next best move outputs a decision tree containing the entire search tree [7]. In this case the data set takes a dataset as input, splitting between validation and induction (training) sets. It then uses this split data in the expansion and simulation phases when applying the tree policy (UCB1).

III. METHODOLOGY

```
def UCTPlayGame(data_frame):
    """ Play a sample game between two UCT players where each player g
        of UCT iterations (= simulations = tree nodes).
    """
    # state = OthelloState(4) # uncomment to play Othello on a square
    state = OXOState() # uncomment to play OXO
    #state = NimState(15)  # uncomment to play Nim with the given numb
    df = pd.DataFrame()
    while (state.GetMoves() != []):
        if state.playerJustMoved == 1:
            m = UCT(rootstate=state, itermax=1000, verbose=False)  # p
        else:
            m = UCT(rootstate=state, itermax=100, verbose=False)
        df = logState(state, m, df)
        state.DoMove(m)
    if state.GetResult(state.playerJustMoved) == 1.0:
        print("Player " + str(state.playerJustMoved) + " wins!")
        df = updateWinField(df,state.playerJustMoved)
        return concat_data_frames(df,data_frame)
    elif state.GetResult(state.playerJustMoved) == 0.0:
        print("Player " + str(3 - state.playerJustMoved) + " wins!")
        df = updateWinField(df,str(3-state.playerJustMoved))
        return concat_data_frames(df,data_frame)
    else:
        print("Nobody wins!")
        df = updateWinField(df,0)
        return concat_data_frames(df,data_frame)
```

```
def logState(state, move,data_frame):

    player = None
    if state.playerJustMoved==2:
        player = 1
    else:
        player = 2
    player_state_move_endgame = {'player':player}
    player_state_move_endgame['0'] = state.board[0]
    player_state_move_endgame['1'] = state.board[1]
    player_state_move_endgame['2'] = state.board[2]
    player_state_move_endgame['3'] = state.board[3]
    player_state_move_endgame['4'] = state.board[4]
    player_state_move_endgame['5'] = state.board[5]
    player_state_move_endgame['6'] = state.board[6]
    player_state_move_endgame['7'] = state.board[7]
    player_state_move_endgame['8'] = state.board[8]
    player_state_move_endgame['best_move'] = str(move)
    df = data_frame.append(player_state_move_endgame,sort=False,ignore_index=True)
    return df

def updateWinField(data_frame,won):
    data_frame['won'] = won
    return data_frame

def concat_data_frames(df,data_frame):
    frames = [df,data_frame]
    return pd.concat(frames)
```

fig 5 and fig 6 for code to log game state and best move and won

The purpose of performing the study is to see whether or not if augmenting MCTS with decision trees for rollout can produce an expert tic tac toe computer player. The methodology for the project can be described as a three stage process of first gathering data, training the data and then with predicted data feeding back into the cross validation process as an iterative process of training an expert tic tac toe AI player. The process of collecting data involved writing some code to log the player currently available to move, the game state at the present time, the recommended move by MCTS, and finally after the fact whether or not the combinations of moves resulted in a win, loss or draw for either player. This is achieved in the code by calling UCTPlayGame with an outer DataFrame from the main function, and then creating a new inner DataFrame for each new session. At the point of each move the function log state is called and then a dictionary is built to initialise the current player, state etc and appends it to the inner DataFrame. When a player has won or drawn a game the algorithm will finally call updateWinField to set the entire game's fields to 0,1 or 2 depending on a draw or win for player 1 or 2. This process was able to generate 10,000 games initially for the purpose of training a decision tree classifier with something like an sci-kit learn DecisionTree. An example of this data can be seen below:

```
,0,1,2,3,4,5,6,7,8,best_move,player,won
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,8,1.0,2
1,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0,2.0,2
2,2.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,5,1.0,2
3,2.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,2,2.0,2
4,2.0,0.0,2.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,1,1.0,2
5,2.0,1.0,2.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,6,2.0,2
6,2.0,1.0,2.0,0.0,0.0,0.0,1.0,2.0,0.0,0.0,1.0,3,1.0,2
7,2.0,1.0,2.0,1.0,0.0,0.0,1.0,2.0,0.0,0.0,1.0,4,2.0,2
8,2.0,1.0,2.0,1.0,2.0,1.0,2.0,0.0,0.0,1.0,7,1.0,2
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,4,1.0,0
```

IV. EXPERIMENTS

The first and initial training of a decision tree from MCTS yielded some very strange results. The first thing noted is that the tree is not nine wide by nine deep. This was expected as there are nine possible beginning moves for any game of tic tac toe and because there are only nine cells to be filled it should be a maximum of nine deep in order to complete any game. Instead the trained network seems to produce a network with eleven initial starting moves. In a similar vain the importance of sequential turns between player 1 and player 2 seems to be lost after training the network. With further development the network may have to drop the player field from the training variables in order to focus the trained decision tree purely on the state of the game and predict outputs (best moves or winning trees).

An alternative view to this based on advice from experts (GLAs) that this is merely the first part of the process of training a decision tree and that the MCTS needs to be run with the decision tree in order to generate the next iteration of decision tree. In practice either of these possibilities needs to be investigated with further experiments to test the sanity of the decision trees that are inferred from the input data. This may also require further investigation of entropy vs Gini criterion as a classification method for building decision trees with scikit learn. Otherwise it may be preferable to build from scratch a decision tree that uses neither Gini or Entropy (information gain) in order to create a network that properly represents a game of tic tac toe. In order to work out what criterion is best and whether or not it's possible to do so with scikit learn it will be necessary to perform further experiments. It is with this aim we look ahead to other research performed by Nunes et al for a comparison of performance between different decision tree (DT) outcomes.

Dataset	(1) C4.5 DT			(2) Evol. DT			(3) Best MCTS DT					(4) Best MCTSp DT				
	Acc	F1	L	Acc	F1	L	Acc	F1	L	D	T	Acc	F1	L	D	T
Balance Scale	81.6	81.1	30	85.6	85.6	35	87.4	**87.4**	12	13	5:29	84.5	84.4	7	35	2:02
Banknote auth.	97.6	97.6	14	93.0	92.8	29	98.1	**98.0**	8	11	24:17	98.1	**98.0**	8	17	14:21
BHP	94.1	93.8	27	74.2	73.5	3	86.3	85.8	9	9	29:39	97.1	**96.9**	28	30	13:37
Biodegradation	81.4	77.6	13	78.5	74.0	21	82.7	**79.9**	9	11	56:53	81.7	78.5	14	19	46:52
Breast cancer	91.9	91.1	5	90.7	89.7	3	92.4	**91.9**	3	5	0:04	92.4	**91.9**	3	5	0:05
Car evaluation	89.4	73.8	32	93.8	**83.8**	218	82.9	71.0	12	14	17:48	88.3	78.1	19	42	5:00
Contraceptive	48.3	45.8	43	55.1	49.5	57	52.4	51.6	8	10	30:22	56.7	**53.4**	28	53	36:56
Credit approval	85.5	85.1	6	87.0	**86.9**	17	80.7	80.6	9	12	25:46	81.6	81.3	10	18	24:23
Solar flare	71.5	58.9	59	72.4	**63.0**	121	70.7	58.8	11	13	21:00	74.2	62.3	40	55	34:26
German credit	75.4	**65.7**	20	72.7	61.8	81	69.9	63.4	12	13	27:58	73.4	63.9	22	31	15:00
Indian liver	71.0	**62.7**	24	64.2	57.2	321	68.8	60.9	8	10	17:58	61.4	58.5	21	40	11:47
Arterial catheter	87.8	87.7	6	88.0	**87.9**	26	86.9	86.8	7	12	29:08	87.1	87.0	5	13	13:59
Language	78.3	60.7	8	79.1	57.8	9	75.1	**61.8**	2	7	2:04	75.1	**61.8**	2	7	3:19
Localization	96.3	96.3	23	96.5	96.5	13	96.8	**96.8**	9	12	17:50	96.5	96.5	17	29	30:44
Mammographic	82.0	**82.0**	11	78.5	78.5	22	80.0	80.0	10	12	3:20	80.3	80.3	9	12	2:11
Diabetic	66.5	66.2	11	67.6	**67.6**	7	65.9	65.9	10	12	86:16	66.8	66.6	18	24	56:01
Phishing	86.3	83.4	34	89.9	**89.7**	96	84.3	76.2	12	12	15:30	86.1	81.3	21	32	14:35
Pima indians	74.0	70.7	24	74.5	68.9	3	77.5	74.0	8	11	17:35	80.1	**77.4**	17	28	14:36
Student	75.0	**74.6**	12	75.0	73.5	9	74.0	72.5	9	12	18:37	69.4	68.9	15	26	26:33
Synthetic 1	70.8	70.9	24	60.7	59.8	29	67.3	67.2	9	10	64:25	73.5	**73.7**	20	29	85:27
Synthetic 2	70.1	**70.0**	27	62.6	61.8	43	63.0	62.7	10	10	59:09	63.0	62.7	10	10	60:33
Synthetic 3	81.1	80.3	21	77.6	76.6	13	74.5	74.0	8	9	38:22	81.6	**80.9**	26	28	31:08
Synthetic 4	79.6	72.8	3	74.0	60.5	9	77.4	**73.1**	5	7	0:05	77.4	**73.1**	5	7	0:10
Synthetic 5	48.4	48.3	78	46.9	46.9	15	47.3	44.1	8	10	78:53	55.1	**55.3**	50	85	105:13
Synthetic 6	51.2	49.8	22	50.2	49.8	17	52.1	50.5	8	11	72:48	51.2	**51.1**	26	37	87:37
Synthetic 7	57.1	56.4	43	64.8	**58.6**	23	55.6	52.1	10	11	70:54	55.6	52.1	10	11	57:22
Synthetic 8	61.9	**55.6**	29	59.1	52.2	95	66.3	53.9	10	12	23:36	63.0	55.1	32	38	6:47
Tic-tac-toe	91.7	90.3	33	78.1	74.8	193	82.6	77.5	13	14	20:25	93.8	**92.9**	30	49	11:44
Titanic	78.7	70.0	8	78.1	**71.9**	3	75.6	71.3	8	9	0:15	75.6	71.3	5	9	0:17
Wine quality	70.5	70.5	107	71.5	**71.3**	117	70.5	69.6	9	9	42:10	70.5	69.6	9	9	46:06
Yeast	67.5	66.4	9	66.0	65.8	23	65.3	65.2	8	12	17:27	68.3	**68.2**	9	20	14:47

fig 7. Results of a comparative study into c4.5 trained DTs, evolutionary DTs, MCTs and MCTs with pruned decision trees [7].

The study by Nunes et al [7] (above) shows best in class performance for tic tac toe as well as overall with it excelling in thirteen categories overall compared to competing algorithms for MCTS DT, Evolutionary DT and C4.5 DT. This demonstrates the potential of MCTS algorithms for training decision trees as the process has created a winning tic tac toe outcome of 92.9% score and a 93.8% accuracy. If the development of a competing solution is to be as accurate it may require implementation of a tree pruning, which is described by Nunes et al as a "way to tackle exponential tree growth by removing suboptimal nodes" [7]. The removal of these nodes meant that in total MCTS with pruning outperformed c4.5 in twenty out of the thirty data sets (including tic tac toe).

Nunes et al also remark that evolutionary decision trees were best at exhaustive tasks. On the other hand the MCTS without move pruning was superior to that of C4.5 without MCTS in eleven out of thirty one data sets. In contrast in twenty out of the thirty datasets where MCTS did worse than vanilla C4.5 decision trees the MCTS solution was considerably more shallow [7].

V. DISCUSSION

As previously discussed the key to evaluating the performance of the decision trees that are trained is to use cross validation over ten partitions, leaving one for purely test (validation) purposes. In each iteration of training the decision tree the MCTS must be run with the decision tree classifier in order to output the next possible set of best moves before being fed back into the MCTS system. This over multiple iterations may provide a solution that is comparable to Nunes et al. It remains to be seen however if or not a solution can be created that prunes sub optimal nodes in order to achieve a 92.9% score or 93.8% accuracy comparable to the best case in Nunes et al's study [7].

However, in order to provide results comparable to that of Nunes et al it will be necessary to provide some measure of accuracy and an overall score. The study by Nunes et al does not detail how exactly these statistics were generated so it remains to be seen how some statistic for accuracy or overall score might be generated.

VI. CONCLUSION

In conclusion further experimentation is required in order to work out the best course of action. This includes selecting

the best criterion for tree construction (Gini or CART), or doing away with automated tree construction altogether and constructing a proprietary decision tree solution. In the latter case this would allow for pruning of trees to provide a comparable performance to that of Nunes et al. However the preferential approach may be to prototype first with sci-kit a decision tree approach and then work on a proprietary system. These approaches need to be further explored in order to provide a conclusive approach that will solve the task at hand.

Also what needs to be considered is an appropriate way to measure accuracy or to generate a score like that in the study of Nunes et al for accuracy in particular so that the performance of the algorithm can be measured in comparison to C4.5 DTs in particular (as this is closest to the paper's solution, but also pruned trees for evaluation of the project in a subsequent report. This report will round up the approach proposed in this initial report and provide results to compare the results to Nunes report.

REFERENCES

[1] Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert et al. "Mastering the game of Go without human knowledge." Nature 550, no. 7676 (2017): 354.

[2] P. Diago, S. Samothrakis and S. Lucas, "Knowledge-based Fast Evolutionary MCTS for General Video Game Playing", 2014.

[3] [Anthony, Thomas, Zheng Tian, and David Barber. "Thinking fast and slow with deep learning and tree search." In Advances in Neural Information Processing Systems, pp. 5360-5370. 2017.

[4] L. Citi, "DECISION TREE INDUCTION", University of Essex, 2019.

[5]J. QUINLAN, "Induction of Decision Trees", *Hunch.net*, 1986. [Online]. Available: http://hunch.net/~coms-4771/quinlan.pdf. [Accessed: 19- Feb- 2019].

[6] "1.10. Decision Trees — scikit-learn 0.20.2 documentation", *Scikit-learn.org*, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart. [Accessed: 20- Feb- 2019].

[7] C. Nunes, M. De Craene, H. Langet, O. Camara and A. Jonsson, "A Monte Carlo tree search approach to learning decision trees", in *17th IEEE International Conference on Machine Learning and Applications*, Orlando, Florida, 2018.

IV. PLAN

The plan for the project is to continue prototyping a solution using SciKit learn until it becomes more clear what criterion to use for splitting decision trees, for example. If the prototype cannot be completed within the deadline, then other methods may be investigated i.e. developing a decision tree class from scratch. This would allow for pruning of sub optimal trees to give a highly accurate solution like in the work by Nunes et al. Fortunately, all of the data has been collected so the next roadblock is training decision trees into something that seems sane.

	18/02/2018	25/02/2018	04/03/2018	11/03/2018	18/03/2018
Collect data					
Develop prototype using scikit learn					
develop full solution (if prototype points to solution (propritory trees or further development))					
write up final report					