CE888 Data Science final report

Peter Gent
CSEE, Essex University
Chelmsford, Essex
plgent@essex.ac.uk

Reg no: 1802559

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

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

code: https://github.com/peterlloydgent/ce888labs/blob/master/CE889ASSIGNMENT_GENTP53602/MCTS2.py

Abstract-

I. INTRODUCTION

This second report focuses on the results of four different approaches to the task of improving upon a decision tree classifier for the task of playing games of tic tac toe. The experiments took four approaches for the algorithm and logged the results to get the most optimal results. In the end some of these approaches faired better than others overall but there was not any approach that produced results that were majorly better than another. In the end most of the results were up to twenty percent scoring, even with a high accuracy rating (eighty percent or more as an improvement beyond the initial score) for the best moves. In at least one of the attempts the dataset was filtered for winning combinations only and duplicates. It was expected that this would yield better results for scores by refining best moves to those set of moves that result in a wining outcome. In practice there was no significant improvement in the scoring of the trees from this approach, contrary to expectations. The results for the respective approaches are stated in the experiments section and discussed in further detail in the discussion section. This leaves the next section to delve into the theoretical background material and some of the key statistics from the first paper out of the two.

II. BACKGROUND

As much of the background on Monte Carlo Tree Search (MCTS) has been covered in the first report this will be summarised and restated here for the purpose of refreshing the main terminology on MCTS and decision trees (DTs). The MCTS algorithm is described [1] as being an "anytime best first tree search algorithm" what this means is that at any point the algorithm can exit and still provide the best move at that point, giving the algorithm an element of redundancy. A critical part of the reinforcement learning element to the algorithm is the upper confidence bound that balances greedy (well travelled routes) over the exploration new routes. In most cases the rollout of the MCTS that explores new routes in the tree is random. For the purpose of the assignment this can be improved upon by using a DT classifier to chose moves that are more optimal, rather than random as in the original MCTS code that has since been adapted.

This is the role of a DT which in itself is fairly straightforward to understand. It is simply a set of nodes for which in a classification mode splits decisions based upon what features give the most information to discern between different end classifications. This is achieved by something called Shannon's information function to work out what decisions in the tree represent the best features to split on to determine what the end result should be. In the previous report the experiment approach was to consider different ways that the decision tree could be trained or even the approach of manually creating the decision tree rather than relying on a library. The approach taken in the actual experiment has been to use a decision tree that is trained repeatedly based on data produced by previous decision trees to attempt to extract features that provide a better i.e. more accurate for the chosen best move and a higher score overall.

In terms of comparable scores or benchmarks the paper cited in the original report (by nunes et al) cites [1] the state of the art DT approach as having a 92.9% score and an 93.8% accuracy [1] as the score to match or beat. This score will be compared to the accuracy of the prediction and the overall score when discussing the results of the experiments.

III. METHODOLOGY

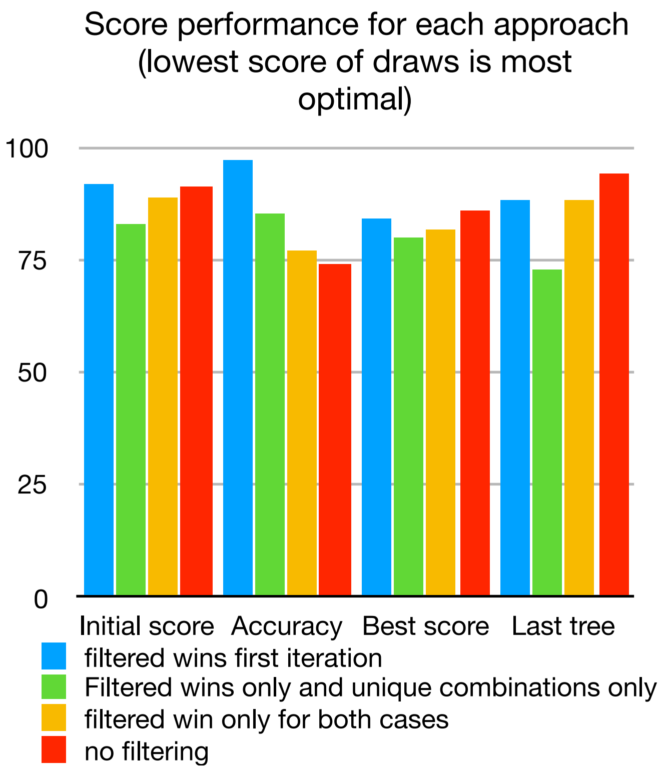
The data in the experiment was collected in a big batch of ten thousand games played in the first instance, as specified in the first paper [1]. This was trained with the original algorithm that explored random moves in order to find the best move. The MCTS algorithm was then modified to use the predictions from the decision tree created from the initial training epoch on the ten thousand games to predict the best move for ninety percent of the cases and ten percent of the time pick a random move. In the case where the DT choses a move that is already taken the algorithm picks a random move that is not taken instead for the sake of robustness. The best moves gleaned from the initial training are then used iteratively to train a new decision tree for five hundred games over ten epochs. In each epoch the last trained tree is used to predict the best moves over the five hundred games and the results from this process is used to fit or train a new decision tree to be used and tested upon a seventy thirty split of training data. This means that seventy percent of the data is used for training and thirty percent of the data is reserved for testing the data to avoid overfitting. This leads to a number of approaches to filtering the data to examine or observe different outcomes of using different modified data sets.

In order to attempt to find the best algorithm, a number of different approaches will be taken to analyse the effect on prediction based on just using the data from winning games or games where duplicate rows are or are not used. This is critical because there are many ways that the data can be filtered and then interpreted. In the end four different conditions were settled upon based on filtering out winning only conditions and the removal of duplicate entries to make sure that the algorithm either trains on winning only best moves and or moves that are not duplicated elsewhere. This leads to an analysis of the code.

In the MCTS2.py the main function calls read_csv to read the games that were computed in the original report with the original code from that report. There are then some commented out functions for filtering out losing games and duplicate rows, which can be commented in to get results for a combination of different approaches that are tackled in the experiments section of this report. The function run n iterations then does an initial scoring of the tree to score it against a tree constructed from the ten thousand games dataset. It then runs ten epochs and stores each of the trained trees on the way so that the best approach can be discovered. At the end of the epochs the scores can then be compared. The comparison of the experimental results is made possible by a set of modifications to the rollout stage of MCTS. This chooses for ninety percent of the time to use the decision tree to suggest the best moves for the MCTS algorithm to use. It constructs a view of the board and then uses the dectree (decision tree) to predict the best moves based upon past experience.

IV. EXPERIMENTS

A number of different experiments were performed for the task. The first is to filter out games that result in a loss and also combinations that are not unique (removing duplicates), the second involves filtering out won games for both new datasets and the original 10000 game dataset created, the third is to filter out games that are only filtered out for wins for the first dataset of 10000 games and not for datasets created thereafter, and the fourth is to run without filtering for won games or duplicate entries. The result is a set of tables that demonstrate the performance of each algorithm, giving an overview of the results for each approach. The results of which are summarised in the chart below for number of drawing games out of a hundred.



The chart shows the number of draws so that the overall number of wins can be tallied. Therefore the lower the number of draws, the more successful the approach is overall. In the graph the best overall improvement against the initial score (also pictured in the first column for each approach) is for decision trees selected for their accuracy after running consecutive training epochs. The largest gains in score were on average resulting from those epochs that resulted in the highest accuracy, with a score of 26% [Appendix IV]. The exception to this rule being the result set for datasets that are a part of win only conditions, filtered for duplicates that creates the highest percentage score of 27% [Appendix I].

However, the overall performance of the approaches or algorithms is sorely lacking at only a max of 27% or 26% score. This is demonstrated also when selecting trees that achieve the highest score as a fraction of a hundred games result in a winning scenario. For whatever reason retraining the trees is not resulting in a major increase for the majority of the cases. This is reflected in the scores for the majority of the trees that result in a best case scenario a small fraction out of five hundred games played. This is much smaller than the benchmark accuracy of 93.8%, and score of 92% respectively. It suggests that a different approach may be required in any future study to achieve greater score and a higher accuracy, but this will be discussed further in the next section.

V. Discussion

The data demonstrates that cherry picking trees based upon the maximum prediction accuracy yields results of more than double the win rate against the original score. The next highest score is achieved in the first case of removing duplicates and filtering for winning scores with the last tree in the list, but overall it could be said that selecting trees that provide the most accurate predictions results in the best results overall. The results of the comparison to Nunes et al's paper demonstrate that the state of the art performance is 92.9% score and 93.8% accuracy [1]. In the results from the experiment the best accuracy achieved was 88.8% for the unfiltered approach that achieved a score of 26%. In only a minority of games a negative score was achieved.

This suggests that the approach given cannot meet the state of the art achieved by Nunes et al. Moreover, the major caveat to this is the the results are different every time the algorithm is run, meaning that at best that can be achieved is a snapshot of the results for each epoch. For each snapshot the accuracy of the predictions was found to vary, but with a general trend towards increasing accuracy up to a point and then falling after the fact even by a few percent. It is not certain why the algorithm is behaving this way but a possible reason is that the decision tree is predicting values that are already taken. In the algorithm the

Although the experiment can be considered of mixed success due to the scores being fairly low in terms of success rate (27-26%) a lot was learned of the process to train decision trees, and the attempt to retrain trees with the results from previous trees. In the conclusion the results, and discussion are summed over to provide an overview of lessons learned and

VI. Conclusion

In conclusion no particular approach resulted in a high scoring result set (more than fifty percent score out of a hundred). This is despite the training iterations improving on the overall accuracy performance for best moves. This means that the approaches also fail to meet the standards set by Nunes et al in the original paper for a 92.9% score and a 93.8% accuracy [1]. The best scoring performance in terms of accuracy for best moves is that of 88.8%, which is good in terms of accuracy but doesn't hold a candle up to the 92.9% score set by Nunes et al [1]. It is suggested by the results that a different approach needs to be taken to achieve a better scoring in particular. It may or may not have something to do with the MCTS rollout not performing as expected, but at this point it is only purely open to speculation.

VII. REFERENCES

[1]P. Gent, "CE888 Assignment 1 - Decision Tree classifier with MCTS for noughts and crosses", 2019.

VIII. Appendix

Appendix I

game number	player 1 win	player 2 win	draw	Accuracy
1	2	10	488	0.2
2	19	56	425	0.375
3	15	62	423	0.282051
4	58	27	415	0.298507
5	23	4	473	0.28
6	9	64	427	0.236842
7	9	29	462	0.282608
8	14	70	416	0.207547
9	13	77	410	0.327868
10	8	25	467	0.272727
mean	17	42.4	440.6	

	player 1	player 2	draw
initial score	4	13	83
Accuracy tree selection	1	14	85
best score	8	12	80
last tree in list	7	20	73
accuracy on old data	0.4086…		
initial accuracy	0.3118279		

fig 1. Dataset filtered for wins only and unique combinations of values only.

Appendix II

game number	player 1 win	player 2 win	draw	accuracy
1	3	29	468	0.758620
2	7	65	428	0.743589
3	113	54	333	0.793791
4	2	119	379	0.880733
5	9	49	442	0.738853
6	1	96	403	0.812977
7	17	57	426	0.81
8	51	97	352	0.875
9	4	24	472	0.657894
10	1	70	429	0.880733
mean	20.8	66	413.2	

	player 1	player 2	draw
initial score	0	11	89
Accuracy tree selection	1	12	77
best score	1	17	82
last tree in list	0	12	88
accuracy on old data	0.588		
initial accuracy	0.7432306		

fig 2. Playing filtered for win only games in both the first and nth examples.

APPENDIX III

game number	player 1 win	player 2 win	draw	accuracy
1	0	35	465	0.877037
2	2	31	467	0.791851
3	6	84	410	0.827407
4	5	7	488	0.811851
5	1	10	489	0.838518
6	4	8	488	0.831851
7	0	19	481	0.86
8	165	24	311	0.814814
9	0	65	435	0.789629
10	0	20	480	0.848888
mean	18.3	30.3	451.4	

	player 1	player 2	draw
initial score	0	8	92
Accuracy tree selection	0	3	97
best score	0	16	84
last tree in list	0	12	88
initial accuracy	0.7429193		

fig 3. for only the first iteration filtered wins only and not thereafter.

APPENDIX IV

game number	player 1 win	player 2 win	draw	Accuracy
1	0	29	471	0.79185 …
2	18	159	323	0.811
3	6	60	434	0.76296 …
4	10	73	417	0.888…
5	0	80	420	0.8407407…
6	10	70	420	0.8088 …
7	0	12	488	0.8133 …
8	7	51	442	0.821481
9	0	6	494	0.8444 …
10	70	42	388	0.82444 …
mean	12.1	58.2	429.7	

	player 1	player 2	draw
initial score	0	9	91
Accuracy tree selection	24	2	74
best score	5	9	86
last tree in list	0	6	94
accuracy on old data	0.6377		

fig 4. Unfiltered for winning cases only and duplicates