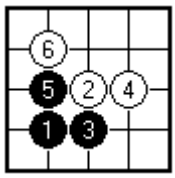
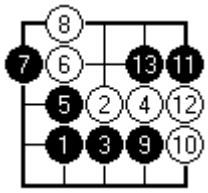
Final Project Proposal

In game theory, there is the concept of "solved" games that describes any game whose outcome can be correctly predicted from any board state--provided all players play rationally. Further, a "strongly solved" game is one in which the algorithm can make perfect moves even if the opponent makes mistakes (Wikipedia, 2021). In December 2002, researcher Erik van der Werf proved just this for Go, though he used the reduced 5x5 board instead of the full 19x19 board (van der Werf et. al., 2003). Yet, even in the almost 20 years since Dr. van der Werf released his Go playing algorithm called MIGOS (MIni GO Solver), the only improvement in solving this game came in 2015, when a group of researchers claimed that they had weakly solved the 7x7 board. Because of the apparent difficulty in finding these solved algorithms, we aim to discover if certain machine learning techniques can be used to extrapolate the optimal game play on a 5x5 to boards of a larger size--namely the 7x7 board.

To do this, we hope to be able to use Dr. van der Werf's MIGOS algorithm, whether he grants us the right to use his source code directly or we develop a version based on his paper, to create a complete data set of all possible board states and each state's corresponding optimal moves. For example, the following are possible board states:

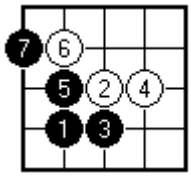
S1: 

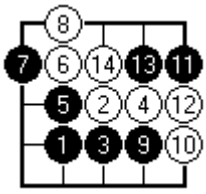
S2: 

Each of these would be represented by a two dimensional array as follows:

S1 = [0,0,0,0,0,0,1,0,0,0,0,2,1,1,0,0,2,2,0,0,0,0,0,0,0]
S2 = [0,1,0,0,0,2,1,0,2,2,0,2,1,1,1,0,2,2,2,1,0,0,0,0,0]

Where a 0 represents an empty cell, a 1 represents a white pawn, and a 2 represents a black pawn. The optimal moves for each state are as follows:

S1: 

S2: 

And would thus generate the labels:

y'(S1) = [0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
y'(S2) = [0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

Where the placement of the 1 in the label array denotes that the piece was placed there. Thus, it is easy to see that the input to our machine would be the board state and the output would be the predicted move.

From initial evaluations of optimal moves in a portion of the data set generated from the oracle algorithm using sklearn's dummy classifier, we found a baseline of 0.05 accuracy when using the "stratified" strategy, which shows that, if successful, our algorithm can drastically improve on baseline strategies. Since we can implement and train against the oracle algorithm as well, we can expect with confidence that this proposed algorithm will outperform the baseline.

```
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0]]
>>> dummy_clf = DummyClassifier(strategy="stratified")
>>> dummy_clf.fit(X,y)
DummyClassifier(strategy='stratified')
>>> dummy_clf.predict(X)
array([[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         1, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
         1, 0, 1],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         1, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0]]])
>>> dummy_clf.score(X, y)
0.05
>>>
```

Figure 1. Baseline code using scikitlearn

Evaluation for this algorithm on the 5x5 board would be quite easy, as we can simply take a part of the training data to be the testing data and find if the trained 5x5 algorithm can accurately predict the labels in the testing set. Further, we can test to see if the 5x5 algorithm will win against an imperfect player. Since the training set is based on a strongly solved game, we would assume that the trained algorithm would be able to win every game on a 5x5 board if it has actually learned the optimal strategy. However, evaluation becomes trickier for testing the trained 5x5 machine on the 7x7 board. Since we are unsure of how an optimal 7x7 Go-playing algorithm would function, the only metric that we can score our results against is how frequently it wins against a rational and an irrational player. If our trained algorithm wins a statistically significant number of games against a rational player, we will show that the 5x5 optimal algorithm is in some way related to the weakly solved 7x7 algorithm. Similarly, if our trained algorithm wins a statistically significant number of games against an irrational player, we will show that the 5x5 optimal algorithm is in some way related to the weakly solved 7x7 algorithm. Finally, if our trained algorithm always wins against one of these opponents, we will have discovered some insight into what an optimal 7x7 Go-playing may look like.

One of the challenges we may face in this is understanding how to use a machine learning algorithm trained on the 5x5 dataset (i.e. 25 inputs) in games on a 7x7 board (i.e. 49 inputs). To overcome this, we could convolve the 7x7 board into a 5x5 board using a kernel that would preserve the information in each cell equivalently. Further, since the 5x5 learning algorithm will only be trained on 5x5 board moves, it may not understand how to function in a 7x7 space, even after scaling or reducing the board. This may come to tuning the kernel to ensure it preserves the most important information in each cell. Another problem arises with learning how to adapt to an irrational player. Since it is infeasible to add all irrational opponent moves to our dataset, the controller would need to be general enough to handle those unexpected events. This problem could be helped with the use of Markov Decision Processes as it uses probabilities to decide what to do, which may make our algorithm more flexible.

Works Cited

Werf, Erik C. D. *AI Techniques for the Game of Go.* 2004. Maastricht University, Computer Science.

Werf, Erik C. D. van der et al. “Solving Go on Small Boards.” J. Int. Comput. Games Assoc. 26 (2003): 92-107.

Wikipedia. “Solved Game.” *Wikipedia*, Wikipedia, 19 February 2021, wikipedia.org. Accessed 6 March 2021.