Learning to Play like an Othello Master
CS 229 Project Report

Shir Aharon, Amanda Chang, Kent Koyanagi

December 13, 2013

1 Abstract

This project aims to train a machine to strategically play the game of Othello using machine learning. Prior to each match, the machine is trained with data sets obtained from professional matches. During each turn, the machine is given a list of its legal moves and selects the one that most closely reflects a move made by an Othello master.

2 Introduction

The game of Othello is a popular, two-player strategy board game in which players take turns placing their pieces on an 8-by-8 board. The game pieces are colored white on one side and black on the other, and each player is assigned one of the two colors. If a newly placed disk bounds the opponent's disks in a straight line, the bounded pieces are to be flipped, adding to the player's piece count. The game ends when all of the positions of the board are all occupied, and the player with the higher piece count is the winner.

Since the 8-by-8 version of the game has yet to be solved mathematically, Othello has been a topic of interest in the field of Artificial Intelligence. Typical Othello AIs use static evaluation functions to assign scores to different board configurations, which include properties such as the number of pieces each player owns or the positions of every piece on the board. These properties can be used as features in machine learning algorithms, which enables the machine to learn the combination of features that allows it to win with a higher probability.

3 Background

Due to the game's complex nature, it is difficult to create an Othello AI that is both fast and powerful. In general, Othello AIs strive to maximize a board score function while minimizing the number of possible moves the opponent can make. The reason is that a high score generated by a good score function correlates with a winning board position, and limiting the number of moves for the opponent forces them to make a bad move. Common implementation approaches that achieve this goal include a combination of alpha-beta pruning, minimax, and opening book strategies.

It is also possible to create a powerful Othello AI using machine learning. Rather than hard-coding features such as the board score, this approach allows the AI to recognize and make winning moves based on the data with which it was trained.

4 Training Set

The data set is taken from the World Othello Federations archive of championship matches for the years 2003-2008, which provides a total of 3840 moves made by Othello masters (58 games that each contains 60 moves). In the training set, these moves are considered as the "correct" moves while the available moves that were not chosen by the masters are considered as the "incorrect" moves. Since on average, there are 5-10 possible moves per turn, this creates a training set of over 26000 examples. From these examples, duplicate feature sets, i.e. situations that are identical under the current feature set have only one copy kept (marked as "correct" if any were "correct"). Additionally, as Othello has four rotational symmetries, each feature is repeated under each rotation. This is done to minimize the noise in the training set and to eliminate the favoring of any specific rotation.

Using this data set, two subsets are prepared. The first is the full set of data that contains the moves from all games, and the second is a reduced set that only contains the moves from one game ($\sim$417 examples before duplicates and symmetry). The smaller set was useful in initial debugging and testing, but also shows how little data is necessary to learn well by some learning models.

5 Methods

The following five feature sets are used to train machines using the algorithms discussed in 5.2.

5.1 Feature Sets

5.1.1 Feature Set # 1

The first feature set contains several simple properties of a player's turn: the turn number, the number of white pieces on the board, the number of black pieces on the board, the number of empty positions, the simple board score, and the complex board score. The number of white and black pieces are included because even though the greedy strategy is ineffective in Othello, these properties give a sense of which player is in the lead. The turn number is included because the playing strategy changes depending on the phase of the match; in general, an Othello match can be decomposed into three phases - early, middle, and late. The simple score is calculated using a board weighting that follows basic Othello strategies. Similarly, the complex score is calculated by accounting for special scenarios involving the board's corners and edges. Note that scores are always calculated relative to the current player (i.e. higher is supposedly better).

5.1.2 Feature Set # 2

This feature set is identical to the first feature set, but rather than tracking the number of white and black pieces, it tracks the number of the player's pieces and the opponent's pieces. These board properties are more useful because the AI could play either color.

5.1.3 Feature Set # 3

This feature set includes all features in the second feature set, along with a feature that tracks the state of all 64 positions on the board. By accounting for the state of every board position, this feature set aims to allow the AI to learn the special situations on its own.

5.1.4 Feature Set # 4

The fourth set of features is equivalent to the second feature set, but has the complex score split into multiple components for each special condition/location pair. It also adds the number of possible moves for the opponent.

5.1.5 Feature Set # 5

Feature set #5 contains the simple score, the multiple components of the complex score, and the number of valid moves. These are reasonable properties to use because they are the same properties used to implement basic, hardcoded Othello AIs (such as the one matched against the machine learning AIs).

5.2 Algorithms

The feature sets described above are used to train the machine using four learning algorithms: Naive Bayes, Decision Tree, Support Vector Machine (SVM), and Linear Regression.

5.2.1 Naive Bayes

The advantage of Naive Bayes is that it is simple to implement and does not require a large data set for training. In addition, its results are insensitive to irrelevant features, allowing for an iterative approach to find an effective feature set. The downside is that it assumes independence of features and that the margin of error is larger than that of more complex methods.

5.2.2 Decision Tree

The Decision Tree algorithm is robust, performing well even when the model does not follow the assumptions made by the algorithm.

5.2.3 SVM

SVM is well-suited for modeling complex data sets that contain many attributes. Its disadvantage, however, is that training is significantly slower than the other algorithms. Since the runtime is prohibitive for large data sets, it did not use the full data set for feature sets 3 and 4.

5.2.4 Linear Regression

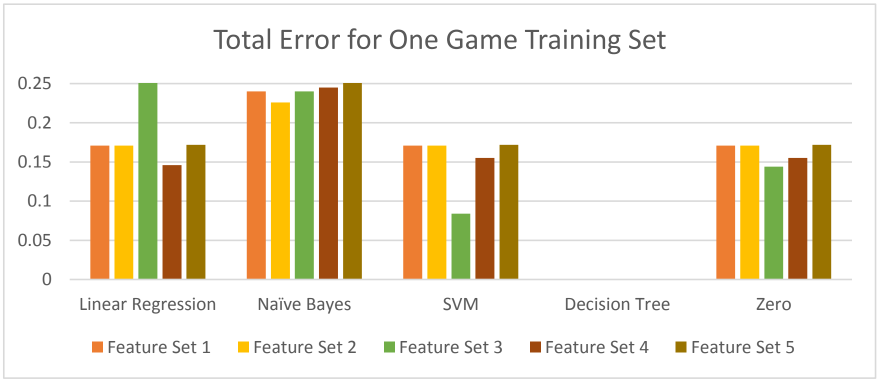
Conceptually, linear regression mimics the linear combination of individual scores that most hardcoded AIs use. It is both simple and the most intuitive of the considered algorithms.

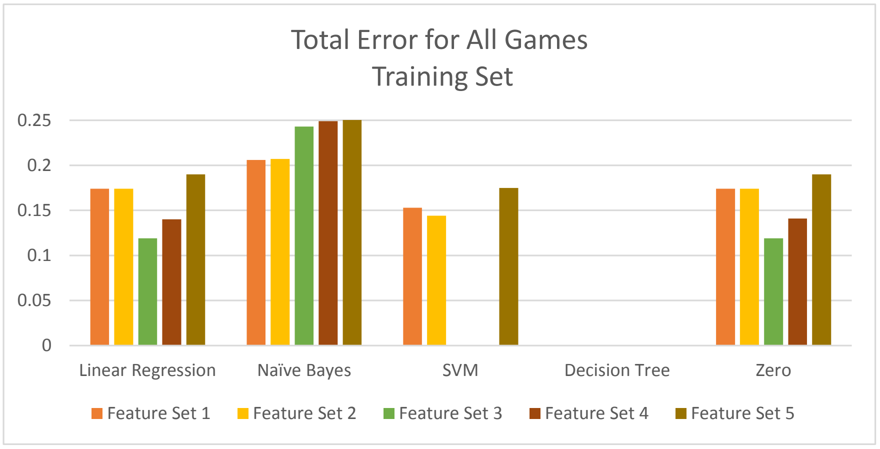
5.2.5 Linear Regression with Lookahead

This is the only algorithm that uses techniques from both machine learning and traditional Othello AIs. It uses both linear regression and a three-step lookahead to select a move out from the list of possible moves.

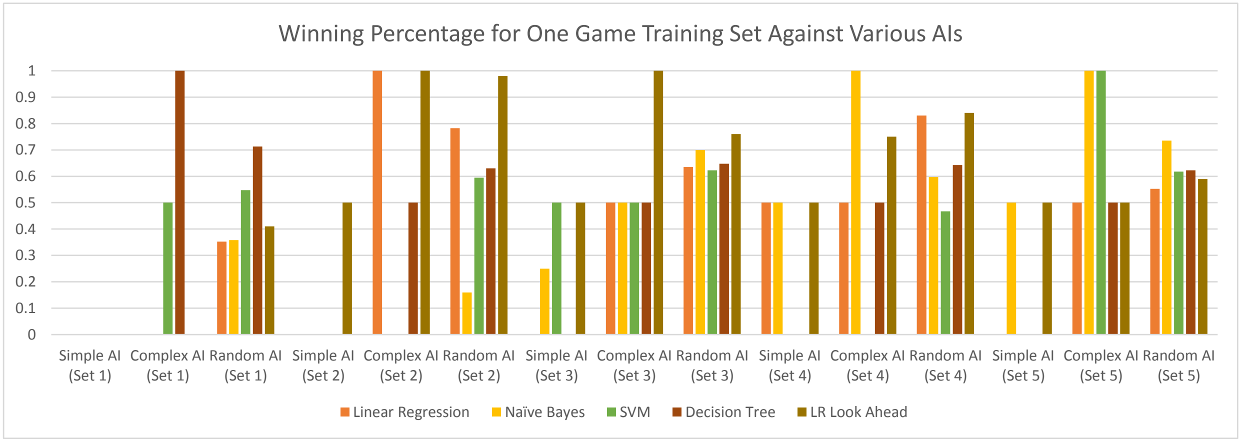
6 Results

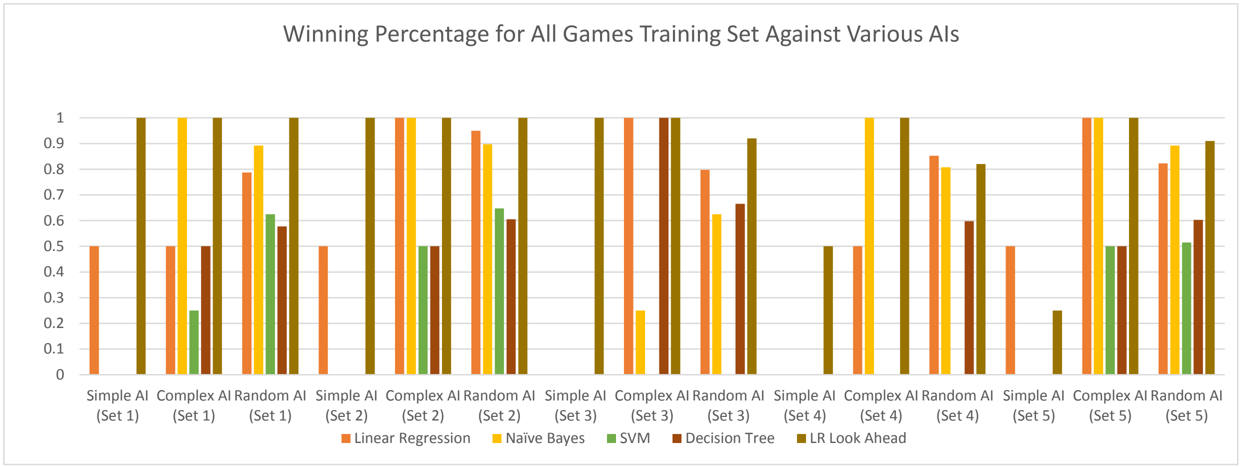
Figure 1 and Figure 2 show the total training errors, which are calculated using the k-fold cross-validation algorithm with $k = 10$.





Once the machine is trained on all combinations of learning algorithms, feature sets, and training set sizes, the effectiveness of each training is measured by matching the machine against three types of Othello AIs. The random AI randomly selects a valid move, while the simple and complex AIs select the valid move that generates the highest score using the corresponding weighting function (note that due to imperfect weights, the simple AI is generally better than the complex AI). This is illustrated by figures 3 and 4 below.





7 Analysis

Overall, all of the machine learning algorithms for all three feature sets have success rates greater than 50% against the Random AI. However, the learning algorithms do not perform well against the Simple AI. On the other hand, against the Complex AI, the results seem to vary depending on the selected feature set (note that due to suboptimal configuration, the Simple AI actually performs better than the Complex AI, contrary to expectations).

The machine that uses the Decision Tree algorithm learns almost perfectly as its errors are 0 for both training set sizes (to 3 digits). However, based on its poor performance against the hardcoded AIs, it appears that the algorithm suffers from overfitting. The machine trained using SVM also seems to suffer from overfitting, and is slow to training on larger feature sets.

Even though the Naive Bayes machine generally performs decently well against the hardcoded AIs, its errors are larger than those of the constant zero prediction, indicating that Othello does not follow the assumptions made by the model. Compared to Naive Bayes, Linear Regression has lower errors, but has the best performance of all of the algorithms. The inherent underfitting in Linear Regression prevents the machine from selecting a move that is specific to the situations that the Othello masters have encountered, but rather one that better reflects an optimal move.

Overall, regardless of the algorithm, the AIs that learned from moves made by the masters did not perform well against the hardcoded AIs. One approach to improve performance is to select a more appropriate feature set. Of the prepared feature sets, feature set #2 has the highest performance. This is most likely the result of including just enough features without introducing noise from the states of all board positions. However, even with feature set #2, the trained machine does not consistently win against the hardcoded AIs.

Another approach is to add more "correct" moves to the data set. By doing so, the training set encompasses a wider range of Othello game positions and the noise in learning specific situations is lessened. From an entropy point of view, each "correct" move has some information on ideal game play independent of the specific case, and lessens the "incorrect" moves that are good but less optimal.This is challenging because the size of the data set is limited and it is non-trivial to otherwise evaluate the moves.Given a large enough training set, feature set #3, which includes all board positions, should have allowed learning the game structure and resulted in an excellent AI, but was impractical.

The third approach is to vary the model depending on the phase of the game (early, middle, or late), as gameplay strategy differs significantly between the three phases. Feature sets #1-3 had the turn number as a feature, but this method forces the way in which the machine learning algorithm utilizes the turn number. It also limits applying learning from one phase to another (i.e. a high board score is likely to be good in all phases).

Overall, the above results may suggest that effective AIs cannot be implemented solely by mimicking moves made by the masters. On the other hand, combining methods from both machine learning and traditional Othello AIs may create powerful Othello AIs. In fact, the machine trained using Linear Regression with Lookahead on feature set #2 has a 100% winning rate against all three hardcoded AIs. It is also interesting to note that even after training on only one game's worth of data for feature set #2 (and to a lesser extent #4), linear regression plays incredibly well as indicated by its win rate against the random AI. This indicates that training on additional games adds very little useful information.

8 Future Work

Future work will primarily focus on improving the performance of the pure machine learning algorithms. As mentioned in 7, this will be done by adding relevant features to the feature set while removing extraneous ones, by adding more "correct" moves to the training samples, and by varying the model depending on the phase of the game. Ideally, these improvements will allow the trained machine to consistently win against the hardcoded AIs while maintaining low error and bias.

Additionally, the effectiveness of combining machine learning with traditional AI techniques will also be explored. For example, the Linear Regression with Lookahead can be improved by incorporating alpha/beta pruning for deeper lookahead. The machine learning component can be used to tune the parameters in evaluating board positions, but an accurate board evaluation as a single-step AI will still be extremely complicated.