Prediction of Chess Endgame using Decision Tree and SVM Classifiers

Anderson, Michael
CS
andermic@eecs.oregonstate.edu

Gutshall, Gregory
ECE
gutshalg@eecs.oregonstate.edu

Abstract

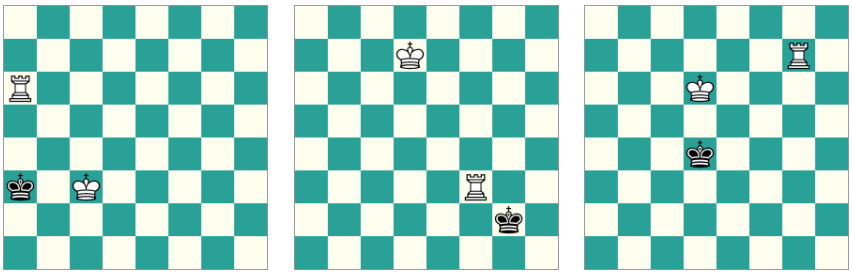
Traditionally, computer chess programs evaluate positions heuristically, by considering for each player factors such as the number of uncaptured pieces, king safety, central control, and number of possible moves available for both players. Chess endgame positions (positions with only a few pieces left) are often evaluated by starting from positions that are known to be won or drawn, and then working backward through a search tree. We would like to propose a third approach: formulate the problem of deciding the result of a chess endgame as a classification problem. Here we describe the application of both Decision Tree and Multi-Class SVM algorithms to a UCI dataset[1] that contains 28056 King and Rook vs. King endgame positions, along with the number of moves required by white to force checkmate in each position, when possible. We show that both Decision Tree and SVM are effective algorithms, although Decision Tree somewhat outperforms SVM. We further show that reparameterizing the dataset in terms of the spatial relationships of the pieces to each other, as well as to the edges of the board, improves the performance of our classifiers.

1 Introduction

1.1 Background\Problem Formulation

In this report, we discuss the problem of finding the number of moves required for white to beat black from an endgame position, with optimal play by both sides. In particular, we are interested in endgames where only three pieces remain on the board: a white king, a white rook, and a black king. A tried-and-true approach to this problem is to use a searching algorithm. The number of possible king and rook versus king checkmate positions is small enough that an algorithm can construct separate trees starting from all of them, and work backwards to any given king-rook king position. Using this method, along with minimax, it is then possible to determine the number of moves required by white to force checkmate with optimal play.

The issue with this approach is that search trees grow exponentially with their depth, and so minimax search requires a lot of computation time to look back more than a few moves. Here we consider another approach, that of using supervised learning to directly find the number of moves to checkmate. Instead of attempting to look forward to future positions or look back from a decisive position, we wish to look only at the current position itself. Here we test the hypothesis that simply the locations of the pieces in a given position provide enough information to determine the outcome of the game, and how far away that outcome is.



(a) Checkmate! (b) Free Rook (c) Free B_k

Figure 1: Example of some common Chess spatial features

In order to decide what sort of information is required to make this determination, we first need to understand the peculiarities of this problem. Figure(1.a) shows the characteristic King-Rook King checkmating pattern; the black king is on the edge, the white king directly opposes it from two squares away, and the white rook attacks the black king. It is important to notice that the black king cannot be put into checkmate in this type of endgame without being on an edge. Figure(1.b) shows a drawn position; black simply captures the white rook, because the white king is too far away to protect it. Figure(1.c) shows a position with the black king in the middle of the board; here the white rook and king must work together to force the black king to the edge, and then deliver checkmate. In the given position white should move his rook 3 squares down to check the black king and force it closer to the bottom edge of the board. The black king cannot stay in the middle of the board, because the white king controls those squares. White will be able to force a repeat of this pattern, until the pattern finally reappears with the black king on the edge of the board, which will be checkmate.

1.2 Outline of Report

In section(2) we describe the dataset from the UCI repository used for our training and testing environment. In section(3) we will give a common framework used for testing our algorithms and we will also provide insight into the parameterizations of the original data and why we think those parameterizations will yield improved classification results. In section(4) we will show results heuristically drawn from simulations for the two proposed methods and discuss the problems encountered to obtain these findings. Finally, in section(5) we will make final conclusions and possible algorithmic strategies to improve the results.

2 Dataset

2.1 Chess (King-Rook vs. King) Data Set

Our approach was tested on an old King-Rook King dataset from the UCI website[1]. This dataset contains 28056 data elements, and each element is a draw from the space of all possible King-Rook King endgame positions. The positions are uniquely described by 6 parameters: the row and column positions of each of the 3 pieces. Along with each position there is also one of 18 possible class labels $y_i \in [-1, 0, 1, 2, 3, \ldots, 16]$: either the position is drawn, the position is checkmate, or it will take white 1-16 moves to force checkmate with optimal play from both players.

We made a few small preprocessing modifications to this dataset to make it easier to work with. In the original dataset, rows are described using letters $(a \ldots h)$, and columns using numbers $(1 \ldots 8)$. We decided to make the rows numeric $(1 \ldots 8)$, so that we could easily do calculations with them later. Similarly, instead of having a "draw" label and a "checkmate" label, we used the numbers -1 and 0 respectively to represent these classes.

2.2 Properties

The distribution of true class labels across the dataset is not uniform, as shown by the left table:

Class	Count
Draw	2796
Checkmate	27
1	78
2	246
3	81
4	198
5	471
6	592
7	683
8	1433
9	1712
10	1985
11	2854
12	3597
13	4194
14	4553
15	2166
16	390
Total	28056

Relation	Mutual Information
I(Random, Random)	0.003
I(W_{k-row}, Class)	0.2903
I($W_{k-column}$, Class)	0.1653
I(W_{r-row}, Class)	0.0462
I($W_{r-column}$, Class)	0.0501
I(B_{k-row}, Class)	0.3145
I($B_{k-column}$, Class)	0.1851
H_{total}(Class)	3.502

This dataset has some other interesting properties, as shown by the right table of mutual information relationships between each of the parameters and the class label.

To establish a baseline, we calculated the mutual information between a vector of 28056 values drawn from $(1 \ldots 8)$ and another vector of 28056 random values drawn from $(1 \ldots 18)$. This told us that if there was no mutual information between a parameter and the class label, we would expect to see a mutual information of around 0.003 from random variance. Note that the mutual information between each parameter and the class label is significantly greater than this, indicating that each one would be at least somewhat helpful in determining the value of the class label.

The most interesting thing about this table is that it shows our data is somehow skewed, and not fully representative of the population of King-Rook King endgame positions. In a representative sample, the mutual information of the row of a piece should be roughly the same as the mutual information of a column, because the four-way symmetry of the chessboard along with the four-way symmetry of the movements of king and rook indicates that each possible position has exactly one corresponding position with rows and columns transposed. Here we see that the row of the kings is more important than the column of the respective kings, which means that a classifier trained on this dataset may not generalize to the space of all possible King-Rook King endgames.

Finally, to give these mutual information numbers more meaning, we calculated the total entropy of the Class labels as 3.502, which is an order of magnitude larger than even the best of these parameters.

3 Methodology

3.1 Algorithm Selection

Our specific approach to this problem was to try two algorithms: Decision Tree and SVM. The selection of the Decision Tree algorithm was motivated by its successful application in a similar problem that aimed to classify a dataset of King-Rook King-Knight endgames [2]. The selection of the SVM algorithm was motived by its reputation as a strong algorithm in many problem domains and our academic curiosity into multi-class learning with a spatial domain oriented learner.

3.2 Testing\Evaluation

In order to evaluate our classifiers, we needed to consider how to measure error. For classification problems, error is typically evaluated as the number of mis-predicted class labels made by the

classifier. However, in our problem this definition of error is somewhat uninformative, because our class labels are numeric and we want to capture how "close" our predicted label is to the true label. For instance, if the true class label is 14, then a prediction of 3 should constitute more error than a prediction of 15, because 3 is farther away from 14 than 15 is. Additionally, we hypothesized that it would be difficult for a classifier to distinguish between a position where forced checkmate is 14 moves away and a position where forced checkmate is 15 moves away. Taking these factors into consideration, we chose the following metric:

$$\begin{aligned} \mu_{error} &= \frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i| \\ \sigma_{error} &= \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}|\mu_{error} - \hat{y}_i|} \end{aligned} \tag{1}$$

This error metric captures average "closeness" of our predictions, and it does not unduly penalize incorrect predictions that are close to the true y. Besides our definition of error, we also needed to consider how to split up our dataset into training and testing data, since it was not neatly split as it has been in previous problems. Our approaches were slightly different for the two algorithms, and are described in their respective sections.

3.3 Re-parameterization

Based on our own intuition, along with previous work [2], we postulated that we would not be able to build strong classifiers based on the original parameterization of the data. Even though the original parameterization uniquely describes a chess position, we believed that other parameterizations would more neatly capture the information that would allow a classifier to distinguish the result of one of our endgame positions. In [2], instead of using the literal positions of the pieces as parameters, Quinlan uses a different set of 39 parameters that describe the spatial relationships of the pieces to one another, and the presence or absence of specific geometric configurations of the pieces. Indeed, this higher form of visual pattern recognition is more in line with how human chess players evaluate positions (along with trying to construct search trees in their heads, of course). Given this, and given what we know about the particulars of King-Rook King endgames, we were led to the following 5 parameter re-parameterizations:

1. L2 distance of black king to nearest corner: $\min \|B_k - corner_i\|^2$
2. L1 distance of black king to nearest edge: $\min \|B_k - edge_i\|^1$
3. L3 distance between kings: $\|B_k - W_k\|^3$
4. Draw on first move. $y_i = -1$
5. Checkmate on first move: $y_i = 0$

We observed that our parameters give a fairly high information gain relative to the original parameters. That notwithstanding, we postulated that our parameters would outperform the original parameters because the information in our parameters was more "non-overlapping", and so that information should require fewer splits in a decision tree (for example) to unlock, as shown below,

Relation	Mutual Information
I(B_k Corner-Distance, Class)	0.3328
I(B_k Edge-Distance, Class)	0.2602
I(W_k to B_k Distance L1, Class)	0.1068
I(W_k to B_k Distance L2, Class)	0.1488
I(W_k to B_k Distance L3, Class)	0.1546
I(W_k to B_k Distance L4, Class)	0.1546
I(W_k to B_k Distance L5, Class)	0.1546
I(W_k to B_k Distance L10, Class)	0.1546
I(W_k to B_k Distance L15, Class)	0.1546
I(W_k to B_k Distance L18, Class)	0.1546
I(W_k to B_k Distance L19, Class)	0.1546
I(W_k to B_k Distance L20, Class)	0.1541
I(W_k to B_k Distance L30, Class)	0.1522
I(W_k to B_k Distance L50, Class)	0.1344
I(W_k to B_k Distance L100, Class)	0.1264
I(W_k to B_k Distance LInf, Class)	0.1077
I(Draw/No Draw, Class)	0.4679
I(Checkmate/No Checkmate, Class)	0.0110

4 Simulation\Classification Results

4.1 Results: Decision Tree

For this set of experiments, we implemented a decision tree classifier that split using the information gain criterion. This included an implementation of entropy, mutual information, a trainer that builds a decision tree, and a classifier that takes a decision tree and a data element and uses the former to classify the latter. Although all of the values that any of the parameters could take were numeric, some testing showed that multi-way splits were superior to binary splits about a threshold. All of the experiments described below ran very quickly: on the order of seconds.

Our first experiment was to construct a decision tree using the original parameterization, trained on the entire dataset. Since this parameterization described each position uniquely, the decision tree had perfect accuracy in classifying our dataset. Unfortunately, this experiment created a decision tree with a whopping 27429 nodes. Since there were only 28056 elements in our dataset, this indicated that almost every split consisted of only a single data element being passed from one parent to a child, and all the rest being passed to another child. In other words, splits gave almost no information about the class label, and the tree was ridiculously overfitted to the data.

Our second experiment was to construct a decision tree using parameters from both the original parameterization, and our own 5 parameter parameterization, trained on the entire dataset. This experiment failed similarly, producing an overfitted tree with 27047 nodes.

Finally we constructed a decision tree using only the parameters in our re-parameterization, trained on the entire dataset. Note that since our parameters did not describe each position uniquely, it was not always possible for this algorithm to split until every single leaf contained only values from one class. Where leafs contained more than one class, classification of test data was done via majority vote. This experiment gave a much smaller and more reasonable model, a decision tree with 263 nodes. As a validation step, we randomly split the test data into 10 roughly equal chunks. We then performed 10 runs, in each run using a chunk as test data and the rest as training data. We originally intended to implement early stop or post-pruning to avoid overfitting issues, but this experiment demonstrated that overfitting was not a problem.

Method	μ error	σ error	Testing Accuracy
Decision Tree	1.063	1.478	85.0%

Here, testing accuracy is taken to be within a ϵ-ball away from the true class label. For this report, we used $\epsilon = 1$. Note, since this is a multi-class problem the testing accuracy expectation for random guessing across 16 class labels is $P_y = y_i * E(y)$, where $E(y)$ is the expectation from table(2.2).

4.2 Results: Support Vector Machine (SVM)

4.2.1 Pegasos

One method employed for this report was *Pegasos: Primal Estimated sub_GrAdient SOlver for SVM*[3]. This is a iterative shrinkage method that alternates between stochastic gradient descent steps and projection steps. This method does **not** improve accuracy over other SVM methods, but does drastically improve convergence time. This speedup is achievable by approximating the sub-gradient with k random samples of the training matrix and projecting $\mathbf{w}$ onto a L_2 ball of radius $1/\sqrt{\lambda}$, where λ and k are tunning parameters.

To start, the Pegasos algorithm[3] was visually tested against the chosen parameter space to see how separable the data was (see, fig(2)) and if the algorithm was working correctly.

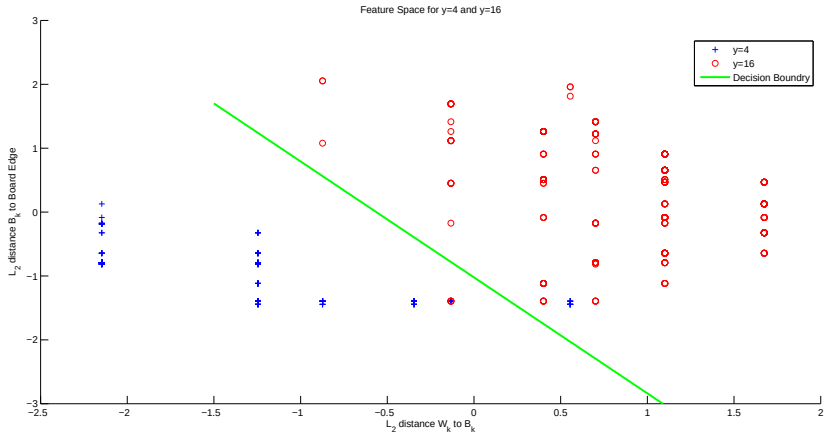


Figure 2: Feature Space Separation for two-parameters ($\|B_k - W_k\|^2$ and $\|B_k - corner\|^2$) for classes $y = 4$ and $y = 16$

With the algorithm working, the next goal is to setup the grid-search for tunning parameters λ and k. The projection distance is controlled by λ, while the subset of training examples for the sub-gradient is controlled by k, i.e. $A_k \subseteq X$. This grid-search had to be performed on each inner-class binary training problem.

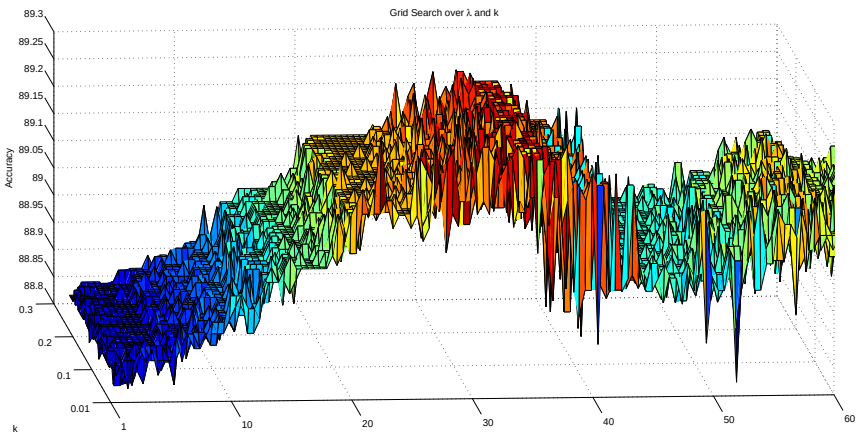


Figure 3: Grid-search over λ and k for a single binary classification instance

For Pegasos, we tried two methods for multi-class classification. The first, was *one-versus-the-rest*[4], where K separate SVMs are trained, with the k^{th} class assigned $+1$ and all other classes assigned to -1. Then the classification is provided by $y(\mathbf{x}) = \max_k y_k(\mathbf{x})$. This provided dismal results, since the decision boundary tended to split the feature space into separate halves, giving classes $\widehat{y} = 1$ and $\widehat{y} = 16$ the farthest distance to any decision boundary. The second method, was *one-versus-one*[4], where $K(K-1)$ separate SVMs are trained against each other. Then the classification is provided by $y(\mathbf{x}) = \max_k \rho_k y_k(\mathbf{x})$. Where, ρ_k is the number of "votes" for a given class y_k. The voting parameter $\rho_k = \sum_{i \neq I_k}(y_i = +1)$ was created by evaluating each row of the weight matrix, corresponding to a class label $\mathbf{y} \in [1, 2, \ldots, 16]$, such that $y_i = sign(\mathbf{W}_i^T \mathbf{x} + \mathbf{b}_i)$.

Now, it was observed from the dataset(2.1) that the number of examples for each class was vastly different. In this case, the SVM approximates a majority-class classifier and places the decision boundary extremely close to the minority class.

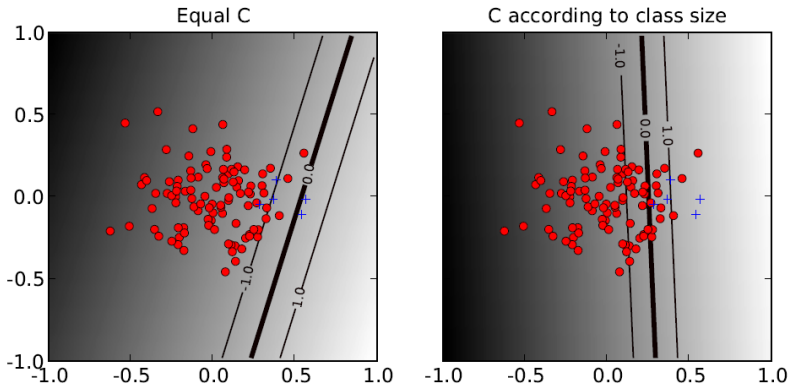


Figure 4: Effect of uneven number of class examples on the decision boundary (figure from [5])

To avoid majority-control, we can randomly sub-sample each class in $\mathbf{y}$ so that each minority class has the same amount of examples to train from. In this case, I sub-sampled a set of 60 examples from each class in $\mathbf{y}$.

The final multi-class prediction was found to be,

Method	μ **error**	σ **error**	**Testing Accuracy**
Pegasos: One-v-One	1.78	1.59	51.25%

4.2.2 LIBSVM

For Pegasos we used a linear kernel for the inner product of $\mathbf{w}^T \mathbf{x}_i$. We can expand this to a larger dimensional space by using a polynomial kernel $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$ of various power d, a radial basis function (Gaussian) kernel $K(x_i, x_j) = \exp(-\gamma \left\| x_i - x_j \right\|^2)$, or a sigmoid kernel $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$. The authors state that this is possible[3], but I was unable to complete the coding. So, all three non-linear kernels were attempted using the LIBSVM solver[6].

LIBSVM comes with its own methods for handling multi-class classification. To the end-user it is more of a "*black-box*", where $\mathbf{X}, \mathbf{y}$, and some [options] are supplied to the function verbatim. Again, using $\epsilon = 1$ for the testing accuracy we found the following results,

LIBSVM: Kernel	μ error	σ error	Testing Accuracy
Linear	1.29	1.42	68.28%
Polynomial d-2	1.75	1.93	59.9%
Polynomial d-3	1.32	1.42	66.7%
Polynomial d-4	1.60	1.71	61.82%
RBF	1.20	1.37	70.1%
Sigmoid	1.94	1.85	52.5%

5 Conclusions

We see from tables(4.1, 4.2.1, and 4.2.2) that the Decision Tree classifier outperforms SVM even with a non-linear kernel. This is because Decision Trees are naturally advantageous for this type of problem where mutual information may be exploited to distinguish classes, where as, the SVM is a spatial classifier seeking to maximize a margin in some domain. In addition, the Decision Tree has no major disadvantages between a binary or multi-class problem. However, SVM is natively a binary classification method, so a multi-class voting mechanism (such as that described in 4.2.1) needs to be employed.

This multi-class problem was unique in that we were attempting to classify a numeric feature as a label. So, we were able to use a heuristic for defining classification error (see, eq(1)). For all methods, we were able to classify the correct number of positions required for checkmate to occur within an average of 2 moves.

References

[1] Michael Bain, "Chess (king-rook vs. king) data set," UCI, Sydney 2052 Australia., 1994.

[2] JR Quinlan, "Induction of decision trees," *Machine Learning*, pp. 81–106, 1986.

[3] Nathan Srebro Shai Shalev-Shwartz, Yoram Singer, "Pegasos: Primal estimated sub-gradient solver for svm," *24th International Conference on Machine Learning (ICML)*, pp. 807–814, 2007.

[4] Vladimir Naumovich Vapnik, *Statistical Learning Theory*, Wiley, New York, NY, 1995.

[5] Asa Ben-Hur and Jason Weston, "A users guide to support vector machines," in *Data Mining Techniques for the Life Sciences*, vol. 609 of *Methods in Molecular Biology*, pp. 223–239. Humana Press, 2010.

[6] Chang Chih-Chung and Lin Chih-Jen, "Libsvm: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011.