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

Insert Abstract Text Here.

1 Introduction

1.1 Background\Problem Formulation

Discuss what a Chess Endgame is. Discuss how one would go about determining a Chess Endgame?

1.2 Outline of Report

In section(2) we describe the dataset from the UCI repository used for our training and testing environment. We also provide insight into the parameterizations of the original data and why we think those parameterizations will yield improved classification results. In section(3) we discuss the theory and limitations of the two proposed classification methods. In section(4) we will show results heuristically drawn from simulations for the two proposed methods and discuss results related to 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

Discuss the dataset from UCI[1]. Format of the data.

2.1.1 Notation

Chess Board Positions: Let the following notation describe the space of a Chess board,

$$\begin{aligned} \text{File} &\in [a, b, c, d, e, f, g, h] \\ &\in [1, 2, 3, 4, 5, 6, 7, 8] \\ \text{Rank} &\in [1, 2, 3, 4, 5, 6, 7, 8] \end{aligned} \tag{1}$$

Game Pieces: Let the three game pieces be defined as $Piece_i$, where $i = [1, 2, 3]$ represents the piece index,

$$\begin{aligned} Piece &\in [W_k, W_r, B_k] \\ W_k &= \text{White King} \\ W_r &= \text{White Rook} \\ B_k &= \text{Black King} \end{aligned} \tag{2}$$

Examples: Let a example of a game be defined as $Game_j$ where $j = [1, 2, 3, \dots]$ is the game row index,

$$Game_j = [Piece_i \{file_j, rank_j\}] \tag{3}$$

The entire set of examples is labeled as $\mathbf{X}$.

Class Labels: Class labels are defined as the remaining moves till checkmate of B_k. Note, checkmate of B_k is called on the m^{th} move of B_k.

$$\begin{aligned} y_j &\in [draw, 0, 1, 2, 3, 4, \dots, m] \\ &\in [-1, 0, 1, 2, 3, 4, \dots, m] \end{aligned} \tag{4}$$

The entire set of training class labels is labeled as $\mathbf{y}$.

2.2 Parameterization

Discuss how we parameterized the data.

Several parameter functions are used to classify or achieve an objective function, these parameter functions are labeled as $\mathbf{\Phi}(\mathbf{X})$.

3 Theory of Proposed Methods

3.1 Theory: Decision Trees

Theory goes here.

3.2 Theory: Support Vector Machines (SVM)

3.2.1 Binary SVM Classification

A support vector machine (SVM) attempts to draw a decision boundary between two classes, which results in the maximum margin[2]. Margin being defined as the orthogonal distance from the decision boundary to a subset of support points $\mathbf{x}_{support} \subset \mathbf{X}$. Since, we have freedom of normalizing the decision space we set this boundary to 1 and then attempt to solve the following quadratic optimization problem,

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right) \geq 1 \, , i = 1, \dots, N \end{aligned} \tag{5}$$

The above equation is useful for the linearly separable Hard-Margin case. Since the dataset(2.1) contains overlapping data points for different classes, we require a non-linearly separable SVM, i.e. we need to incorporate a Soft-Margin. This is achievable by relaxing the normalized constraint to allow support points to be within the margin[3]. The common approach, from Linear Programming, shows that adding slack variables $\boldsymbol{\xi}$ along with a trade-off parameter C can achieve this,

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \xi_i \\ \text{subject to} \quad & y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right) \geq 1 - \xi_i \, , \xi_i \geq 0 \, , i = 1, \dots, N \end{aligned} \tag{6}$$

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.

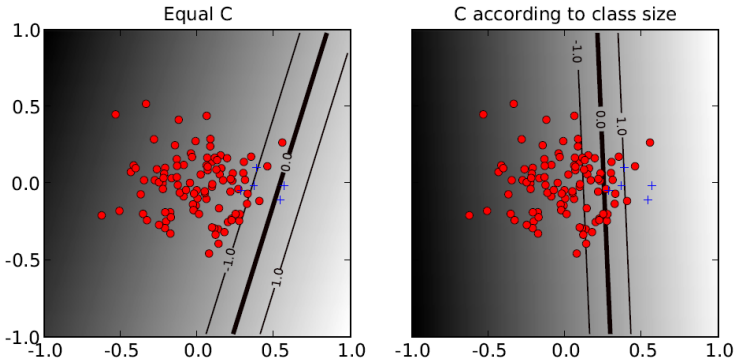

Figure 1: Using different trade-off parameters $\mathbf{C}$ to prevent majority-control (figure from [3])

To avoid majority-control, we can scale the trade-off parameters $\mathbf{C} \in [C_1, C_{-1}]$ so that the minority class caries more weight when misclassified. Such that eq(6) becomes,

$$\begin{aligned} \mathbf{C}\sum_{i=1}^{N}\xi_i &= C_1\sum_{i\in+}^{N}\xi_i + C_{-1}\sum_{i\in-}^{N}\xi_i \\ C_1 &= \frac{n_{-1}}{n_1}C_{-1} \end{aligned} \tag{7}$$

So far eq(6) uses 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 $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$. All three kernels were attempted using the LIBSVM solver[4] (see, sec(4.2)).

One method employed for this report was *Pegasos: Primal Estimated sub_GrAdient SOlver for SVM*[5]. 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.

3.2.2 Multi-Class SVM Classification

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.

For Pegasos, we tried two methods for multi-class classification. The first, was *one-versus-the-rest*[6], 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*[6], 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.

4 Simulation\Classification Results

4.1 Results: Decision Tree

4.2 Results: Support Vector Machine (SVM)

4.2.1 Pegasos

To start, the Pegasos algorithm[5] 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.

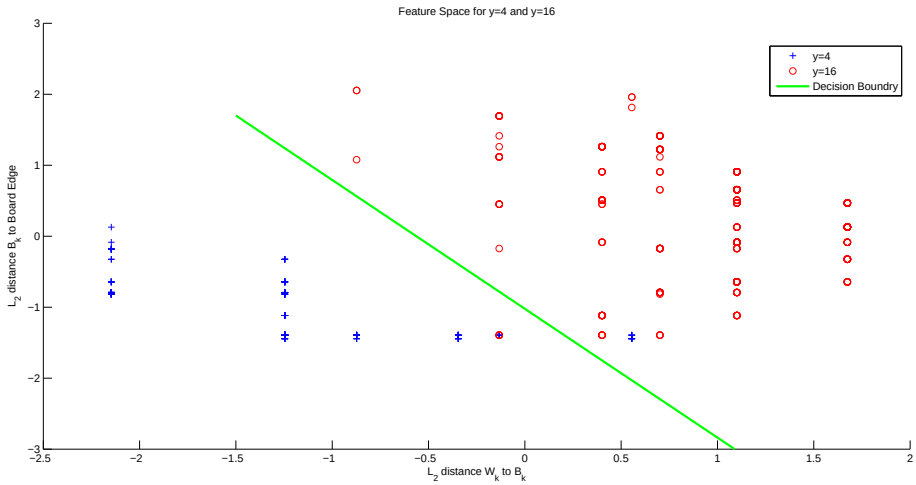

Figure 2: Feature Space Separation for two-parameters ($\|B_k - W_k\|^2$ and $\|B_k - corner\|^2$) for classes $y = 4$ and $y = 16$

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.

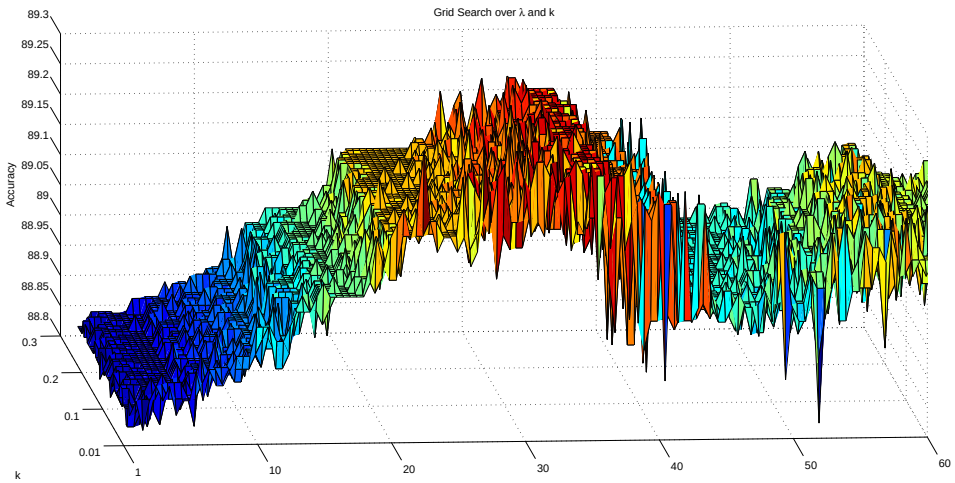

Figure 3: Grid-search over λ and k for a single binary classification instance

For the multi-class problem, we used the one-vs-one method with a voting system defined by $y(\mathbf{x}) = \max_k \rho_k y_k(\mathbf{x})$. 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 $y_i = sign(\mathbf{W}_i^T \mathbf{x} + \mathbf{b}_i)$. 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%

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 $P_y = 1/16 = 6.25\%$.

4.2.2 LIBSVM

As mentioned, LIBSVM[4] is a off-the-shelf implementation for SVM. We tried several different kernels with our dataset and found the following results, using $\epsilon = 1$ for the testing accuracy,

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

References

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

[2] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, NY 10013, 2006.

[3] 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.

[4] 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.

[5] 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.

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