Predicting Outcome of Chess Game with Statistical Learning

Harsha Kalidindi
University of Central Florida
MAP 4191 Fall 2021

1 Introduction

The history of machines in chess dates back to 1890, when Spanish scientist Torres Quevado introduced an electromagnetic device that was capable of checkmating a human opponent in a simple endgame. Since the 1950s, various aspects of the game have been studied as a subject of Artificial Intelligence (AI) and Machine Learning (ML). Silver et al. described the game as "the most widely-studied domain in the history of artificial intelligence". In 1997, IBM's supercomputer, Deep Blue, became the first chess AI to defeat a world chess champion (Kasparov). Today, modern chess theory is inseparable from AI and ML. The World Chess Championship is currently taking place, and Game 3 (Nov 28th, 2021), which ended in a draw, was described as the "most accurate world chess championship [game] ever played" - the played moves were almost identical to how the strongest chess engines would play.

Goal

Predict the outcome of chess games using the first 10 moves of a game ("Opening Phase" where material is usually equal) and the ratings of the two players.

2 Description of Data set and Algorithm

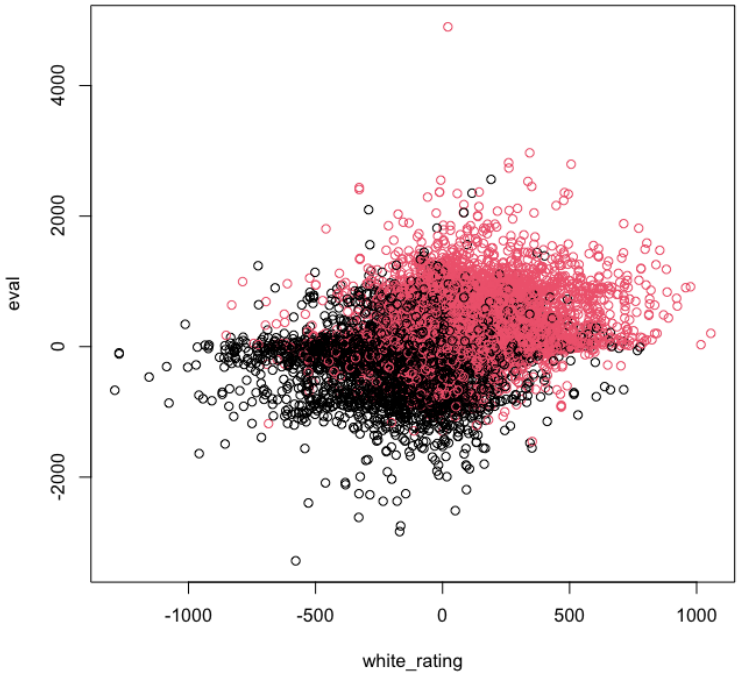
Dataset

Original dataset is a collection of over 20,000 games played on Lichess.org (Source: https://www.kaggle.com/datasnaek/chess):

id	rated	created_a	last_move	turns	victory_st	winner	increment	white_id	white_rat	black_id	black_rati	moves	opening_	opening_	opening_ply
TZJHLljE	FALSE	1.50E+12	1.50E+12	13	outoftime	white	15+2	bourgris	1500	a-00	1191	d4 d5 c4 c(	D10	Slav Defe	5

Only a subset of available features was used. Games with low increment and draws were removed, and rating difference was calculated as white_rating = white_rating - black_rating. Movelist was then ran through Python's chess library: First, board representation of position after 10 moves was created, then position was manually evaluated using SimpleEngine function. Finally, all features except rating difference, evaluation, and winner were removed. The final data set included 15122 observations with 3 variables:

winner	white_rat	eval
0	61	-1313



Algorithms

All models used rating difference (white_rating) and evaluation of position after 10 moves (eval) to predict winner. At first, only classification models covered in class were considered:

Logistic Regression

Logistic Regression uses logistic function:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

Then finds coefficients β_0, β_1 using *maximum likelihood:*

$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'})).$$

Coefficients are then used to predict class:

$$\hat{p}(X) = \frac{e^{\hat{\beta}_0+\hat{\beta}_1 X}}{1 + e^{\hat{\beta}_0+\hat{\beta}_1 X}}$$

Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) for p = 2 assumes $X = (X_1, X_2)$ is drawn from *multivariate Gaussian,* with a class-specific mean vector μ_k and a common covariance matrix Σ. The multivariate Gaussian density is defined as

$$f(x) = \frac{1}{(2\pi)^{p/2}|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \mathbf{\Sigma}^{-1}(x-\mu)\right)$$

The *Bayes classifier* then assigns an observation to the class for which

$$\delta_k(x) = x^T \mathbf{\Sigma}^{-1} \mu_k - \frac{1}{2} \mu_k^T \mathbf{\Sigma}^{-1} \mu_k + \log \pi_k$$

is largest.

K-Nearest Neighbors

K-Nearest Neighbors (KNN) identifies the K points in training data closest to some test observation x_0, represented by N_0, then estimates the conditional probability for class *j* as the fraction of points in N_0 whose response value equal *j:*

$$\Pr(Y = j | X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j)$$

KNN classifies the observation to the class with the largest probability from the above formula.

Support Vector Machines

Support Vector Machines (SVM) enlarge the feature space in a specific way, using kernels, then classifies data in a similar manner to the *Support Vector Classifier:*

$$\underset{\beta_0, \beta_{11}, \beta_{12}, \ldots, \beta_{p1}, \beta_{p2}, \epsilon_1, \ldots, \epsilon_n, M}{\text{maximize}} \quad M$$

$$\text{subject to } y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_{j1} x_{ij} + \sum_{j=1}^{p} \beta_{j2} x_{ij}^2 \right) \geq M(1 - \epsilon_i),$$

$$\sum_{i=1}^{n} \epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \sum_{j=1}^{p} \sum_{k=1}^{2} \beta_{jk}^2 = 1.$$

Classification Tree

Classification trees are decision trees that are created in a very similar way to *regression trees,* where *Gini index* is calculated as:

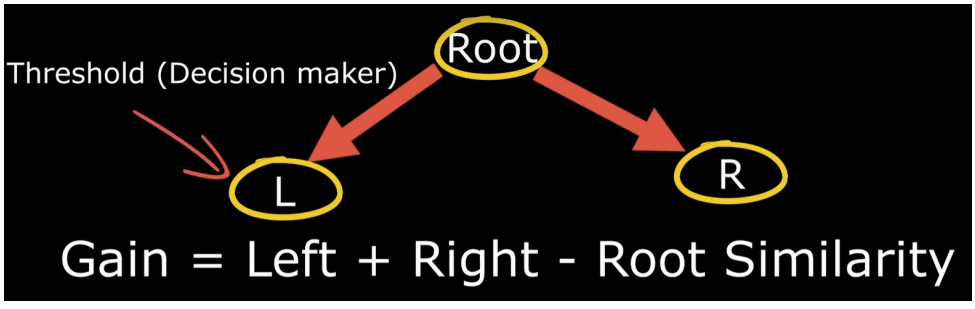
$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

XGBoost

XGBoost (eXtreme Gradient Boosted trees) algorithm was implemented, after research, in order to improve error rate. XGBoost is an ensemble boosting method which creates a decision tree, then passes one observation from validation set into the tree, where output gives initial probability. Initial probability is then used to calculate Similarity Score:

$$\frac{\sum(\text{Residual})^2}{\sum(\text{Previous Probability}(1 - \text{Previous Probability}) + \lambda)}$$

Where λ is regularization term which reduces similarity score - makes nodes easier to prune and helps prevent overfitting. Once similarity scores are calculated for each node in the tree, Gain Value is calculated for each branch, where branches with the highest Gain Value are kept



When one observation has a higher gain value than another, the higher gains observation's feature becomes the threshold of the tree. Once the tree has been filled with maximum gain nodes, the tree is then pruned with parameter term gamma (typically between 0 and 1), which is subtracted from the Gain Value of a node. A node is then pruned if the result is less than zero.

Another way XGBoost trims trees is by calculating Cover:

$$\text{Cover} = \sum \text{Previous Probability}(1 - \text{Previous Probability})$$

If the Calculated Cover is less than some threshold (default = 1), we prune the branch or node.

Next, output value of each node is calculated:

$$\frac{\sum(\text{Residual})}{\sum(\text{Previous Probability}(1 - \text{Previous Probability}) + \lambda)}$$

Output values are used to calculate output of log odds prediction:

$$\log(\text{odds})\ \text{Prediction} = \text{Initial Prediction} + \text{Learning Rate}(\text{Output})$$

Where learning rate (eta) is typically 0.3. Finally, using output of log odds prediction we calculate probability, which predicts class if less than 0.5:

$$\text{Probability} = \frac{e^{log(odds)}}{1 + e^{log(odds)}}$$

This process is repeated for N trees. Through N iterations the prior probability is always changing, which allows XGBoost to continuously learn from its mistakes.

3 Key results

Logistic Regression

Validation set approach gave following coefficients:

```
Coefficients:
  (Intercept)  white_rating          eval
    -0.073376      0.003903      0.001839
```

LDA

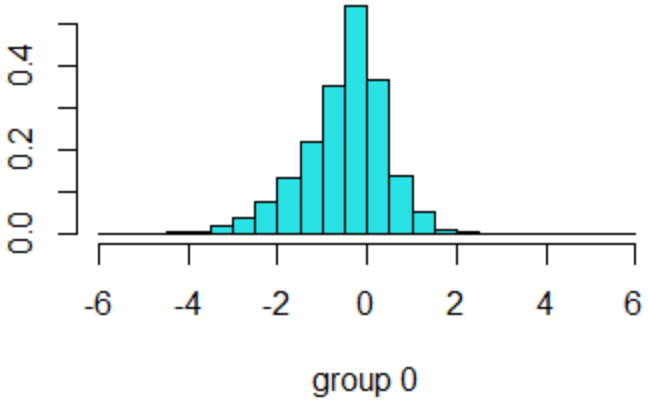
Validation set approach gave following coefficients, group means, and prior probabilities:

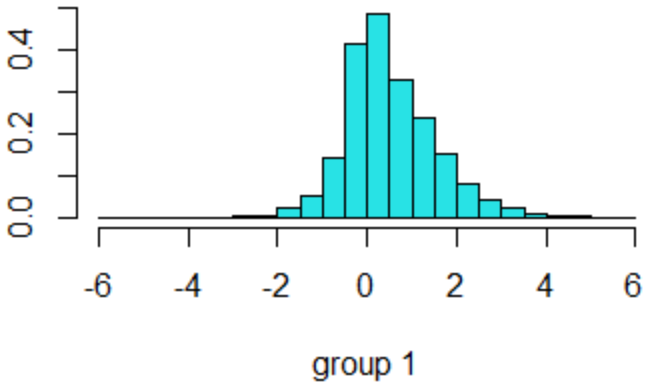
```
Prior probabilities of groups:
        0         1
0.4792346 0.5207654

Group means:
  white_rating      eval
0    -82.18068 -86.64729
1     82.64375 261.52828

Coefficients of linear discriminants:
                    LD1
white_rating 0.003245076
eval         0.001489108
```

Visual:





KNN

KNN was tested with values of K ranging from 1 through 450. In validation set approach, K = 100 gave lowest test error rate, while K = 450 performed better with 10-fold CV (30.1%)

Validation set (K=100):

```
   Ypred_knn
       0    1
  0 1230  617
  1  540 1394
> 1-mean(Ypred_knn == test$winner)
[1] 0.3060037
```

10-Fold CV (K=450):

```
[1] 0.3324521
[1] 0.2843915
[1] 0.2850529
[1] 0.3220899
[1] 0.2791005
[1] 0.3002646
[1] 0.3115079
[1] 0.3062169
[1] 0.2857143
[1] 0.3053536
> meanerr <- mean(cverr)
> print(meanerr)
[1] 0.3012144
```

SVM

SVM was tested with linear, polynomial, and radial kernels. Radial kernel performed best using validation set approach, with a resulting test error of 31.4%

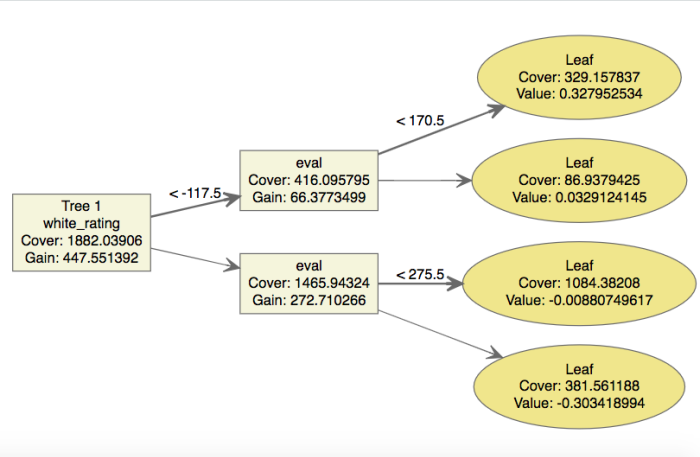
Classification Tree

Classification tree was created using tree() function in R. Classification tree performed worst, using validation set approach, with Error of 31.9%:

```
> 1-mean(tree.pred == test$bin)
[1] 0.3186988
```

XGBoost

Visualization of a decision tree obtained when tuning parameters for XGBoost:



Final results with error rates (misclassification) are outlined in table below. Dummy models were created for comparison: “Dummy model 1” and “Dummy model 2” classify data based on rating and evaluation only, respectively. Each Classification model outperformed Dummy models, with XGBoost yielding best performance (28.8% error rate). Final error rate was calculated using 10-fold Cross Validation for all models except XGBoost. 3-fold Cross Validation was used for XGBoost due to computational limits.

Model	Error (10-fold CV)
“Dummy model 1” - Rating only	36.5%
“Dummy model 2” - Evaluation only	34.9%
Logistic regression	30.8%
Linear Discriminant Analysis	30.7%
KNN (K=450)	30.1%
SVM	30.8%
Classification Tree	33.8%
XGBoost	28.8% (3-fold CV)

4 Conclusion

Using XGBoost, a 28.8% error rate was achieved, which was an improvement over other classification models and dummy models. Out of the models covered in this course, KNN with K=450 performed best. This may be due to the fact that the data is quite noisy, so high K value lead to smoother and less variable fit. 10-fold CV resulted in slightly different error rates than validation set approach.

Future Work

Outline of ideas for future work and improvement of model:

-Manually create decision tree, run individual moves through tree

-Run board representation of position through neural net

-Change rating difference to proportion

-Test model on larger data set

-Run model on certain time controls only

-Remove games between low rated players

-Run model on both white_rating and black_rating rather than use rating difference - may lead to differences in data at extremes

References

Code available on github:

https://github.com/harshak13/MAP4112

Python standard library:

https://docs.python.org/3/library/index.html

Textbook:

https://statlearning.com

XGBoost:

https://xgboost.readthedocs.io/en/stable/

History of Chess and AI:

https://www.britannica.com/topic/chess/Chess-and-artificial-intelligence