University of California
Los Angeles

Using AdaBoost to Implement Chinese Chess Evaluation Functions

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Statistics

By

Chuanqi Li

2008

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT OF THE THESIS

Using AdaBoost to Implement Chinese Chess

Evaluation Functions

By

Chuanqi Li

Master of Science in Statistics

University of California, Los Angeles, 2008

Professor Yingnian Wu, Chair

The paper describes a possible implementation of evaluation functions for Chinese chess using the AdaBoost algorithm - a machine learning method. It also describes implementations of evaluation functions (weak classifiers) in details. An experimental implementation is written in Java and its result is discussed.

1. Introduction

Computer Chinese Chess is an interesting research topic in computer science, mathematics and statistics. Many researches have been done mainly focusing on search algorithms, but this paper is focusing on machine learning methods. As Deep Blue successfully defeated top human master on European Chess, Computer Chinese Chess is expected to defeat top human master before 2010. And Deep Blue is based on search algorithms.

2. About Chinese Chess

2.1 History

Chinese chess (Xiangqi) is a very old board game in China and it has a long history. The earliest game mentioned by historical documents can be dated back to 4th BC, a game played between TianWen and Lord MengWhang, but it was probably invented even long before 4th BC. It is believed that Chinese chess was used to model wars and predict the results at that time. The early Chinese chess took very different forms from the modern one. It kept changing as Chinese history evolves. Significant changes can be identified during the Spring and Autumn Period, Tang Dynasty, Song Dynasty and Ming Dynasty.

Modern Chinese chess appeared in the Ming Dynasty (14th century), and become very popular since then. The board setting and pieces remain unchanged till now. Couples of chess books and manuals were written during the period. Chess books and

manuals such as JuZhongMi, MenRuShenJi and BaiBanXianqiPu contributed significantly to modern openings and end games. Some of them include game puzzles, which serve the same functions as word puzzles in many newspapers.

Modern Chinese chess are popular in Asia, European and even in America. Each year there are hundreds of official contests held in the areas. Those games draw a lot of attention from the local people, especially in China. And thus in a grading system similar to the European chess exists. The best player in China in 2006 is grand chess master XuYinChuan with a ranking of 2628. Grand chess masters are considered to be above 2500. Average levels are considered to be around 1500.

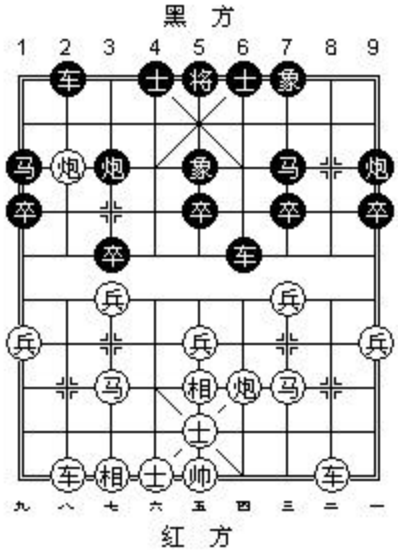

Figure1: modern Chinese chess: a game played by grand masters XuYinChuan and HuYongHua in the 20th *Five Goats Cup*

2.2 Computer Chinese Chess

Computer Chinese chess is a part of artificial intelligence research. It is a joint field of computer science, mathematics and statistics. The first generation of chess

programs focusing mainly on search algorithms, which are more computer science related. Brute force search is a major characteristic. Thus the concepts of game trees, Alpha-Beta pruning are the important concepts of the research. As statistics playing more and more important role in artificial intelligence, computer chess programs focus more on applications of statistics theories. Thus the second generation of computer chess are featured with machine learning, automatic improving and more statistics related techniques.

As new theories and technologies such as AdaBoost and Support Vector Machine emerge, new machine learning methods are to be considered into implementation of the new generation of computer chess. Nowadays the popular use of Internet where public online Chinese chess games can be easily found provides a fruitful resource of data samples, and they can be automatically collected easily.

Currently top computer Chinese chess programs are around 2500. With the great success of Deep Blue, computer Chinese chess is expected to defeat top human players by the year 2010. Chinese chess is considered to be a little more complex than European chess, while Go is the toughest one. Its complexity comparing with European chess and Go is listed in Figure2.

Game	**State-space complexity**	**Game-tree complexity**
European Chess	50	123
Chinese Chess	48	150
Go	171	360

Figure2: complexity of three games (The numbers are as log to base 10)

3. Defining the Problem

Chinese chess is a two-player, zero-sum game with complete information. A chess game can be represented as a game tree with each node representing a game situation. A current node represent the current situation, and the children of the current node are all the possible situations from the current situation, when taking the corresponding moves. Figure3 is an example of a game tree.

To find the optimal move for each game situation, a complete search of the whole game tree is required. Each leave in the game tree represents the final situation. It is a win, loss or a draw. The average of legal moves per step is about 19 and so if a game is about 60 steps then 19^{60} situation need to be searched. Obviously such a space to search is out of the reach of any computer nowadays. Even though an Alpha-Beta pruning algorithm is implemented, the space to search is estimated to be 19^{30}, which is still too huge.

The solution suggested by Shannon's 1949 paper is not to reach the leaves when the search is infeasible, but to estimate the situation at some depth. And this is where evaluation functions come from. Traditionally an evaluation or a heuristic evaluation function for Chinese chess is defined to be a probability function of a certain situation to win, lose and draw for a player at a node.

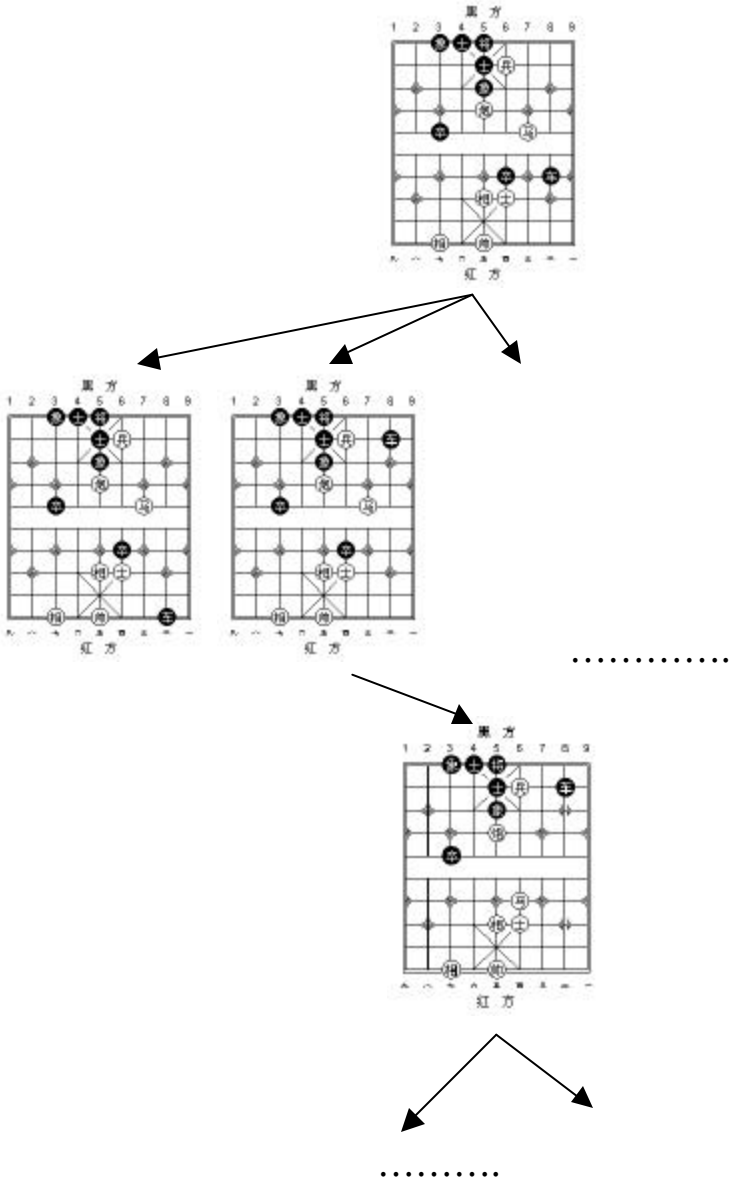

Figure3: This game tree represents a current game situation (the root node) and its possible game situations of the next steps. The dots represent the all the other nodes not shown here but they actually exist. The arrow represents the corresponding move that leads to the node.

4. Statistics Behind Evaluation Functions

If the result of a game is viewed as a random variable G (G can take values *win*, *loss* or *draw*), and state *s* is a game situation of a dimension 32 vector where $s \in S$ (*S* is about size of 10^{48}; dimension 32 comes from the fact that there are 32 pieces in total), then *P(G|S)* is the probability of a game to find.

Assumption1: assume there are *n* evaluation functions $H_1(s), H_2(s), \ldots H_n(s)$ such that *P(G|S)* is closed enough to $P(G| H_1(S), H_2(S), \ldots H_n(S))$ for all $s \in S$ in the sense of

$$e = \sum_{s \in S} | P(G | s) - P(G | H_1(s), H_2(s)..H_n(s)) |^2$$

is small enough.

Assumption1 actually assumes there is a mapping that can map the high dimensional *s* into a lower dimensional *H* (where $H = <H_1, H_2, \ldots H_n>$ is a vector) and only a little information is lost regarding to the probability measure. This is why evaluation functions will work and *P(G|H)* can approximate *P(G|S)* .

By Bayesian's theorem

$$P(G | H_1(s), H_2(s)...H_n(s)) = \frac{P(G)P(H_1(s), H_2(s)..H_n(s) | G)}{P(H_1(s), H_2(s)...H_n(s))}$$. Thus $P(H_1(s), H_2(s), \ldots H_n(s))$ is a normalizing constant and *P(G)* can be sampled easily. $P(H_1(s), H_2(s), \ldots H_n(s)|G)$ can be learned by sampling a huge amount of labeled samples. Today with the popular use of Internet and public online games, sampling such a game space is possible, but there is still a problem: storing the function into a computer is infeasible. If storing the function requires an array of size m_1 by m_2 by $m_3 \ldots m_n$ (where *m* is total possible value of a *H*), then the product of *m* increases in $O(N^2)$ that it easily reach the

size of the game space 10^{48}, which is no different from storing the whole game space! Thus comes the second assumption.

Assumption2: assume there are many independent evaluations functions *H* can be found with assumption1 still holds.

If those independent functions are found, then $P(H_1(s), H_2(s), \ldots H_n(s)|G)$ becomes $P(H_1(s)|G)\ P(H_2(s)|G)\ldots P(H_m(s)|G)\ P(H_m(s), \ldots H_n(s)|G)$. The last term is a joint distribution for which H are dependent. The other terms are independent. With a hope of the last term has small numbers of *H*, the space need to store such a function is about the sum of the *m* adding the product of the last few *m*, and it is feasible for modern computers. Furthermore the last term of the joint distribution function can be represented by a Bayesian network if it is resolved into conditional distributions so that the required space can be even less and training can be easier.

To summarize

In order to find good evaluation functions, two conditions need to meet: *least information loss and least dependency*. The conditions are not only useful for human but also for computers if they are used to search evaluation functions automatically.

If a *log* is taken on the product of the likelihood $P(H_1(s)|G)\ P(H_2(s)|G)\ldots P(H_m(s)|G)\ P(H_m(s), \ldots H_n(s)|G)$, it becomes a linear combination of terms. And that is the reason why many modern computer chess programs take a form of a linear combination as their evaluation function.

5. Implementations of Evaluation Functions

This section will discuss some real implementations in details. In Chinese chess theory, evaluation of a game situation is divided into four parts: pieces' value evaluation, position evaluation, flexibility evaluation and relationship evaluation.

In a mathematical and statistical way, the division is to divide raw data into four dimensions for better data handling and description. If each evaluation result of the four parts is seen as a random variable, the division makes the covariance or dependence between any two of the four variables as little as possible as the previous section discussed.

5.1 Piece Value Evaluation

This is the simplest and widely used method to evaluate a game situation, yet it is a very important one. In a certain point of view, the goal of the game is to maximize the value of the player's pieces and minimize his opponent's. The King has an infinite value, or greater than the sum of all pieces. When either player's King is captured, the game is ended, that is one player successfully minimized its opponent's value.

Other piece' values affect the game differently. Generally speaking, loss of a Rook can be considered as definite loss. Loss of a horse or a cannon can be considered as likely loss of the game.

Pieces:	**King**	**Guard**	**Elephant**	**Rook**	**Horse**	**Cannon**	**Pawn**
Value:	6000	120	120	600	270	285	30

Figure4: piece values used by ELP Chinese chess program

Pieces:	**King**	**Guard**	**Elephant**	**Rook**	**Horse**	**Cannon**	**Pawn**
Value:	6000	133	166	600	266	300	66

Figure5: piece values suggested by LiuDianZhong

Pieces:	**King**	**Guard**	**Elephant**	**Rook**	**Horse**	**Cannon**	**Pawn**
Value:	6000	120	120	600	300	300	60

Figure6: piece values used by XieXieMaster Chinese chess program

The values are summarized from millions of game experience since modern Chinese chess. People agree that a Rook roughly equals to a horse and a cannon and so on, but the exact values are still controversial. Figure3 is a list of piece values used by ELP Chinese chess program. Figure5 is a list of piece values suggested by grand chess master LiuDianZhong. Figure6 is a list of piece values used by XieXieMaster Chinese chess program. The lists are multiplied by linear factors that they all have the same value of a Rook and they can compare other pieces to other lists.

Similar to Shannon's 1949 paper, the evaluation for this type of evaluation is defined as

$$T = T_{red} - T_{black}$$

Where

$$T_{black} = number\_Guards * 133 + number\_of\_Elephants * 166 + number\_of\_Rooks * 600 + number\_of\_Horses * 266 + number\_of\_Cannon * 300 + number\_of\_pawns * 66$$

$$T_{red} = number\_Guards * 133 + number\_of\_Elephants * 166 + number\_of\_Rooks * 600 + number\_of\_Horses * 266 + number\_of\_Cannon * 300 + number\_of\_Pawns * 66$$

Red should try to maximize T as large as possible while the black should try to minimize T as small as possible.

Figure7 is a standard draw game in the fact that black has only an attacking piece Rook that is of value 600, while red has a horse and a cannon in the amount of 555. T is -45 by definition, and so the piece value difference is not too huge and thus it leads to a draw.

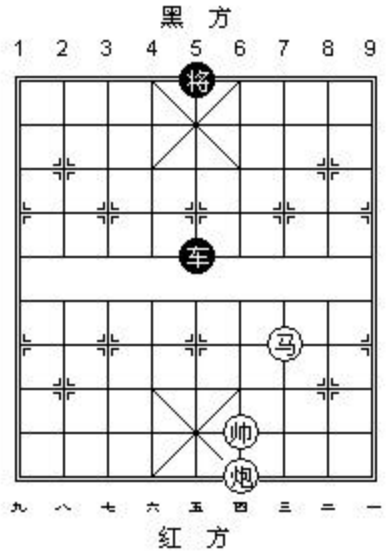

Figure7: a standard draw game.

5.2 Position Evaluation

Every piece has its own important positions. Figure5 is the important positions for a Rook used by chess program ELP. Generally speaking the important positions for a Rook are the positions that are close enough to enemy's palace, easy to defense (both

riversides and self-pawn row) and easy to attack (enemy's pawn row). Other pieces can be evaluated in a similar way.

```
 14 14 12 18 16 18 12 14 14
16 20 18 24 26 24 18 20 16
12 12 12 18 18 18 12 12 12
12 18 16 22 22 22 16 18 12
12 14 12 18 18 18 12 14 12
12 16 14 20 20 20 14 16 12
 6 10  8 14 14 14  8 10  6
 4  8  6 14 12 14  6  8  4
 8  4  8 16  8 16  8  4  8
-2 10  6 14 12 14  6 10 –2
```

Figure8: Position values of a Rook used by ELP Chinese chess program

The evaluation function for this type is a table recording the values indexing by positions. Figure8 is an actual example implementation.

The positional effects are much more obvious on Pawns than any other pieces, since Pawns can move horizontally once across the river. According to LiuDianZhong's *Chinese Chess for Beginners*, two Pawns together across the river have value slightly less than a Horse, but three Pawns together will be greater than a Horse. Figure9 and figure10 show positional examples of extreme cases.

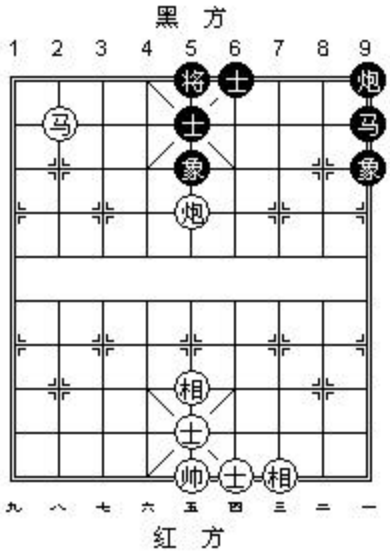

Figure9: an extreme situation where red is wining since red pieces have much better positions than black's.

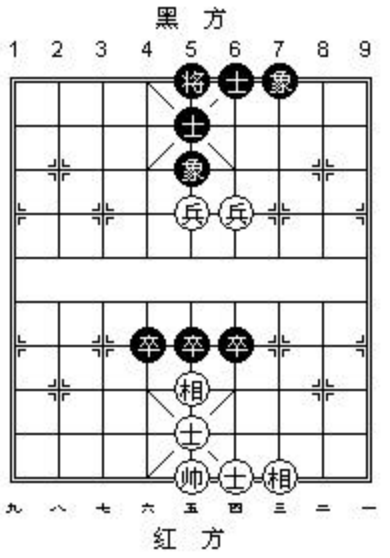

Figure10: two Pawns across the river and three Pawns across the river.

5.3 Flexibility Evaluation

Flexibility of a chess piece directly relates to its attacking and defending power. If a piece's value is considered as static, then the flexibility of the piece reflects its dynamic value. This is more obvious for a Horse or an Elephant than a pawn, since a Horse or an Elephant cannot move to a direction when there is an obstacle before it. (See appendix for more details of the chess rule).

A simple formula for flexibility evaluation is:

$$D = S * total\_crosses\_controlled / total\_possible\_crosses\_to\_controll * b$$

Where S is the static value from its piece value evaluation. For a Rook S equals to 600. b is a linear coefficient used to adjust D. As the previous method, there are many possible implementation of the method. For example, b can take many different values or D can be a function of polynomials degree two.

Figure11 is a case when flexibility determines the result of the game. Note both players have same pieces and only one piece have a different position.

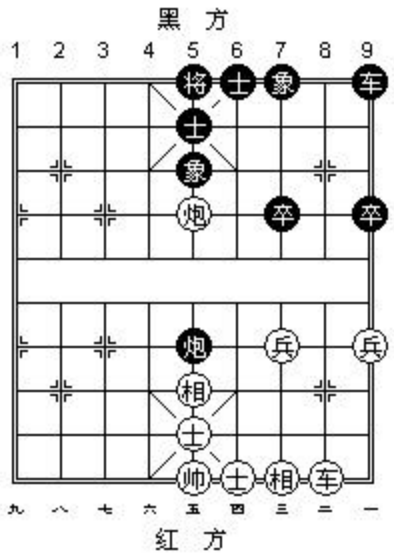

Figure11: red is winning due to the red Rook has better flexibility

A Horse has greater value than a Cannon in end games due to fewer obstacles than opening games. So there is such a chess saying that “A Horse is better than a Cannon in end games, and A Cannon should go to the palace to defend in end games”.

5.4 Relationship Evaluation

The previous three methods of evaluation are all focusing on a single piece and they do not evaluate the relationships between two or more pieces, pieces and positions, pieces and game stages. When two or more pieces appear, there are combinational effects. It is similar to linear regression, when two or more predictors appear, there may be combinational effects.

5.4.1 Relationship Between Pieces Values

Thus three kinds of relationships exist: piece value and game stage (opening, middle game, and endgame), piece value and piece value, positions and game stage, positions and positions.

From figure12, the value of two cannons has greater value than the sum of one cannon and one horse. Two cannons have the value of 580 from Figure 1 and one cannon and one horse have the value of 555. Actually one cannon and one horse have greater value than two cannons, because in practice cooperative relationship between one cannon and one horse has greater value than two cannons. That is, more tactical skills can be applied to one cannon and one horse. One of the tactical skills is call *a cannon behind a horse.*

A simple evaluation function of the type can be defined as:

$Px = 1 * d$, if x exists, otherwise 0. Where d is a linear factor to adjust P, *and* x is the combinational effect. A concrete example is $P_{cannon\_horse\_combinational\_effect} = 1 * d$, if at least one horse and one cannon exist, otherwise 0. And other combinational effects can be calculated in a similar way.

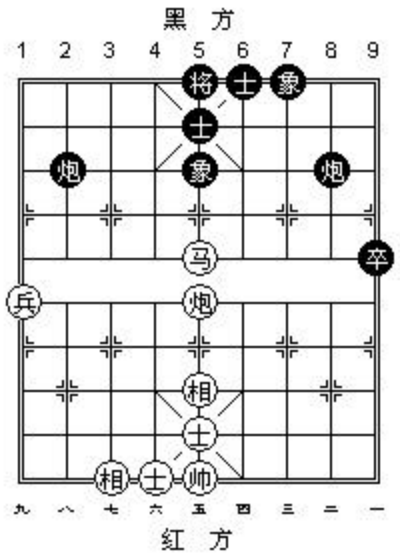

Figure12: red is better than black due to horse-cannon has better cooperative relationship

5.4.2 Relationship Between Pieces and Game Stages

A piece value varies during game stages (opening, middle and end game). A pawn has much larger value in an end game than its value in an opening game due large value pieces (Rooks, Horses, Cannons) captured in a middle game. In many master games one more pawn makes a difference, and it often leads to a winning game. A Horse has a smaller value than a Cannon in an opening game due to too many obstacles, but it has larger value than a Cannon's due to many of the obstacles removed. This is another example of a piece-stage combinational effect.

This type of evaluation function can be defined as:

$S_{pawn\_stage\_combinational\_effect} = number\_of\_Pawns * c$, if the game stage is an end game, otherwise 0, where c is a linear factor to adjust S.

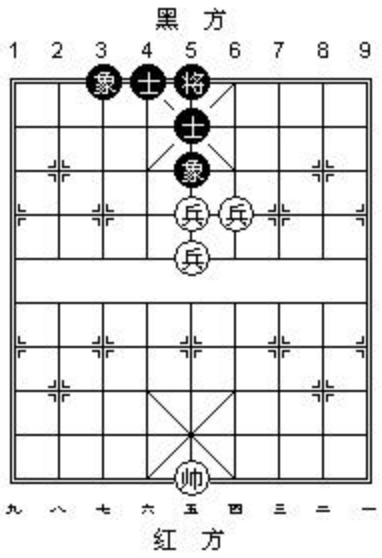

Figure13: red is winning due to the three more pawns.

5.4.3 Relationship Between Pieces

Formations play a major role in the piece relationship evaluation. Chess books spend most of their pages talking about formations, from opening games to middle games and from middle games to end games. People categorize the formations by pieces types. Figure11 shows six types of car-horse formation: palace-corner Horse (with a Rook), fishing Horse (with a Rook), WoChao Horse (with a Rook), willow-fish Horse (with a Rook), tiger-Horse (with a Rook), high Horse (with a Rook).

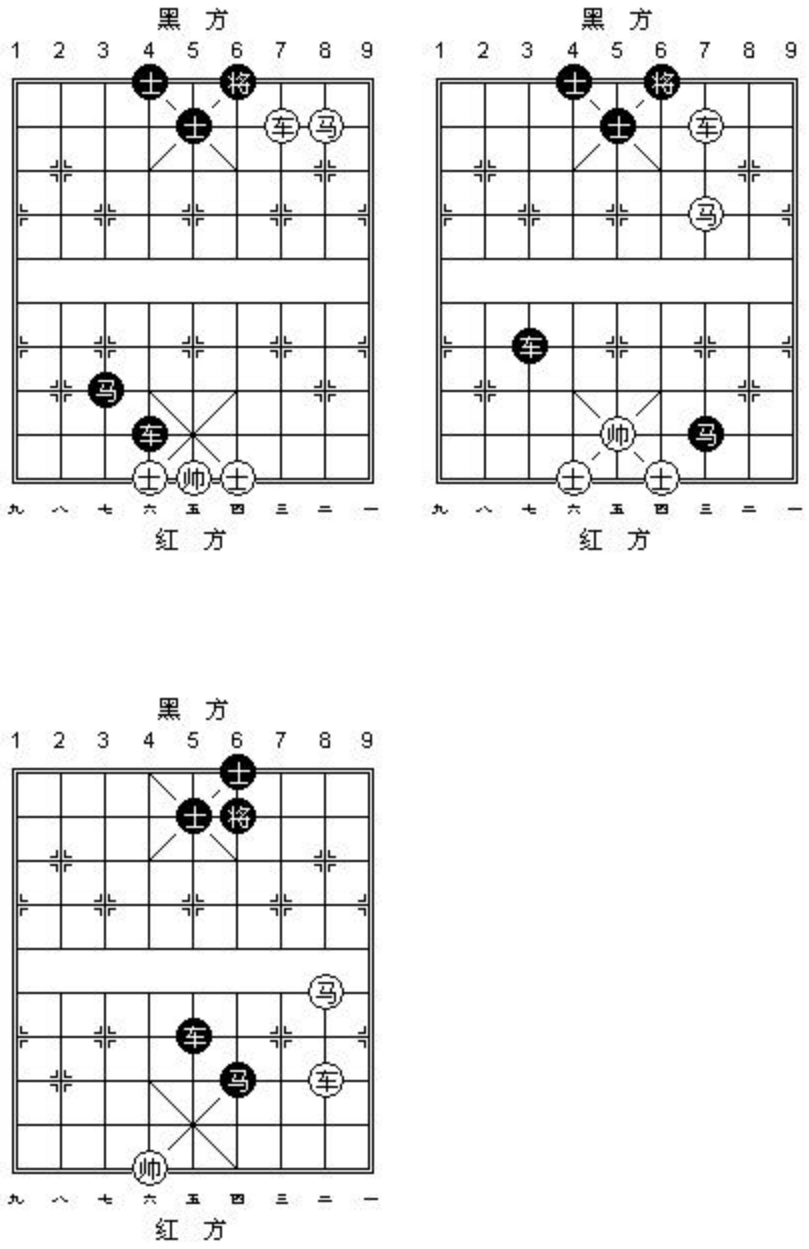

Figure14: six types of car-horse formation.

Once again, this type of evaluation function can be defined as:

$Fx = 1 * g$, if formation x exists then the value is $1 * g$, otherwise 0, where g is a linear factor to adjust F.

The protecting and attacking formation may be the most frequently used relationship. In figure15, red two Horses are protecting each other, and it is called *Linked Horses*. It is a good formation to attack. The two black Cannons are protecting each other, and it is called *Linked Cannons*. The black Rook is protecting the black Horse.

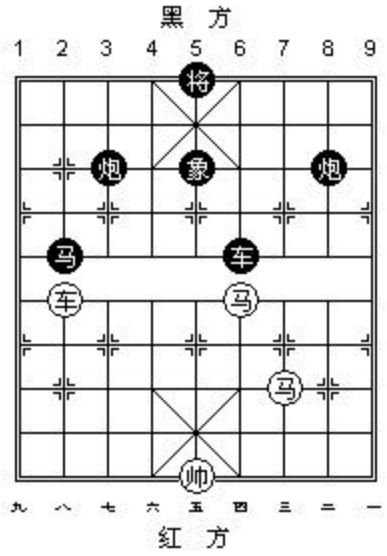

Figure15: protecting formations.

6. Combining the Four Evaluations

The four methods of evaluation above are not the only methods to evaluate a game situation. They are traditionally more human-favor methods. Chinese chess books usually talk about the methods individually and apply them to real samples. In a machine learning and statistics view, the evaluation functions are weak classifiers, which cannot classify a game situation perfectly individually. Even under the same method, there are

many alternative functions and different values. Furthermore even in the same method the adjustable factors in the formula can lead to many different functions. For example, in the position evaluation method there are two tables of values to evaluate a piece, and each of them has slightly different values for pieces evaluation.

Another problem comes from the fact that they may contradict to each other sometime. Figure16 shows an extreme situation from the puzzle game *An Angleworm and a Dragon*, in which the piece value evaluation function favors red, suggesting a winning situation for red and the position evaluation function favors black, suggesting a losing situation for red. Actually this puzzle game turns out to be a draw game. So it needs some combination mechanism to balance the functions.

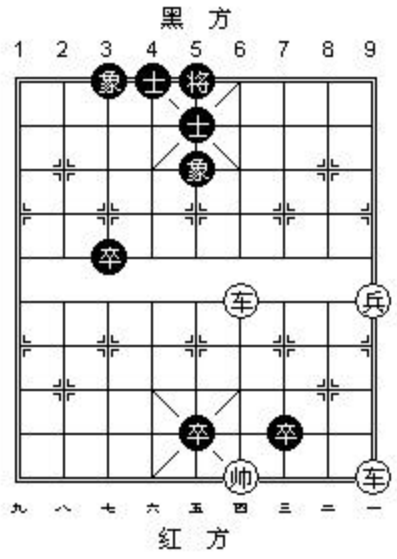

Figure16: An Angleworm and a Dragon

The third problem comes from the power of the evaluation functions. Since no function can work perfectly individually, it needs some way to find a combined function that works better than using just a single function. Here comes the need of combing the functions.

To summarize there appear three problems:

1) Which evaluation functions are good to use and which functions are redundant (providing similar information and thus can be ignored if one in the group is chosen)? How to choose the best evaluation from those among the same category.
2) If the functions suggesting different values (contradict values), how to balance them?
3) How much do they contribute to the whole situation, assuming a linear combination is used to form a combined evaluation function?

The solution is to use the AdaBoost algorithm.

7. Using AdaBoost to Combine Evaluation Functions

Although sampling methods can be used for choosing and balancing the evaluation functions (see the discussion in *Statistics Behind Evaluation Functions*), AdaBoost is chosen in this paper. AdaBoost is chosen to solve the problems in the last section. The term *classifier* is used and it is potentially the same as an evaluation function.

7.1 Discretizing Evaluation Function's Values

First of all in order to use AdaBoost, another problem needs to be solved. Since the evaluations functions mentioned use real values rather than discrete values, it is required to discretize their values. This is reasonable since accurate evaluation of a game situation is impossible. Even chess masters cannot exactly tell how much better one game situation than another, if the two situations have close values. Also in order to train the strong classifier, game samples must be given and labeled. So discretizing the function values become the only choice.

Thus by dividing game evaluation into categories: *worst, much worse, worse, slightly worse, even, slightly better, better, much better, best*, the problem can be solved. These values defined how much do they contribute to a game situation. In the piece value evaluation case, figure17 can be a way to discretize the function values.

T's Absolute Value	Discrete Value (for T > 0, when T favors red)	Discrete Value (for T < 0, when T favors black)	Possible Causes
= 0	Even	Even	No pieces lost
> 30 and < 70	Slightly better	Slightly worse	One Pawn lost
> 70 and < 170	Better	Worse	One Guard or One Elephant Lost
> 170 and <310	Much better	Much worse	One Horse or Cannon Lost

> 310	Best	Worst	One Rook or more lost

Figure17: Discrete version of piece value evaluation function

Appendix B has given the discrete version of the position tables, called converted important positions tables. They serve the same function as Figure17.

7.2 Designing Weak Classifiers

Designing of weak classifiers (evaluation functions) depends on the designer's Chinese chess and computer programming experience. It generally follows a design, trial, failure, and redesign cycle. Keeping the fast and accurate ones and discarding the slow and vague ones seem to be the only way to do it.

After the designing is done, there still can be adjustable constants that need to be determined. In the AdaBoost algorithm, all the weak classifiers can be put together and chosen by the algorithm.

7.3 The AdaBoost Algorithm

After the weak classifiers are ready, it is time to run the algorithm. The algorithm for multiple classifications follows:

```
//m: total of data samples
//n: total of weak classifiers
//D: data sample weights, size m vector
```

```
//error: classifier errors, size vector


// initialize data sample weights to 1/m for each sample
step1:
for i = 1 to m
  D(i) = 1/m


// for each classifier, find its total error respect
// to the weights D
step2:
for i = 1 to n
   error (i) = 0
   for j = 1 to m
       if (classifier i classifies sample j incorrectly)
           then error (i)= error (i) + D (j)


//pick up the classifier with the smallest error
step3:
min_error = infinity
for i = 1 to n
   if (error (i) < min_error)
      then chosen_classifier = i; min_error = error (i)


//update sample weights for all data
```

```
step4:
classifier_weight = log [(1-error(chosen_classifier))/
                    error(chosen_classifier)] + log (9 – 1)
for i = 1 to m
   if (the chosen classifier correctly classifier sample i)
      then D(i) = exp(-classifier_weight)
normalize D(i) so that sum of all D(i) equals to 1

step5:
go to step2 and repeat until enough classifiers are chosen
```

The final strong classifier is

```
n: total classifiers chosen
s: data sample
k: k classes, or k categories
classifier_weight: vector size n from the training procedure

max_vote = 0
for j = 1 to k
   vote = 0
   for i = 1 to n
      if (classifier i classify sample s to class k)
          then vote = vote + classifier_weight (i)
      if (vote > max_vote)
          then max_vote = vote; chosen_class = j
```

```
return chosen_class as the final class
```

8 Experiments

8.1 An Actual Implementation and Training Data

An actual implementation is written in Java. The four types of weak classifiers are implemented as described in the sections and the strong classifier. Major chess rules are also implemented. In order to simplify the experiment, only a min-max searching algorithm with a single threaded routine is implemented. The searching algorithm is set to search within a max width of 5 and a max depth of 4. That is: five moves per game state and 4 steps ahead (2 steps for red and 2 steps for black). The searching is about 600 nodes in total, plus some overhead searching and filtering. The total searching is about 800 nodes per move.

To simplify the experiment, only the basic features talked in the paper are implemented. Piece values are from figure5 and important positions from appendix B. Relationship evaluations only consider basic relationships such as attacking relationships, protecting relationships and simple formations.

Then the classifier is trained by 113 game data samples, which are mainly from master games. The samples are commented by professional players and chess masters. Linear factors are chosen by first running the AdaBoost algorithm then choosing the best ones.

They are:

Weak Classifiers	AdaBoost Weights	Linear Factors	Correctly Classified Samples
Piece Value Evaluation	1.59	0.7	43
Position Evaluation	1.23	1.54	34
Flexibility Evaluation	0.55	0.1	31
Formation Evaluation	0.27	0.58	26

Figure18: real data from the experiment in a total of 113 training samples

Figure19: the actual implementation of the game. (The images of the chessboard and pieces are from game.sina.com and licensed by Sina)

8.2 Results

On an AMD AthlonXP 1700+ CPU with 1G RAM, it takes about 15 seconds to compute one step. The result is amazingly good. Without complicate designs and chess knowledge included, the game engine exceeds a beginner's level. By enlarging the search width and depth and implementing more delicate designs of the classifiers, it is believed to reach a commercial level. More data sample training can improve the accuracy of the classifiers. Also the implementation of background calculating and Alpha-Beta pruning can greatly improve the performance.

9. Other Issues

More professional designs incur opening game and end game databases, which are talked in many papers. And there are possible applications of statistics theory and technologies in those designs.

10. Future Research

The costs of the evaluation functions are not considered in the AdaBoost algorithm. More complex evaluation functions can provide better classification results, but they cost more CPU time. In a real-time environment, imperfect decision must be made and the trade-off between time and evaluation quality must be considered.

The piece values and position values are from experience and more accurate values can be achieved by sampling huge game space. Yet this is another topic for research.

Similar to the *Face Recognition* paper, there is a cascading design by which can speed up the classification. This is especially useful when searching a huge amount of nodes from a game tree. The design can screen the useless nodes rapidly and apply limited CPU time to other useful nodes.

Appendix A

Introduction to Chinese chess (Xiangqi)

Board

The chessboard is rectangle with 9 by 10 lines. Chess pieces are placed at the crossings. The middle row without crossings is called a *river*, and it mimics the border between the two players. Figure20 shows the initial game setting. The player on the top is the black player, and the bottom player is the red player. The combined four cells with a great "X" are called the *palace*. There are two places on board, one for red and the other for black. The Kings and Guards can only move inside their own palaces.

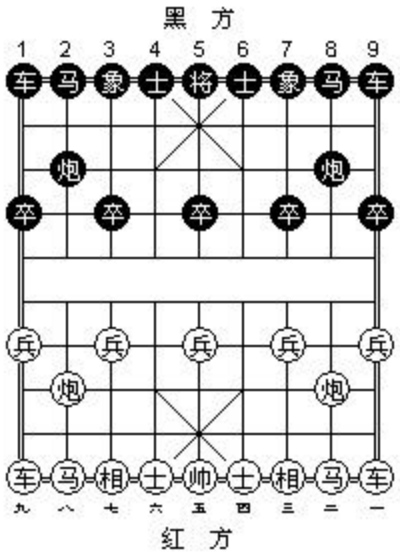

Figure20: initial setting of the game. The bottom is the realm of the red player, and the top is the realm of the black player. The central rectangle is the river.

Pieces: every piece has a pair except Pawns and Kings.

Rook/Car

The piece can move horizontally or vertically up to another piece or the borders. It can attack any enemy pieces on its way.

Horse/Knight

It can move two crosses away from its original position along the diagonal. So it can move in eight directions. When one of the obstacles next to it, it only move to the other six directions. When there are four obstacles next to it, the piece cannot move at all. The piece can attack any piece on its way when there is no obstacle on that way.

Cannon/Catapult

It moves in the same way as a Rook, but it cannot attack any pieces on its way. It can only attack the pieces behind the first obstacles on its way.

Elephant/Bishop

It can move and attack two crosses away along the diagonal, when there is no obstacle piece cross away along the diagonal.

Guard

It can move and attack only in the palace to a cross along a diagonal. It is a bodyguard to the King.

Pawn/Solider

The piece can only move forward one step, but after it crosses the river (two steps up from its original position), it can move forward, and horizontally one step, but never back. It can attack any pieces on its way.

King

The piece can move and attack one step horizontally and vertically. But the King can never move out of its palace. There is a special rule that when there is no obstacle between the two kings, the King who moves first can capture the enemy's King. Loss of the King results in the loss of the game.

Summary

Figure21: it shows the possible positions for Elephants and Guards; the Red cannon is attacking the black Rook; the two black Cannons are attacking nothing; the right black cannon can only move vertically; the red horse has no way to move; the black horse can move to either one of the eight directions; the red pawn across the river can move horizontally; the red Rook is attacking the black elephant.

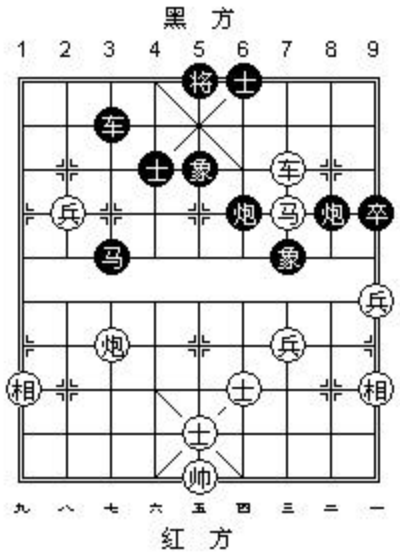

Figure21: summary all the pieces.

General Game Rules

Red player moves first and then it is the black's turn. The game ends when either player captures the other player's King. It is a draw if both players agree to do so, or no player can capture at least one piece in 120 steps.

Repeated checking and repeated capturing the pieces are forbidden except the situation is not repeated.

General Game Strategies

1) In opening games, pieces should move out from their starting positions to the important positions as soon as possible. For car important positions are riversides, palaces lines. For cannons the positions are central line and palace lines. For horses they are riversides, line 3 and 7.
2) Rook should be kept in safe places because loss of a Rook possibly causes a loss.
3) Remove Horses' obstacle when possible. Sometime Pawns are given up for the purpose.
4) Elephants and Guards should move up to tie together when possible.
5) Central Pawn should keep its original position and seldom move up.
6) Pawns have small values in opening games. They are often given up in opening games for sake of being obstacles. Pawns have large values in end games. They should be kept carefully in end games. Try to move pawns across the river when possible. A pawn reaching the top or bottom line is generally considered loss of

its value, but there are a few interesting exceptions. Figure22 shows a game puzzle for which one more bottom Pawn results in a win.

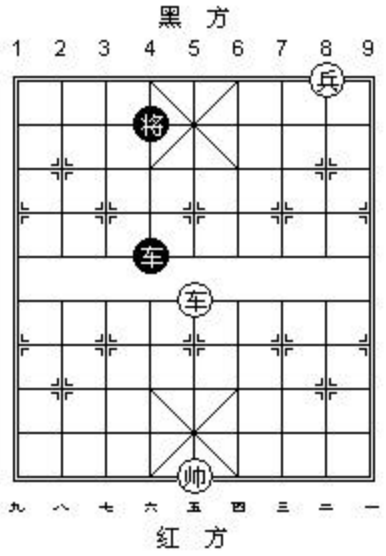

Figure22: game puzzle *" Salvage of the moon from the bottom of the sea"*

7) A single Pawn across the river is still weak and easily under attack, it should have some pieces to protect them.
8) Never fight with just one piece. Try more cooperative actions among two or more pieces.
9) Serious attacking potential in a game is worthy of a horse or a cannon. Expert game players often give up a horse or cannon to achieve serious attacking potential.

Game Puzzles

Many game puzzles are made in such a way that the situation of the game seems to favor a player and the player can win the game in just a few steps, but actually there are many traps and usually lead a draw game. Figure23 shows such a puzzle *Seven Stars*.

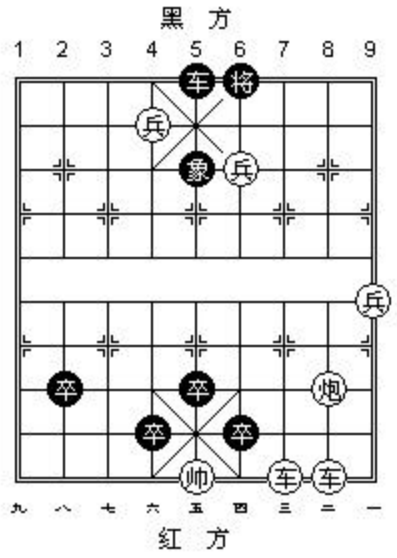

Figure23: *Seven Stars*. The name of game puzzle comes from the fact that both players have seven pieces and when the game is over, there leaves seven pieces in total. The puzzle has many different solutions that lead to a draw. It is highly recommended by chess masters for learning and researching purposes. It is designed so trickily that a wrong move out of all possible solutions must lead to a loss.

Appendix B

Important Position Tables

The important position tables are used by ELP and the converted tables are discrete version of the original tables. They are used by the AdaBoost algorithm.

Important Position Tables

14 14 12 18 16 18 12 14 14	4 8 16 12 4 12 16 8 4
16 20 18 24 26 24 18 20 16	4 10 28 16 8 16 28 10 4
12 12 12 18 18 18 12 12 12	12 14 16 20 18 20 16 14 12
12 18 16 22 22 22 16 18 12	8 24 18 24 20 24 18 24 8
12 14 12 18 18 18 12 14 12	6 16 14 18 16 18 14 16 6
12 16 14 20 20 20 14 16 12	4 12 16 14 12 14 16 12 4
6 10 8 14 14 14 8 10 6	2 6 8 6 10 6 8 6 2
4 8 6 14 12 14 6 8 4	4 2 8 8 4 8 8 2 4
8 4 8 16 8 16 8 4 8	0 2 4 4 -2 4 4 2 0
-2 10 6 14 12 14 6 10 -2	0 -4 0 0 0 0 0 -4 0

Figure24: Position values of a Rook.

Figure25: Position values of a Horse.

6 4 0 -10 -12 -10 0 4 6	0 3 6 9 12 9 6 3 0
2 2 0 -4 -14 -4 0 2 2	18 36 56 80 120 80 56 36 18
2 2 0 -10 -8 -10 0 2 2	14 26 42 60 80 60 42 26 14
0 0 -2 4 10 4 -2 0 0	10 20 30 34 40 34 30 20 10
0 0 0 2 8 2 0 0 0	6 12 18 18 20 18 18 12 6

-2	0	4	2	6	2	4	0	-2
0	0	0	2	4	2	0	0	0
4	0	8	6	10	6	8	0	4
0	2	4	6	6	6	4	2	0
0	0	2	6	6	6	2	0	0

Figure26: position values of a Cannon.

2	0	8	0	8	0	8	0	2
0	0	-2	0	4	0	-2	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

Figure27: position values of a Pawn.

Converted Important Position Tables

Worst: -4, much worse: -3, worse: -2, slightly worse: -1, even: 0, slightly better: 1, better: 2, much better: 3, best: 4

3	3	2	3	3	3	2	3	3
3	4	3	4	4	4	3	4	3
2	2	2	3	3	3	2	2	2
2	3	3	4	4	4	3	3	2
2	3	2	3	3	3	2	3	2
2	3	3	4	4	4	3	3	2
2	2	2	3	3	3	2	2	2
1	2	2	3	2	3	2	2	1
2	1	2	3	2	3	2	1	2
0	2	2	3	2	3	2	2	0

Figure28: position values of a Rook.

1	2	3	2	1	2	3	2	1
1	2	4	3	2	3	4	2	1
2	3	3	4	3	4	3	3	2
2	4	3	4	4	4	3	4	2
2	3	3	3	3	3	3	3	2
1	2	3	3	2	3	3	2	1
1	2	2	2	2	2	2	2	1
1	1	2	2	1	2	2	1	1
1	1	1	1	0	1	1	1	1
1	0	1	1	1	1	1	0	1

Figure29: position values of a Horse.

4 3 2 1 1 1 2 3 4

3 3 2 2 1 2 2 3 3

3 3 2 1 1 1 2 3 3

2 2 2 3 4 3 2 2 2

2 2 2 3 4 3 2 2 2

2 2 3 3 4 3 3 2 2

2 2 2 3 3 3 2 2 2

3 2 4 4 4 4 4 2 3

2 3 3 4 4 4 3 3 2

2 2 3 4 4 4 3 2 2

Figure30: position values of a Cannon.

0 0 2 2 2 2 2 0 0

3 4 4 4 4 4 4 4 3

3 4 4 4 4 4 4 4 3

2 4 4 4 4 4 4 4 2

2 2 3 3 4 3 3 2 2

0 0 2 0 2 0 2 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 0

Figure31: position values of a Pawn.

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 -1 0 0

0 0 0 0 0 0 0 0 0

-2 0 0 0 3 0 0 0 -2

0 0 0 0 0 0 0 0 0

0 0 1 0 0 0 1 0 0

Figure32: position values of an Elephant.

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 0

0 0 0 -1 0 -1 0 0 0

0 0 0 0 3 0 0 0 0

0 0 0 1 0 1 0 0 0

Figure33: position values of a Guard.

The only important position for the King is its original position, marked as 2, while others are –2.

References

[1] Ji Zhu, Saharon Rosset, Hui Zou and Trvor Hastie , (2006) Multi-class AdaBoost <www-stat.stanford.edu/~hastie/Papers/samme.pdf>

[2] Shi-Jim Yen, Jr-Chang Chen, Tai-Ning Yang and Shun-Chin Hsu, (2004) Computer Chinese Chess <http://www.csie.ndhu.edu.tw/~sjyen/Papers/2004CCC.pdf>

[3] Claude E. Shannon (1949), Programming a Computer for Playing Chess <http://archive.computerhistory.org/projects/chess/related_materials/text/2-0%20and%202-1.Programming_a_computer_for_playing_chess.shannon/2-0%20and%2021.Programming_a_computer_for_playing_chess.shannon.062303002.pdf>

[4] Paul Viola and Michael Jones, (2001) Rapid Object Detection using a Bossted Cascade of Simple Features,

[5] Wikipedia Xiangqi Introduction <http://en.wikipedia.org/wiki/Xiangqi>

[6] Chess Sky (top level and bilingual Chinese chess game site) <http://www.chesssky.net>

[7] An Introduction to Chinese Chess <http://home1.gte.net/res1bup4/chess_intro.htm>