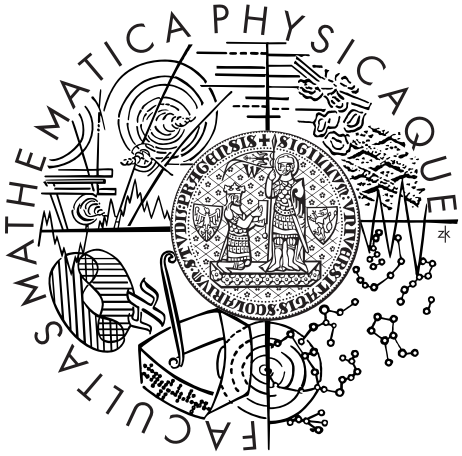
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Josef Moudřík

Meta-learning methods for analyzing Go playing trends

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis:	Mgr. Roman Neruda, CSc.
Study programme:	Informatics
Specialization:	Theoretical Computer Science

Prague 2013

I would like to thank my supervisor Mgr. Roman Neruda, CSc. for his help and insightful comments. I would like to thank Petr Baudiš for discussing topics regarding Pachi and Vladimír Daněk for his observations regarding the definition of the style scales. Moreover, I would like to thank Alexander Dinerchtein, Motoki Noguchi, Vladimír Daněk and Vít Brunner for filling in the style questionnaire.

Finally, I would like to thank Jan Moudřík for proof-reading the text and the rest of my family including Šárka for love and support.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... date .....

Title: Meta-learning methods for analyzing Go playing trends

Author: Josef Moudřík

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Roman Neruda, CSc., Institute of Computer Science, Academy of Sciences of the Czech Republic

Abstract: This thesis extends the methodology for extracting evaluations of players from samples of Go game records originally presented in (Baudiš – Moudřík, 2012). Firstly, this work adds more features and lays out a methodology for their comparison. Secondly, we develop a robust machine-learning framework, which is able to capture dependencies between the evaluations and general target variable using ensemble meta-learning with a genetic algorithm. We apply this framework to two domains, estimation of strength and styles. The results show that the inference of the target variables in both cases is viable and reasonably precise. Finally, we present a web application, which realizes the methodology, while presenting a prototype teaching aid for the Go players and gathering more data.

Keywords: Go, function approximation, machine learning, evolution of ensembles

Název práce: Meta-učící metody pro analýzu trendů her Go

Autor: Josef Moudřík

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: Mgr. Roman Neruda, CSc., Ústav informatiky AV ČR

Abstrakt: Práce rozšiřuje metodiku pro ohodnocování hráčů hry go na základě záznamů jejich her, kterou jsme dříve publikovali v (Baudiš – Moudřík, 2012). V této diplomové práci jsme nejprve přidali některé featury a navrhli metodiku pro jejich porovnávání. Následně jsme představili robustní framework, který je pomocí metod strojového učení schopen zachytit závislosti mezi ohodnoceními hráčů a obecnou závislou proměnou. Tento framework spočívá v evoluci ansámblových metod strojového učení. Aplikovali jsme jej na dva problémy — predikci síly hráčů a stylů jejich hry. Výsledky ukazují, že v obou případech je možné tuto predikci provést s rozumnou přesností. Jedním z výsledků práce je i webová aplikace, která demonstruje metodiku navrženou v této práci, slouží jako pomůcka pro studium hry go a umožňuje další sběr dat.

Klíčová slova: Go, aproximace funkcí, strojové učení, evoluce ansámblů

Contents

Introduction

Go is a board game with simple rules, yet very complex gameplay and strategies. It has a strong support in many countries, with millions of players worldwide.[1] As many of these (mainly amateur) players play online[2] and the top professional matches are usually recorded and published in magazines or online (Yevstygnyeyev, 2013; van der Steen, 2013), there exists a large number of game records for players of different skill. Moreover, since the game held a strong social status in the past, there are lots of historical records as well, though mainly from Japan of 17th century and later (Hall – Fairbairn, winter 2011a). Usually, the records of the master-level players are studied manually to grasp a deeper understanding of the game and to improve one's intuition.

So far, not much has been done in analysing these records using computers. There are programs that serve as tools to study the opening phase of the game by giving simple statistics of next move from professional games (Görtz, 2012; de Groot, 2005). The professional games have also been used in computer Go; patterns from the professional games are used as a heuristic to improve the tree searching, e.g. (Coulom, 2007). Apart from these, we are not aware of any other uses.

In (Baudiš – Moudřík, 2012), we have devised a general methodology for evaluating a player based on a sample of games he played. By comparing the evaluations of different players we were able to distinguish between players of e.g. different strengths, under the assumption that players who have similar strength should have similar evaluations.

This work presents the methodology, extended in several ways:

1. We introduce new features into the evaluation and compare their contributions.

2. We refine the machine learning methods used to analyze these evaluations.

3. We test the methodology on larger samples of games, and improve the dataset sampling to be more accurate.

4. We demonstrate the concept by a web application, which also serves as a simple teaching aid to Go players, while gathering more data.

Outline

This thesis is organized as follows. Firstly, we present the game of Go (Chapter 1). Secondly, we discuss the problem at hand and sketch our approach to it (Chapter 2). Next two chapters present the features used to extract information from the games (Chapter 3) and the machine learning methods used to learn the dependencies (Chapter 4). The actual experimental results are detailed in Chapter 5. Finally, we discuss the results and future directions (Chapter 6).

[1] A Japanese 2002 estimate of Go Census (2002) gave an estimate of 24 millions of Go players worldwide.

[2] E.g. on the Kiseido Go Server (Shubert, 2013a).

1. Game of Go

The game of Go is one of the oldest board games known to humankind, the earliest records spanning back to 500 B.C. (Fairbairn, 1995). As time passed by, Go became an important part of East Asian culture. Especially in Japan, Go has had a very important social status. Despite this popularity, the game was largely unknown in the West until the beginning of the 20th century (van Ees, 2005). Today, Go's popularity is spread almost all over the globe, though the strongest players still reside in Korea, China and Japan.

Go is a two-player game with perfect information. The two players take turns in placing stones on free intersections on a playing board (*goban*). The first player has black stones, the second one has white stones. The most widely used board size is 19×19 intersections, the 13×13 or 9×9 boards are also quite usual.

The simplicity of rules implies that a player can play on almost all of the empty intersections. In no way does this mean that every valid move is a *good* move as well. Quite the opposite is true — the majority of valid moves are terrible. This means that Go has a very large branching factor, which makes it a hard problem for computers. The computer Go is currently a very popular field of study, because it eludes the traditional AI techniques. For instance, brute-force searching is not applicable in Go, because the search space explodes long before any nearly good solution is found. In the last decade, a big progress has been made with Monte-Carlo tree search methods (*MCTS* for short). The main idea is that the probability of a player winning given he plays a particular move can be approximated using random simulations. It is surprising how well this technique performs. Recently (in march 2012), a computer program Zen beat a former top Japanese professional Takemiya Masaki at a 4-stone handicap game (the computer being the weaker player) (Wedd, 2013).[1] A good survey on the MCTS is given by Browne et al. (2012).

1.1 Basic Rules

This section presents a minimal working overview of the game rules and it might be skipped if you are familiar with the game. However, it is not meant as a tutorial to the game, for there is an abundant supply of study material online. For example, see Sensei's Library (2013k) for a nice introduction.

Go is a game which has very simple rules. Players[2] take turns in placing black and white stones at the free intersections on the board. Players may choose to give up on the right to play, this is called a *pass*. When both players *pass*, the game is finished. Next, we shall clarify what comprises the moves.

Definition. *A **liberty** of a stone (or a group of stones) is an open intersection*

[1] The top programs combine many techniques in line with MCTS, usually by skewing the distributions obtained by random simulations in direction given by some prior knowledge. This might include knowledge from pattern matching (good shapes have bonus) dictionaries of openings (good openings have bonus), local searches and other heuristics. See (Wikipedia, 2013) for a good overview.

[2] In Go, the black player *Black* is referred to as she, while the white player *White* is referred to as he.

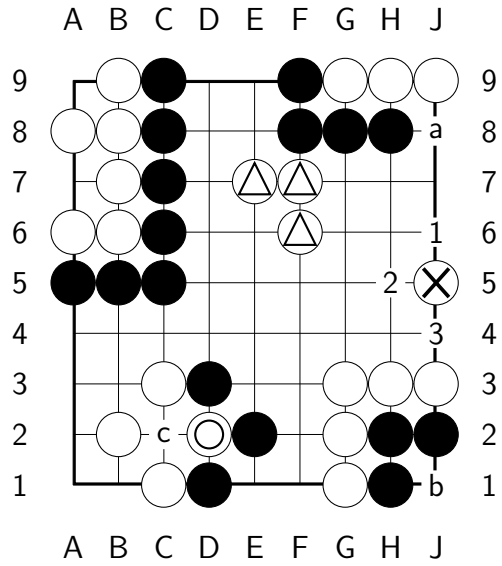


Figure 1.1: Basic situations.

directly next to the stone (or in the direct neighbourhood of the group of stones).

For example, the stone ⊗ in Figure 1.1 has 3 liberties (intersections at the top, left and bottom of the stone, labelled 1, 2, 3 in the figure), the △ group of 3 stones has 6 liberties in total. We say that a stone (or a group of them) is in **atari** if it has only one liberty.

Essentially, there are only two main rules in Go:

Definition. ***Rule of liberties***. *Every stone (or group of stones) on the board must have at least one liberty. Stones that have just lost their last liberty are removed from the board, we say they were captured by the player who removed the last liberty.*

Definition. ***Rule of ko***. *The stones on the board must not repeat any previous position of stones. Such moves are forbidden.*[3]

Example situations: (see Figure 1.1)

- The first rule defines basic mechanics of the game. For example, because the 3 white stones in upper right corner of Figure 1.1 are in *atari*, Black can capture them by playing at **a** to take the last liberty away. Once he places his stone at **a**, he removes the 3 white stones from the board.

- Because it has only one remaining liberty (at **b**), the black group of stones in the lower right corner can be captured if White plays at **b**. Effectively, at the time when White puts down his stone at **b**, this stone also has no liberty. This "suicide" is *only* allowed if the move itself removes the last liberty of some other stones. These stones are removed from the game by

[3]This rule effectively denies infinite loops.

the first rule and the capturing stone regains liberties. By the end of the move, once White removes the black stones, the white stone at **b** has 2 liberties — at H1 and J2.

- The white group in the upper left corner has two liberties (coordinates A7 A9). To capture the group, Black would need to take both of these liberties away with a single move, but this cannot be done. Therefore, the white group in the upper left corner is unconditionally *alive*.

- To illustrate the *ko*-rule, lets have a look at the lower left of the figure. Suppose it is Black's turn and she captures the ◎ stone by playing at **c**.

  Next, it is White's turn, but because of the *ko*-rule, he cannot recapture the black C2 stone by playing at D2 and so he has to play elsewhere. Because this other play changes the position of stones on the board, he can capture the black stone the next time he plays.

Definition. *An **eye** of a group of stones is a liberty that is enclosed by stones of one colour.*

The group in the lower right corner of Figure 1.1 has one eye, the group in the upper left corner has two eyes.

Definition. *A group of stones that has at least two eyes is called **alive** — it cannot be killed. There is no way for the opponent to capture the stones. On the other hand, a group that can neither make two eyes, nor be rescued*[4] *is **dead**.*

The black group in the lower right of Figure 1.1 is *dead*, whereas the white group in the upper left is *alive*.

Scoring and rulesets

The main objective players are trying to accomplish is to have more points than the opponent and thus win the game. So far, we have only presented rules that define where the stones can be put. The scoring proceeds as follows:

Definition. ***Territory scoring*** *(Japanese scoring) — during the course of the game, each player keeps the stones he has captured (these are called* prisoners*). At the end of the game,* dead *stones are removed from the board. The dead black stones are added to White's prisoners and vice versa.*

The total number of points each player has equals the number of free intersections enclosed by his stones plus the number of his prisoners. Usually, White also receives a compensation (usually 6.5 *points) for Black playing first, this compensation is called **komi**.*[5]

The player with more points wins the game.

Example of a finished game: (see Figure 1.2)

- Because the area marked with **a** is enclosed by black stones, it is Black's territory. There are 18 open intersections, so Black has 18 territory points here.

[4] E.g. by connecting with a group which is alive or by capturing the enclosing enemy stones.
[5] By playing first, Black has the initiative in the beginning.

- The area labelled with **b** is not completely enclosed by white stones, because there is the black group of 5 stones marked with **d**. But since this group does not have *two eyes* and there is no way for Black to make them, it is regarded as *dead*. (For example, by playing at A8, White can always capture the two Black stones at A7 and B7. If Black plays A8 in order to prevent this, White captures the 4 stones by playing B9.) Notice, that if there was a black stone at **c**, the group would be *alive*. When White played **c** he therefore *killed* the group. In this situation (the first one to take the point has the profit) we say that the **c** point is a *vital point* (of the black group in the upper left corner). Usually, it is a good idea to play such points before the enemy does.

  So, White has 22 points of territory here (including the intersections under the 5 dead black stones) plus 5 points for the black prisoners.

- Next, there are 2 intersections at **e**. Because they are not solely enclosed by any of the players, these intersections are not counted as territory. We call such intersections *neutral points*.

- Finally, there is the white group labelled with **f** in the bottom-right corner. This group has two eyes (H2, J1) and is thus alive. White has another 2 points of territory here.

- To sum up, Black has 18 territory points in the bottom-left corner. Because he killed the Black's group in the upper left corner, White has 5 prisoners plus 22 territory points for the upper part. White also has 2 points in the lower-right corner. Finally, because Black was the first to play, White receives a compensation of 6.5 points.

  Together, Black has 18 points, while White has $22 + 5 + 2 + 6.5 = 35.5$ points. White wins with a clear lead.

In the prior example, determining the *status* (whether it is *alive* or *dead*) of the black group in the upper left corner was quite easy. It is not necessarily always so. The disputes over statuses of groups are usually settled by resuming the game and playing the situation out.

There are also certain situations involving groups of both players which are neither alive nor dead. This situation is called *seki* (dual life). The situation occurs when groups of different colors share some liberties and the first player to fill one of the shared liberties gets captured. Therefore, neither player will play there and neither group of stones will be captured. See Sensei's Library (2013i) for a complex discussion of the topic.

It should be noted that there are other variations of the scoring and rules. For example, under Chinese scoring, players get points for *area under stones* instead of the *territory enclosed by the stones* as we saw here. Both scoring methods do not differ vastly, usually, the difference is at most 1 point. Refer to Sensei's Library (2013h) for details.

Apart from the scoring method used, the rulesets may differ in their approach to the *ko* situations and other minor settings. However, the gameplay and strategy remains almost the same. See Sensei's Library (2013g) for detailed overview of different rulesets.

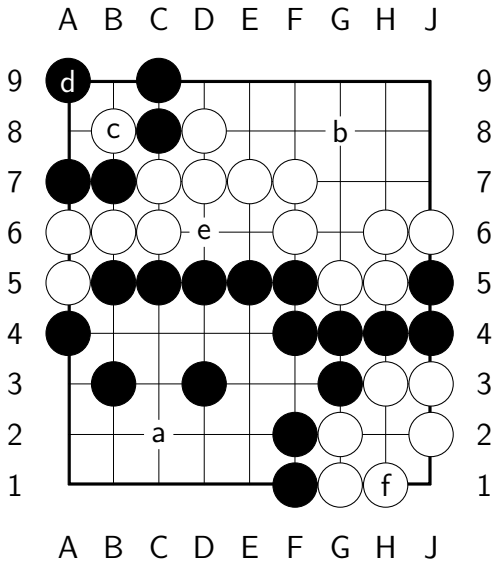


Figure 1.2: A finished game.

1.2 Rating and Handicap Games

Even though the basic rules of the game are quite easy and even very small children are able to absorb them, mastering the art of Go takes many years of study. To compare strengths of different players, a ranking system has been devised. Traditionally, the ranks are divided into *dan* and *kyu* classes. The *dan* classes are regarded as *master* ranks, *kyu* ranked players are regarded as students.[6] (Wikipedia, 2012)

The *kyu* scale spans from absolute beginners (20-kyu), to moderately skilled players (1-kyu), the *dan* scale spans from 1-dan (directly above the 1-kyu) to 9-dan.[7,8]

In comparison with other games, most notably Chess, Go is unique in a sense that even players with different skills can set up an even game. This is done using so-called *handicap stones*. The weaker player can place down a certain amount of stones before the game starts[9] — so that he has an advantage to begin with. This advantage balances the difference in strengths. The ranks are cleverly scaled; the difference in rank is equivalent to number of handicap stones needed.

[6]Intriguingly, this ranking system originally devised for Go during the Edo period in Japan (Hall – Fairbairn, winter 2011b) has also been adopted in some martial and fine arts of eastern origin.

[7]Professional players also use the dan ranks, but the scaling is a bit different. Approximately, the first professional dan (1-pro) is equivalent to amateur 7-dan. Historically the difference of 1 rank between two professional players was about 1/3 of a handicap stone. Nowadays however, the pro-dan scale serves as an indication of achievements, rather than an exact comparison of players' strengths. (Sensei's Library, 2013c)

[8]There exists a number of different ratings, that are often not directly comparable to each other. For example, the KGS (one of the popular online Go servers) 3-kyu could play evenly against European 7-kyu (the official European rank, given by the European Go Federation). Refer to (Sensei's Library, 2013f) for a deeper comparison and discussion.

[9]The positions of these stones are defined depending on their amount.

For example, 10-kyu player should place five handicap stones against a 5-kyu player for the chances of winning to be roughly similar. Also, when the handicap stones are in use, the *komi* compensation is usually changed to 0.5 (to avoid a tie).

1.3 Important Concepts

The following lines present some important concepts of Go in a strongly simplified manner.

1.3.1 Gameplay

The way humans play Go has some characteristics. The usual game can be roughly divided into three stages, opening (*fuseki*), middle game and endgame (*yose*). This distinction is very rough and there are no precise boundaries. Sometimes, the middle game fighting occurs almost immediately after the start, sometimes, the middle game is "skipped" altogether.

The *opening* sketches the main territories and war zones, optimally in line with some high level strategy. The stones do not usually come in direct aggressive contact with each other during the opening. Usually, the corners are occupied first, because it is easiest to get territory here (the corner territory needs to be enclosed only from two sides). After the corners, extending to the sides and then into the center are next big moves.

The *middle game* is dominated by attack and defence. Players struggle to reduce enemy territory, increase their own territory or otherwise gain an advantage. Sometimes, a running fight occurs — a group which is not locally alive runs away towards friendly forces in order to connect with them and assure life.

The *endgame* phase begins once statuses of all groups have been more or less determined. In the endgame, players seek to gain local advantage. In the first two stages, it is usually critical to view the board globally (as groups influence each other), while the endgame may usually be broken down to individual independent plays (how to combine them and how to choose the proper play is however a thing that must also be regarded globally).

1.3.2 Effectiveness

In Go, the goal is to have more points than opponent. The player who safely encloses more territory with fewer moves than the opponent wins. Therefore, the effectiveness of one's moves is of utmost importance. This does not limit us to enclosing territory directly. The player who needs less moves to stabilize his group (make sure it is alive) can "spend" the remaining moves e.g. to attack enemy groups, increase his own territory and so forth.

Shape

For example, the concept of **shape** is an incarnation of this principle. *Good shape* refers to a formation of stones that has good tactical possibilities: space for eyes, high number of liberties, possibilities of escaping, and so on. On the other hand, *bad shape* may easily be attacked, does not defend one's other weaknesses,

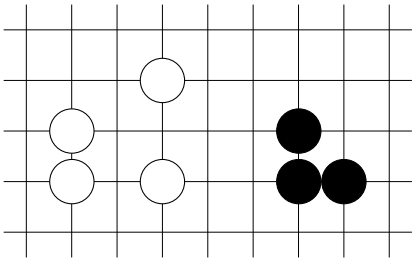


Figure 1.3: Shapes. White has a good shape, while black stones form the terrible empty triangle.

etc. Figure 1.3 shows such shapes. The white stones show a typical good shape (called the table shape). The white stones are connected, have a lot of liberties and have a chance of forming an eye in the middle. The black stones do, however, form a bad shape (the notoriously known empty triangle). This is a formation of 3 stones that has the fewest liberties, not much eye-space and can usually be attacked profitably. Black will need to invest further stones to make these stones useful.

Sente/Gote

Another really important concept is the notion of **sente** and **gote**. We say that a move (or a series of moves) is *sente* if the opponent has to respond to it or something bad happens to him (the burden of not responding is not worth the initiative taken by not responding). On the other hand, when a player plays a *gote* move, the opponent does not have to respond (the burden of not responding is relatively small), or the player who started has to respond to the opponent's response. Of course, the sequence might take more moves. The important thing is that with *sente* play, the player retains the initiative after the sequence ends. With *gote* play, he does not.

For example, Figure 1.4 shows an example endgame situation. Black descend to edge at **a** is a *sente* move, for if White does not protect at **b**, Black **b** kills the white corner group by destroying its eye. Later, Black can utilize the stone at **a** to reduce white territory by playing **c**, possibly again in *sente*. On the other hand, if White decides to protect by playing at **a** before Black does so, he loses the initiative, because Black is not forced to reply. White **a** is thus a *gote* move.

The *sente* play keeps the initiative, the *gote* play gives it to the opponent. This does not mean that *gote* is necessarily a bad move. It may be the case that the player has no *sente* play anymore, or that by playing the *gote* move a player neutralizes a big *sente* move of the opponent. Keeping an eye on what moves are *sente* or *gote* is one of the most important things a player needs to learn in order to improve. Keeping the initiative, making profits in *sente*, preparing own *sente* plays while neutralizing opponent's is crucial in Go.

1.3.3 Balance of Power

Beyond the relatively simple local goals (such as keeping one's stones in a good shape), players usually follow some deeper strategy. For example, someone who

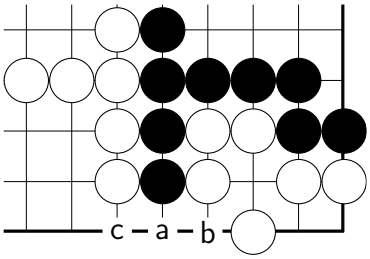


Figure 1.4: Sente and Gote. Black descend at **a** is sente, Black **b** would kill the white corner group. Later, Black can play **c**.

likes to have secure territory will probably keep his stones near the sides and his corners. Usually, moves played on third line (from the edge) are hard to invade, so they guarantee relatively secure piece of territory. By playing on a larger scale (playing higher, and perhaps less safely) his opponent might however sketch a much bigger area, which if turned into real territory would guarantee him the victory.

Also, there is the concept of *outer influence*. Player might have a strong wall of stones facing the center, which — if used wisely — will provide invaluable help in the middle game fights and indirectly provide score points. For example, if you have a strong position right where the unstable enemy's group is looking for support, then you managed the situation well, remember that points are for *both* dead enemy stones and enclosed intersections.

To sum up, player should try to balance the influence with territory, *thickness* (thick positions have little or no weaknesses) with *speed* (fast positions have potential to expand and secure large territories, yet they inherently have weaknesses) to maximize the gain and win the game. There exist whole anthologies of books dealing with these concepts, we have just presented a small, shallow part of it to illustrate the complexity of the matter. If you want to learn more, the Sensei's Library (2013a) is a good place to start.

2. Methodological Approach

2.1 Central Problem

We shall now formalize the problem we are dealing with. Suppose we have a set of players P and for each player $p \in P$, we have a sample of p's games, G_p. Moreover, suppose that for each player p we also have an externally given information $y_p \in \mathbb{R}$. For example, this might be p's strength. The central question is: *What is the relation between games G_p and the external information y_p?* The motivation is that understanding this relation may help with the general understanding of y_p. In the example case where y_p is strength of player p, this has obvious importance — it might help a player to become stronger, deepen our understanding of Go, or just improve the performance of Go playing programs.

The methodology in this work deals with a slightly weaker questions: *What can we deduce from games G_p?* and *How well can we predict the information y_p supposing we know G_p?* We approach these questions by evaluating the games G_p on a per-move basis and applying machine learning algorithms to the evaluations.

Generally, we consider the problem not to be an easy one. A crucial principial obstacle is illustrated by a Go proverb[1]: "If you want to improve, do not look on *what* moves do the professionals play, but *why* do they play them." In some situations a particular move is perfect, other times the same move is no good — without reasoning about *why* are the moves played, we cannot hope to fully tackle the problem. Moreover, there are many other factors hindering the process, to name a few:

- Often, we are dealing with small samples of data.
- The set of games G_p may be taken from a larger interval of time, during which the y_p might have changed considerably.
- The uniqueness of every single game introduces inhomogeneities into the data.
- Games might have different time setup. For example, very fast *"blitz"* games (time for one move is very small) do not make it possible to examine the positions thoroughly; these games are mainly played by intuition.
- During online games, players might not be concentrated fully on each game, resulting in unstable performances. For example, this is the case when opponents are from different timezones (for instance, one player is playing in the morning, the other one in the night) as noticed in (Sensei's Library, 2013f).

2.2 Processing Overview

The processing pathway presented in this work follows the structure from our previous paper (Baudiš – Moudřík, 2012). The pathway has two logical parts

[1] See (Sensei's Library, 2013e) for more proverbs.

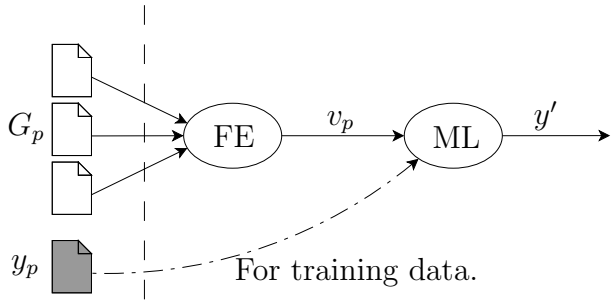


Figure 2.1: Simplified overview of the processing of data. Player's games G_p are processed using *Feature extraction* part (FE). The resulting feature vector v_p serves as an input for the *Machine learning* part (ML) which outputs predicted y'. For players from the training dataset, the machine learning methods learn from v_p and the external information y_p.

— feature extraction step, where we transform a set of player's games into an evaluation vector, and machine learning, where we learn the dependency between evaluation vectors v of different players and the information y we study in each particular dataset.

Dataset

Dataset is a set of tuples where the first element represents games G_p of a player, and the second element represents the external information $y_p \in \mathbb{R}$ we are studying.

$$D = \{(GC_i, y_i), \ldots\}$$

Because a game is played by two players and we need to distinguish between them, each game is accompanied with color of the player of interest (p's color). To emphasize this, we use GC_p instead of G_p. The GC_p is a set of tuples $GC_p = \{(game_1, color_1), \ldots\}$ (we will call it a *set of colored games* of player p).

Feature extraction

The goal of the feature extraction part is to make a complex evaluation $v \in \mathbb{R}^f$ out of a set of colored games GC. The process is detailed in Chapter 3.

Machine learning

The machine learning part tries to capture the dependency between evaluation vectors v_p and the external information y_p. Details are given in Chapter 4.

3. Feature Extraction

This chapter presents a methodology for extracting information from a set of colored games GC (see Section 2.2 for definitions).

In some of the methods below, we will need a set of games, we call it A, that reasonably represents all the games from the dataset D. By a reasonable representation we mean that the set A has the same (or almost the same) distribution of moves and other "events" of interest. All the games from D serve ideally, typically some reasonably large subset of D performs well, while saving computational resources.

The feature extractors basically map a set of colored games to a feature vector:

Definition. *A **feature extractor** is a function*

$$f : f(GC) = v$$

where GC is a set of colored games, and v is the resulting feature vector.

In the sections to come, we will separately present feature extractors f we have used. The machine learning methods in Chapter 4 will use the concatenation of these different features.

In the following text, we distinguish between *raw features* — the per-move features matched by Pachi Go engine (see the next section) — and normal *features* — results of applying aforementioned *feature extractors* on a set of colored games. The *features* are computed using the information from the *raw features*. Also, any single element v_i (of a feature vector v) is called an *attribute*.

The implementation details are given in Appendix B.

3.1 Raw Features and Pachi

To extract the *raw features*, we have used the Pachi Go engine (Baudiš et al., 2012). Apart from being quite a good-performing Go bot, Pachi engine has a replay mode, that scans a single game on a per-move basis. For each move, it outputs a combination (called *pattern*) of several raw features (key-valued pairs). These raw features include:

- **atari flag** — whether the move put enemy stones in atari,
- **atari escape flag** — whether the move saved own stones from atari,
- **capture** — number of enemy stones the move captured,
- **contiguity to last move** — the gridcular distance (presented below) from the last move,
- **board edge distance** — the distance from the nearest edge of the board,
- **spatial pattern** — configuration of stones around the played move.

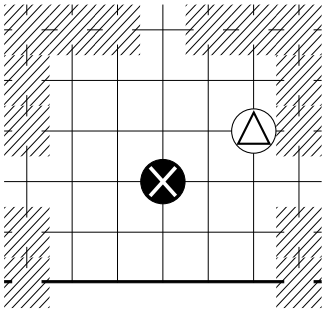


Figure 3.1: An example spatial pattern of size 6. The dashed parts of the goban are not regarded.

The spatial patterns are always normalized (using a dictionary, below) to be Black to play and to be invariant under rotation and symmetry. For each move, spatial neighborhoods of sizes 2 to 6 in gridcular metric are matched.

For example, the following pattern

$$(border : 2, cont : 5, spatial : 88)$$

has three raw features. The first one, the distance from the board edge is 2, which means that the move is on the third line. The contiguity feature says that the gridcular distance from the last move is 5. From Figure 3.2, we can see that distance 5 is the horse move approach (both Go and Chess have what is called a horse move, with the same L-shape).

The last feature — spatial pattern — gives index to the spatial dictionary (below), the particular pattern for this example is shown in Figure 3.1. From the figure, we can see that the move ⊗ was probably some low counter-extension (or invasion) to answer △.

Gridcular metric

Gridcular metric approximates a circle on the square grid of a goban. It is defined by the formula:

$$d(x, y) = |\delta x| + |\delta y| + \max(|\delta x|, |\delta y|)$$

The gridcular metric has been successfully applied for pattern-matching in e.g. (Stern et al., 2006; Coulom, 2007). The gridcular distance is illustrated in Figure 3.2.

Spatial dictionary

Before running the engine on a per game basis, the engine is run on a large number of games (the A set) to create a dictionary of spatial patterns that have occurred at least N times, N chosen so that the number of spatial patterns is sufficiently large. The raw spatial feature matcher uses this dictionary to look for spatial patterns (and their rotations, color inversions if White is to play, symmetrization) during the matching.

9	8	7	6	7	8	9
8	6	5	4	5	6	8
7	5	3	2	3	5	7
6	4	2	0	2	4	6
7	5	3	2	3	5	7
8	6	5	4	5	6	8
9	8	7	6	7	8	9

Figure 3.2: The gridcular metric on a 7×7 grid. The numbers show the gridcular distance from the center, the background lightens with increasing distance. Inspired by (Stern et al., 2006).

3.2 Pattern Features

This section presents a *pattern feature extractor* that tries to capture a distribution of patterns among the colored games GC. In this case the pattern is a tuple consisting of the atari flag (and atari escape flag) and the spatial pattern raw features. Other raw features are ignored because the use of more raw features causes a big granularity of the data — a lot of patterns that are not played often.

This feature extractor counts the number of occurrences of the top N most played patterns from GC. The counts are then normalized using one of the normalization schemes. The process is detailed in Algorithm 1.

We presented this feature in (Baudiš – Moudřík, 2012); this work extends it slightly by testing two more normalization schemes (independent and proportional) in addition to original linear normalization[1].

Normalization

The normalization step in Algorithm 1 (line 11 in the pseudocode) is very important to maintain the invariance under number of games in the GC. Without it, the values of $\vec{v}$ would increase proportionally to $|GC|$. To alleviate this, we use one of following scaling schemes:

- **independent normalization** — $\vec{v} \leftarrow \vec{v}/|GC|$,
- **proportional normalization** — $\vec{v} \leftarrow \vec{v}/sum(\vec{v})$,
- **linear normalization** — $min(\vec{v})$ is mapped to -1, $max(\vec{v})$ to 1, the rest is mapped linearly into the interval $(-1, 1)$.

[1] In the (Baudiš – Moudřík, 2012), we also used what could be called logarithmic normalization but it did not perform well; see the paper for details.

Algorithm 1: Pattern Feature Extractor

input : set of colored games GC, number of top patterns N, set of all games A
output: feature vector $\vec{v}$

```
1  TopPatterns ← PachiGatherTopNPatterns(A, N);
2  v ← Zeros(N);
3  foreach (game, color) in GC do
4      foreach pattern in PachiGatherPatterns(game, color) do
5          if pattern in TopPatterns then
6              i ← IndexOf(pattern, TopPatterns);
7              v[i] ← v[i] + 1 ;
8          end
9      end
10 end
11 v ← Normalize(v);
12 return v
```

3.3 Local Sente (Gote) Sequences

Besides the pattern feature, we have implemented some higher level features that try to capture deeper concepts within the data. The first one of them deals with sente and gote plays. We have discussed (see Chapter 1.3.2) that a sente play is a move the opponent has to respond to or something bad happens to him. Often, the reply is local, as we have seen in Figure 1.4.

Assuming that *sente* and *gote* sequences are always local (*assumption of locality of replies*) and that all local plays are part of some *sente* or *gote* sequence (*assumption of exclusivity*) is the basis of a method we have devised to make statistics of *sente* and *gote* play within a game.

Certainly, the assumption of the locality of replies does not always hold. Sometimes, a response to a *sente* move has to be played on the other side of board. Imagine, for example, that a large white group has one eye and it has only two possibilities to make the second eye. If Black neutralizes one of them, White should not hesitate to make the second eye or his whole group dies. Because the two possible eyes of White's group might be distant from each other, the Black's play that destroyed one of the possible eyes might not be answered locally, though it certainly is a *sente* move.

Neither the second assumption always holds. Two moves that are played next to each other might be separate *gote* plays, instead of a part of one larger exchange.

Even though the assumptions are not always true, the resulting feature vector proves to be useful, as detailed in Chapter 5.1.2.

In the following, we view a game g as a sequence of moves:

$$g = (m_1, m_2, m_3, \ldots, m_{last})$$

We consider the *pass* to be a special kind of move. Now, we shall formalize the concept of locality within the g sequence.

Definition. *(All the following definitions are considered to be within the g sequence.)*

We say that a move m_i from g is ω-local if its gridcular distance from the previously played move m_{i-1} is less or equal to ω.

We say that a sequence $M_{i,j} = (m_i, m_{i+1}, \ldots, m_j)$ for any $1 \leq i \leq j \leq |g|$ is ω-local if $\forall x, i < x \leq j$, move m_x is ω-local.

We say that an ω-local sequence $M_{i,j}$ is maximal if it cannot be extended into ω-local M' by adding move m_{i-1} or m_{j+1}.

We say that a maximal ω-local sequence $M_{i,j}$ is sente if $color(m_i) \neq color(m_j)$. Similarly, the $M_{i,j}$ is said to be gote if $color(m_i) = color(m_j)$. The $color(m)$ is a color of a player who played the move m.

The pass is not considered to be ω-local.

Lemma 1. *For a fixed ω, the game sequence g can be covered by a set of disjunct maximal ω-local subsequences.*

Proof. It is obvious that if we have two neighboring ω-local sequences $M_{i,j}$ and $M_{j+1,k}$ such that also the move m_{j+1} is ω-local, we can merge the two sequences into $M_{i,k}$ which is also ω-local.

Therefore, if we start by splitting the g into $|g|$ disjunct ω-local subsequences $M_{i,i}$ of size 1, we obtain the cover by repeatedly applying the merge (if it is possible) on neighboring pairs of these sequences until no more merges are possible.

The final set of sequences, *Cover*, has only ω-local sequences because the merge operation preserves the locality and the initial splits M_{ii} were ω-local; they are maximal, because no more merges are possible and they are a cover because the initial set was a cover and no elements are skipped during the merge operation. □

Based on the *Cover* set, we can easily count the approximated statistics of *sente* and *gote* plays as Algorithm 2 shows. The function `IsSente` is a predicate to test if the maximal ω-local sequence is *sente* as defined above. The gridcular distance to determine the ω-locality is taken from the contiguity raw feature.

3.4 Histograms Features

This section presents two histogram-based feature extractors that focus on capturing distributions of certain events within the games.

3.4.1 Border Distance

The task of the first histogram feature is to capture the distribution of distances from the board edge. In Chapter 1.3.3, we have briefly mentioned that in the opening, playing on third line generally stresses secure territory, while higher lines (e.g. the 4th) stress influence. Frequently, the difference of one line has a huge impact on the flow of the game. We could do a simple statistics of the border distance, but because the game has stages that differ significantly from each other (see Section 1.3.1), one has a feeling that some sort of differentiation based on the game stages should be used. One simple heuristic to tell the current stage is

Algorithm 2: Local Sequence Extractor

input : set of colored games GC, locality threshold ω
output: feature vector $\vec{v}$

```
1  S ← 0;
2  G ← 0;
3  foreach (game, color) in GC do
       /* The Cover function returns the Cover of game from the
          proof of Lemma 1 with respect to ω.                    */
4      foreach M in Cover(game, ω) do
           /* We ignore opponent's sente and gote sequences.     */
5          if M[1] == color then
6              if IsSente(M) then
7                  S ← S + 1;
8              else
9                  G ← G + 1;
10             end
11         end
12     end
13 end
   /* We output the average number of Sente and Gote sequences
      per game, and also their average difference.               */
14 v⃗ ← (S, G, G − S)/|GC|;
15 return v⃗
```

the number of the current move. (For instance, we can roughly say that first 10 moves are usually the early opening.)

We have used a two-dimensional histogram in this feature extractor. The first dimension is specified by the move's border distance, the second one by the number of the current move. The size of each dimension is given by intervals dividing the domains. For example, if we use $ByMoves = \{\langle 1, 10\rangle, (10, \infty)\}$ for the move coordinate (motivation is to distinguish between opening — say first 10 moves — and the rest of the game), and $ByDist = \{\langle 1, 3\rangle, \langle 4, \infty)\}$ (distinguish playing on first three lines to stress territory from playing higher to stress the influence) to split the border distance dimension, then we obtain a histogram of total $|ByMoves|*|ByDist| = 4$ elements. In the end, the histogram is normalized to establish invariancy under the number of games scanned (by dividing the histogram elements by $|GC|$).

The pseudocode is shown in Algorithm 3. The function `IndexOfElement` (*element*, *Intervals*) returns the index of interval $int \in Intervals$, such that $element \in int$.

Algorithm 3: Border Distance Histogram Extractor

input : set of colored games GC, an ordered set of disjunct intervals $ByDist$, an ordered set of disjunct intervals $ByMoves$
output: feature vector $\vec{v}$

```
1  V ← Zeros(|ByDist|, |ByMoves|) ;
2  foreach (game, color) in GC do
3  |  foreach move in game do
   |  |  /* We ignore the opponent's moves.                    */
4  |  |  if ColorOf(move) == color then
5  |  |  |  bdist ← GetBorderDistance(move);
6  |  |  |  X ← IndexOfElement(bdist, ByDist);
7  |  |  |  movenum ← GetMoveNumber(move);
8  |  |  |  Y ← IndexOfElement(movenum, ByMoves);
9  |  |  |  V[X][Y] ← V[X][Y] + 1;
10 |  |  end
11 |  end
12 end
   /* Serialize the normalized matrix into a vector.          */
13 v⃗ ← RowWise(V/|GC|);
14 return v⃗
```

3.4.2 Captured Stones

The second histogram feature reflects the distribution of captured stones in different game stages. The motivation behind this is the fact that one would expect generally different numbers of captives in the opening — where the stones are not usually in direct aggressive contact (see Section 1.3.1) — and in the endgame, where e.g. small captures are quite common.

The methodology here is very similar to the previous feature extractor. The first dimension distinguishes between the game stages, the second dimension has

a fixed size of three bins. Along the number of captives of the player of interest (the first bin), we also count the number of his opponent's captives (the second bin) and a difference between the two numbers (the third bin). See Algorithm 4.

Algorithm 4: Captured Stones Histogram Extractor

input : set of colored games GC, an ordered set of disjunct intervals $ByMoves$
output: feature vector $\vec{v}$

```
1  V ← Zeros(3, |ByMoves|) ;
2  foreach (game, color) in GC do
3  |  foreach move in game do
4  |  |  if ColorOf(move) == color then
5  |  |  |  X = 0 ;                                  /* The player.   */
6  |  |  else
7  |  |  |  X = 1 ;                                 /* The opponent.  */
8  |  |  end
9  |  |  movenum ← GetMoveNumber(move);
10 |  |  Y ← IndexOfElement(movenum, ByMoves);
11 |  |  capt ← GetNumCapturedStones(move);
12 |  |  V[X][Y] ← V[X][Y] + capt;
13 |  |  V[2][Y] ← V[0][Y] − V[1][Y];
14 |  end
15 end
   /* Serialize the normalized matrix into a vector.                */
16 v⃗ ← RowWise(V/|GC|);
17 return v⃗
```

3.5 Win/Loss Statistics

Finally, we came up with a pair of very simple features which make statistics of wins and losses and whether they were by points or by resignation[2]. (When player resigns, he declares his loss without finishing the game.)

For example, quite a lot of weak players continues playing already lost games until the end, mainly because their counting is not very good (they do not know there is no way to win), while professionals do not hesitate to resign if they think that nothing can be done.

For the colored games of GC we count how many times did the player of interest:

- win by counting,
- win by resignation,

[2]Sometimes — mainly in online games — players might also lose on time. In rare cases, the game might as well end as a tie or be unfinished or forfeited. We disregard such games in this feature because the frequency of these events is so small it would require a very large dataset to utilize them reliably.

- lost by counting,
- and lost by resignation.

The result of the first feature extractor are these four numbers, divided by $|GC|$ to maintain the invariancy under number of games in GC.

Furthermore, for the games won or lost by counting, we count the average size of the win or loss in points. Similarly, these two numbers in a vector form the output of the second feature extractor.

4. Machine Learning

This chapter presents machine learning methods we have used throughout this work. Most of the methods are well-documented in literature, so we only give a brief overview here.

In this chapter, suppose we have a set of data

$$Tr = \{(x_1, y_1), \ldots, (x_N, y_N)\}, \forall i : x_i \in \mathbb{R}^p, y_i \in \mathbb{R}$$

and we want to find a function r which is able to predict the value y_i from x_i with a reasonable accuracy and can generalize this dependency to unseen pairs.

Definition. *A **regression function** is a function*

$$r : \mathbb{R}^p \to \mathbb{R}$$

where p is a dimension of space of vectors of predictor variables. We also call the domain $\mathbb{R}^p$ the feature space. The codomain $\mathbb{R}$ is called a space of dependent variables.

The machine learning methods presented here are regarded as *learners*. For a given data Tr, the *learner* should output a *regression function* (also called *predictor*) which performs the regression of the dependent variable, as learned from the data.

Definition. *A **learner** is a function*

$$l : \mathcal{T} \to \mathcal{R}$$

where $\mathcal{T}$ is a space of all training datasets, and $\mathcal{R}$ is a space of all regression functions.

Of course, some *regression functions* perform better than others. Mainly, this is because each *learner* has different (inherent) assumptions about the form of the function it is looking for; we call this the *inductive bias* (of the *learner* and the underlying model). For example, linear regression assumes that the dependency between predictor variables and the dependent variables is linear. Often, we deal with data where the underlying dependency and properties of the data are unknown, so it is hard to say whether assumptions of a particular model are right. To overcome this problem, usually a bunch of models is tried and the best one is chosen.

Another approach, the one we use in this work, is not to choose the best, but rather try to combine the different approaches to create one higher-level method. Because different methods have different biases, they might be able to capture different dependencies in the data. If we combined the methods (*base learners*) usefully, we could get better performance than with the "use the best learner" approach. We call this the *ensemble meta-learning*.

Definition. *A **meta-learner** is a function*

$$ml : \mathcal{P}(\mathcal{L}) \to \mathcal{L}$$

where $\mathcal{L}$ is a space of all learners and $\mathcal{P}$ denotes a power set.

A meta-learner takes a set of learners Bl and returns a learner, possibly a result of aggregation of learners in Bl.

The different *base learners* are presented in Section 4.1. The *ensemble meta-learners* are shown in Section 4.2, and in Section 4.3 we detail the process of choosing the right base-learners into the ensemble. Finally, Section 4.4 discusses evaluation of learners and feature extractors and procedures for comparison of their performances.

The implementation details are given in Appendix B.

4.1 Base Learners Overview

4.1.1 Mean Regression

The *mean regression* is a very simple method, which we use as a reference for comparing performances of other learners.[1] It simply outputs the mean of the y's in the training set and is thus constant regardless of the input x.

$$mean(x) = \frac{1}{|Tr|} \sum_{(x',y') \in Tr} y'$$

4.1.2 Neural Networks

Artificial neural networks (*NN*) are standard technique used for function approximation. The idea behind this model is inspired by the function of biological neural tissues. The artificial neural networks are known for their ability to find dependencies between inputs and outputs in the training data and generalize this knowledge to previously unseen inputs. This section presents a very crude overview of the method, see the monograph by Haykin (1998) to learn more.

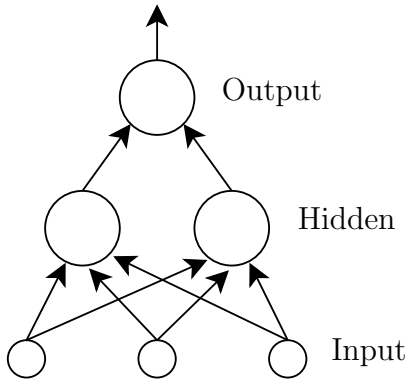


Figure 4.1: Illustration of the layered topology of a simple feed-forward neural network. The labels mark different layers.

The artificial neural network is a network consisting of interconnected computational units called *neurons*. Each neuron has several inputs x_j and one output y. For each input x_j, the neuron has a weight w_j, which is incorporated into the computation as follows:

$$y = f(\sum_{j \in J} w_j x_j)$$

where the f is a so-called activation function, for instance the sigmoid[2].

[1] A random regression is also quite frequently used for this purpose.

[2] A special case of the logistic function $\sigma(x) = (1 + e^{-rx})^{-1}$, where r controls growth of the function.

There are many topologies (defining the connections) of the *artificial neural networks* in use. We have used the typical *feed-forward* network topology, where neurons are organized in layers (one *input layer*, arbitrary number of *hidden layers*, one *output layer*). The layers are ordered so that the input to each layer only comes from the previous layer, as shown in Figure 4.1. The input layer has the same dimension as the input data, the outputs of these neurons are set to be the input data. Computation of other units proceeds by layers according to the formula above. The activity of the output layer is said to be the result of the computation.

Training

To be able to approximate the target function, the weights of the neurons w_{ij} need to be set up properly. This is a typical optimization problem, we are trying to minimize the error ϵ on the training data. In this work, we use the iterative first-order gradient-descend method called RPROP (Riedmiller – Braun, 1993). Usually, the maximal number of iterations is bounded by a limit, max.

4.1.3 k-Nearest Neighbor Regression

The *k-nearest neighbor* algorithm (Cover – Hart, 1967) is a commonly used machine learning tool. The assumption of this model is that we can deduce the dependent y by looking at vectors from the feature space that are close to the x.

Definition. *For a fixed k and x,*

let the $Nb = \{x'_1, \ldots, x'_k\}$ denote a set of k closest vectors to x from the T with respect to some metric δ;

let D be a vector of distances, such that $D_i = \delta(x, x'_i)$;

for each x'_i, let y'_i be the associated dependent variable from the training set T.

For a given x, the idea is to find the nearest k vectors (the Nb set) from the training set, and then estimate the dependent variable y from the associated $y'_1, \ldots, y'_k$.

In this work, we have used the Manhattan (p-1) and Euclidean (p-2) distances as δ. To infer the y, we define the model to be:

$$y = \frac{\sum_{i=1}^{k} w(D_i) y'_i}{\sum_{i=1}^{k} w(D_i)}$$

for some weighting function w. We have used the inverse of the distance between x and the particular neighbor instance:

$$w(D_i) = 1/D_i^{\alpha}$$

where α is a parameter specifying the effect of increasing distance. When the α is equal to zero, we obtain the averaging scheme, where the weights do not depend on the distance D_i — all the k neighbors are valued equally. With increasing α, the x'_i instances closer to x are preferred over more distant neighbors. When the α goes to infinity, the method essentially becomes one-nearest neighbor.

4.1.4 PLS

The family of *partial least squares* (*PLS*) methods assumes that the observed variables can be modelled by means of a few latent variables (their number is specified by a parameter l). The method projects the data onto this latent model in a way that minimizes error. The process is somewhat similar to Principal Component Regression.

For a good overview, see the work by Rosipal – Krämer (2006).

4.2 Ensemble Learners

In the introduction of this chapter, we have discussed that sometimes, a combination of learners can have better performance than the "winner takes all" approach. In practise, the three most used families of ensemble learning methods are bagging, boosting and stacking. In this work, we have experimented with the bagging and stacking. These are detailed in the rest of this section.

4.2.1 Bagging

Bagging[3] is a simple ensemble method introduced by (Breiman, 1996). The idea in bagging is to train a particular base learner *bl* on differently sampled data and aggregate the results. The method has one parameter t which specifies the number of the data samples. Each of them is made by randomly choosing $|Tr|$ elements from training set Tr *with repetition*. The base learner *bl* is trained on each of these samples. The regression simply averages results from the t resulting models.

Breiman (1996) discusses, that this procedure is especially useful for learners *bl* which are unstable — small perturbations in the data have big impact on the resulting model. Aggregating the bootstrapped models essentially introduces robustness to such models. Examples of learners where the bagging is beneficial are neural networks (where overfitting is often a serious problem) and regression trees (especially variants without pruning) — Random Forests presented beneath are essentially bagged tree learners.

On the other hand, it needs to be said that bagging can worsen the performances of learners that are stable.

4.2.2 Stacking

The *stacking* (or stacked generalization) is a more sophisticated approach. The original idea was pioneered by Wolpert (1992). The method is basically a two level hierarchical model of learners with a clever scheme for training. The first level is composed by an ensemble of (possibly different) learners. The second level is a single learner which aggregates guesses from the 1st level models and outputs the final prediction. Figure 4.2 shows the topology.

The training dataset is divided into smaller parts (by cross-validation, see Section 4.4.1). The 1st level learners are trained on some of them and their *generalization* biases are measured by testing their performance on the rest. The

[3]The name bagging stands for **b**ootstrapped **agg**regat**ing**.

2nd level learner learns to correct these — it learns what the correct output is, given what the 1st level predictors output. Algorithm 5 hopes to make the procedure clear.

Having different base learners often proves to be effective. The performance of stacking is usually better than the best of the base learners on its own. It is not the case, however, that having badly performing learners in the ensemble does not worsen the performance. Choosing the right set of 1st level learners is very important if we are to attain the best performance, as is the choice of the 2nd level aggregating learner and the number of folds for the cross-validation step. We discuss this matter in Section 4.3.

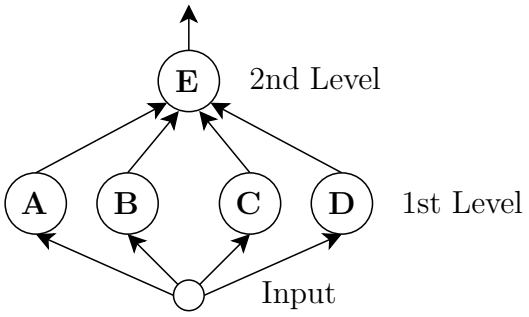


Figure 4.2: The topology of the stacking ensemble method. **A**, **B**, **C** and **D** are the level 1 learners, **E** is the level 2 learner.

Algorithm 5: Stacking

input : an ordered set of 1st level learners *ensemble*, a level 2 learner $l2$, training data Tr, number of folds $Folds$
output: regression function f

`/* Training set for the level 2 learner. */`
$L2Tr \leftarrow \{\}$;
foreach (Tr', Ts') *in* `CrossValidation(`Tr, $Folds$`)` **do**
  `/* The level 1 learners trained on split` Tr'. `*/`
  $L1 \leftarrow (ensemble_0(Tr'), \ldots, ensemble_n(Tr'))$;
  foreach (x', y') *in* Ts' **do**
    `/* Responses of level 1 predictors to unseen` x' `and the real reply` y'. `*/`
    $L2Tr \leftarrow L2Tr \cup \{((L1_0(x'), \ldots, L1_n(x')), y')\}$;
  end
end
`/* Train the level 1 learners on the real data. */`
$L1 \leftarrow (ensemble_0(Tr), \ldots, ensemble_n(Tr))$;
`/* Train the level 2 learner on the prepared data. */`
$L2 \leftarrow l2(L2Tr)$;
return `Compose(`$L1$, $L2$`)`;

4.2.3 Random Forests

Random Forests (Breiman, 2001) utilize an ensemble of tree learners to predict the dependent value (for an overview of the regression trees, see (Breiman et al., 1984)).

Each tree from the forest (of size N) is trained on an independently chosen subset of training data, exactly as the bagging in Section 4.2.1 does.

However, there is one tweak of the process of learning one tree. During each training step a random subset of attributes is chosen, and the tree node is split on the best attribute of this subset. See the Breiman's paper for details.

The aggregation step is the same as in the bagging, simply averaging the outputs of the trees in the forest.

4.3 Choosing the Best Stacked Ensemble – Genetic Algorithm

We have discussed that ensemble learning might be beneficial in terms of performance. For stacking, it is desirable to form the ensemble out of diverse base learners. The problem however is, how to choose the learners into the ensemble. This becomes apparent once one tries to hand-tune the parameters of different base-learners, find the best combination of them and find the best aggregating 2nd level learner.

We have used a simple genetic algorithm (GA) to search the space of possible ensembles for the stacking. Genetic algorithms are an universal optimization tool, see (Whitley, 1994) for a good tutorial. The general procedure is iterative. In each iteration, individuals (candidate solutions) are evaluated using a *fitness function* and an intermediate population is formed by randomly choosing individuals, with probability proportional to the fitness (roulette selection). From this intermediate population, the population for the next step is taken by making pairwise *crossover* operation and mutation on the newly formed individuals.

In the text below, we operate with a set of base learners BL, from which we choose the learners into the ensemble. We should note that the set of base learners BL is not strictly limited to learners we have listed as base in Section 4.1 — we use both differently parameterized base learners and various bagged learners (neural networks and forests).

We have used a very simple **encoding for an individual**. An individual is a triple of $(I, Folds, \vec{v})$. The first two values I and $Folds$ define the 2nd level learner. I is the index of the 2nd level aggregating learner in BL and $Folds$ is the number of folds for the stacking procedure. The vector $\vec{v}$ of size $|BL|$ marks a subset of BL that forms the ensemble: $\vec{v}_i = 1$ if the base learner BL_i belongs to the ensemble; $\vec{v}_i = 0$ when it does not.

We have used two independent **mutations** to modify the individuals. Firstly, with probability Pm_M, we either change I to any of $1 \ldots |BL|$, or we change the number of $Folds$ to $2 \ldots 6$, (`MutateM` in the pseudocode). Whether we change I, or $Folds$ is decided using a further random coin toss. Secondly, with probability Pm_v a random position i in v is selected and the bit v_i is swapped, (`MutateV` in the pseudocode).

The **crossover operation** of parents $P = (I, Folds, \vec{v})$ and $P' = (I', Folds', \vec{v}')$ selects a random position $i \in \{1 \dots |BL|\}$ and outputs the following tuple

$$(I, Folds, (v_1, \dots, v_i, v'_{i+1}, \dots, v'_{|BL|}))$$

as the new individual. Please note that the index I (and number of $Folds$) of the 2nd level learner is taken from the first parent P. This is compensated for by the fact that crossover is always performed in pairs (lines 8 – 9 in Algorithm 6).

The **fitness** function we have used is inversely proportional to $RMSE$ error of the resulting stacked ensemble.

Also, to make sure we do not lose the best solution, we have used elitism, which brings the top E individuals unchanged into the next generation.

Algorithm 6: Genetic Algorithm for finding optimal stacking ensemble

input : size of the population S, size of the elite E, probabilities of mutation Pm_M and Pm_v, maximal number of steps Max
output: The best individual.

```
1  Pop ← RandomPopulation(S);
   /* The best individual so far.                                   */
2  Best ← {};
3  foreach iteration in 1 ... Max do
4  |  evaluation ← Fitness(Pop);
   |  /* PI is the intermediate population.                        */
5  |  PI ← RouletteSelection(Pop, evaluation);
   |  /* PN is the intermediate population after Crossover.        */
6  |  PN ← {};
7  |  foreach i in 1 ... (S − E)/2 do
8  |  |  PN ← PN ∪ Crossover(PI[2 ∗ i], PI[2 ∗ i + 1]) ;
9  |  |  PN ← PN ∪ Crossover(PI[2 ∗ i + 1], PI[2 ∗ i]) ;
10 |  end
   |  /* Save the best individual.                                 */
11 |  Best ← TakeTop(Pop, evaluation, 1);
   |  /* Top E best continue unchanged.                           */
12 |  Pop ← TakeTop(Pop, evaluation, E);
13 |  foreach individual in PN do
14 |  |  if Rnd(0,1) < Pm_M then
15 |  |  |  individual ←MutateM(individual);
16 |  |  end
17 |  |  if Rnd(0,1) < Pm_v then
18 |  |  |  individual ←MutateV(individual);
19 |  |  end
20 |  |  Pop ← Pop ∪{individual};
21 |  end
22 end
23 return Best;
```

4.4 Evaluating Learners

To compare performances of different regression functions (learners), we need a reliable metric. The goal is to estimate the performance of a particular regression function on real unseen data. We can estimate this performance by splitting the data we have into parts that are only used for training (Tr) and testing (Ts).

4.4.1 Cross-Validation

Cross-validation is a standard statistical technique for estimation of parameters. The idea is to split the data into k disjunct subsets (called *folds*), and then iteratively compose the training and testing sets and measure errors. In each of the k iterations, k-th fold is chosen as the testing data, and all the remaining $k-1$ folds form the training data. The division into the folds is done randomly, and so that the folds have approximately the same size (in cases where the number of samples $|D|$ is not divisible by k, some folds are slightly smaller than others). Please note that each sample from the data is a part of the testing fold exactly once (it is part of a training set $k-1$ times).

Refer to (Kohavi, 1995) to learn more.

4.4.2 Error Analysis

To evaluate a regression function r reliably, we are looking for a robust error measure. Commonly, the *mean square error* is used:

$$MSE(r) = \frac{1}{|Ts|} \sum_{(x,y) \in Ts} (r(x) - y)^2$$

Where r is trained on the training data Tr. The MSE is an estimate of variance of the population of errors.

In this work, we have used the MSE's square-root, $RMSE$, which is an estimate of standard deviation of the errors. Because the square root is a monotonically increasing function, sorting the learners based on MSE and $RMSE$ yields the same order. Moreover, the $RMSE$ has a clear interpretation — under the assumption of normality of the distribution of errors with zero mean, confidence intervals on precision of the regression function r can be given.

$$P(-\sigma \leq y - y' \leq \sigma) = \Phi(1) - \Phi(-1) \approx 0.6827$$

$$P(-2\sigma \leq y - y' \leq 2\sigma) = \Phi(2) - \Phi(-2) \approx 0.9545$$

$$P(-3\sigma \leq y - y' \leq 3\sigma) = \Phi(3) - \Phi(-3) \approx 0.9973$$

Where Φ is the cumulative distribution function of the standard normal distribution. For instance, we can say that with the probability of 95%, a prediction $y' = r(x)$ for a feature vector x is within a range of 2σ from the true value y. Of course, these estimates *only* hold when the training and testing data are sampled reliably (e.g. with many-fold cross-validation) *and* even more importantly, when the dataset reflects the real distribution of the problem's data.

4.5 Evaluating Features and Attributes

We have devised a number of features that try to capture information from a set of games. Each of them is based on different rationale and assumptions. A numerical measure of their performance (*feature evaluation*) is beneficial from two main points of view.

Firstly, evaluating the features tests whether (and how much) do the particular assumptions hold. Apart from being useful on its own, knowing what can be assumed about data gives directions for further improvement. The second, rather practical, benefit is that this feature evaluation allows to search for the best parameters of the feature extractors.

Besides the features as a whole, we can also analyze the performance of particular attributes[4]. An analysis of attributes might be useful in numerous applications. For instance, if we find out that there is a linear dependency between attribute of playing a particular move and the strength of a player, we could warn the user: "this move is usually not very good". This might also have applications in computer Go, as we discuss in Chapter 6. Of course, correlation does not imply causation and simply playing the "good" move more often does not make us really stronger.[5] Still, the dependencies give hints about some deeper imbalance in moves one plays.

4.5.1 Feature Evaluation

We have used a simple scheme for feature evaluation. The assumption of this scheme is that we are interested in features which perform good. We define the performance of the feature to be the performance of a fixed learner (the same for all the features). Of course, the learner has to be able to benefit from the usefulness of the features.

Definition. *For a fixed learner l_{ev} the RMSE error of feature extractor f on data:*

$$D = \{(GC_i, y_i), \ldots\}$$

is defined as the RMSE error of l_{ev} on

$$T = \{(f(GC), y), \ldots\}, (GC, y) \in D$$

4.5.2 Attribute Evaluation

To analyze performances of single attributes, we have used the following scheme. For the k-th attribute, we inspect the dependency between its values x_{ik} (value of kth attribute in the ith input vector) and the target variable y in the data $Tr = \{(x_i, y_i)\}, i = 1 \ldots N$.

$$X_k = (x_{1k}, \ldots, x_{Nk})$$

[4]Remember from Chapter 3, that we distinguish between *features* and *attributes*. Example of features are the pattern features, local sequence feature or the histogram features. An attribute is a particular number v_i, where v is a feature vector. For example, if v is a particular pattern feature vector, v_i is an attribute giving relative frequency of pattern i.

[5]The problem of course is that the "good" move is not good under all circumstances and one should rationalize *why* and *when* is it so.

$$Y = (y_1, \ldots, y_N)$$

The linear dependency between X and Y is measured by the Pearson's correlation coefficient r, (Rodgers – Nicewander, 1988):

$$r = \frac{\mathrm{cov}(X, Y)}{\sigma_X \sigma_Y}$$

Pearson's r has a range of $\langle -1, 1 \rangle$, where 1 means perfect linear dependence (y's grow with growing x_k's), 0 means no linear dependence and -1 signals that both vectors are anticorrelated (y's decrease with growing x_k's). In the following text, we shorten the $r_{X,Y}$ to r when the variables X and Y are clear from the text.

Limiting ourselves to *linear* dependencies is not a major issue. Because the problem is quite hard, the dependencies are rather weak even with the simplest linear model. If we were to test more complex dependencies, much more data would be needed.

5. Experiments

5.1 Strength

The major application of the methodology developed throughout the thesis is the prediction of strength of players. This part documents the process we undertook. Firstly, we present the dataset we have gathered (Section 5.1.1). Then, performance of different feature extractors (Section 5.1.2) is analyzed. Next, we investigate possibilities of strength prediction (Section 5.1.3). Finally, we scrutinize single attributes and their relationship with the strength (Section 5.1.4).

5.1.1 Dataset

We have collected a large sample of games from the publicly available archives of the Kiseido Go server (Shubert, 2013b). The sample consists of over 100 000 records of games in the *.sgf* format (Hollosi, 2006).

For each rank r in the range of 6-dan to 20-kyu, we gathered a list of players P_r of the particular rank. To avoid biases, the sample only consists of games played on 19×19 goban without handicap stones.[1] The set of colored games GC_p for a player $p \in P_r$ consists of the games player p played when he had the rank r. We only use the GC_p if the number of games is not smaller than 10 games. Similarly, if the player played more than 50 games when at rank r, we randomly sampled k of them, where k was uniformly randomly chosen from interval $\langle 10, 50 \rangle$.[2] The number of games is limited in this manner because for some ranks it is hard to find players with large samples — i.e. weak players and beginners (e.g. on 20-kyu) usually improve very fast. For each of the 26 ranks, we gathered 120 such GC_p. The distribution of number of games in GC_p is comparable for all the ranks, as Figure 5.1 shows. The target variable for regression y directly corresponds to the ranks: $y = 20$ for rank of 20-kyu, $y = 1$ for 1-kyu, $y = 0$ for 1-dan, $y = -5$ for 6-dan, other values similarly. (With increasing strength, the y decreases.)

5.1.2 Feature Evaluation

We evaluated the performance of various features as discussed in Section 4.5.1. We have used the initial hand tuned learner (from Appendix C.3) as the evaluation learner l_{ev}.[3] We have evaluated many different parameterisations for different features, as detailed in Appendix C.1. Table 5.1 shows the best parameters found for each feature extractor along with the $RMSE$ scores.

[1] Gameplay and strategies on different board sizes differs. Similarly, handicap games force the stronger player to play more aggressively than he would in an even game.

[2] By cutting the number of games to a fixed number (say 50) for large samples, we would create an artificial disproportion in sizes of GC_p, which could introduce bias into the process.

[3] With the exception of the Win/Loss points statistic, where we changed the number of components of PLS regression to 2 (from 3 in l_{ev}). This was needed because this feature has small dimension (2) which causes instability of the PLS with 3 latent variables.

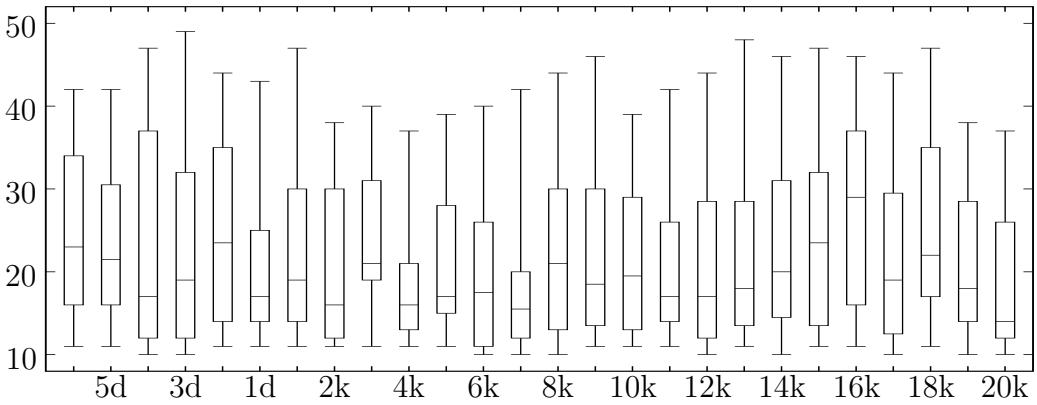


Figure 5.1: Boxplot of game sample sizes. The box spans between 25th and 75th percentile, the center line marks the mean value. The whiskers cover 95% of the population. The kyu and dan ranks are shortened to k and d.

Feature Extractor	**RMSE**	**Parameters**
Pattern feature	2.755	$N = 1000$, independent normalization, A randomly sampled as 20% of all the games.
Local sequences	5.754	$\omega = 10$
Border distance	5.448	$ByDist = \{\langle 1,2\rangle, \langle 3\rangle, \langle 4\rangle, \langle 5,\infty)\}$, $ByMoves = \{\langle 1,10\rangle, \langle 11,64\rangle, \langle 65,200\rangle, \langle 201,\infty)\}$
Captured stones	5.878	$ByMoves = \{\langle 1,60\rangle, \langle 61,240\rangle, \langle 241,\infty)\}$
Win/Loss statistics	6.806	—
Win/Loss points	5.158	— (see Footnote 3 on page 33)
None	7.507	(obtained by mean regression learner)

Table 5.1: Comparison of the best feature extractors of each kind. The complete list of features evaluated is given in Appendix C.1. The learner used for evaluation is given in Appendix C.3. The results were computed using 5-fold validation. The last row shows performance of the mean regression learner and serves for comparison.

5.1.3 Regression

In the process of finding the best learner, we started with a hand-tuned learner shown in Appendix C.3. Using this learner (which we found to perform reasonably well, as shown in Table 5.3) we evaluated different feature extractors (previous section). At first the dataset was processed using the best feature extractors, which were concatenated to form the data T for regression.

We then used the genetic algorithm (Section 4.3; abbreviated to *GA*) to find the best performing stacked ensemble. The initial population was seeded by the hand tuned learner. The parameters of the genetic algorithm are given in Table 5.2.

Unfortunately, because of the large dimension of the feature vector (especially the pattern feature which has dimension of 1000 in the best setting) and large dataset (3120 samples), the time needed for a single iteration was very large in this setting.[4] To speed up the process, we used a smaller pattern feature size

[4]More than 15 hours for first iterations using parallel evaluation on Intel i3 - 4 core machine

Parameter	Value
Set of base learners BL	Is given in Appendix C.2.
Population size S	16
Elite size E	1
Number of iterations Max	100
Mutation probability Pm_I	0.2
Mutation probability Pm_v	0.5
Fitness function	$1/RMSE$ of the resulting stacked learner. The $RMSE$ is computed using 5-fold cross-validation.

Table 5.2: The parameters of the genetic algorithm for the strength dataset.

Learner	RMSE	Mean cmp
Mean regression	7.507	1.00
Random Forrest	3.869	1.94
PLS	3.176	2.36
Bagged NN	2.701	2.78
Initial hand-tuned learner	2.635	2.85
Best GA stacking ensemble	2.607	2.88

Table 5.3: Regression performance of different learners on the full dataset. The feature set used is shown in Table 5.1. The results were computed using 5-fold cross-validation. Parameters of the best GA stacking ensemble are given in Appendix C.4, the other learners are taken from the Initial hand-tuned learner from Appendix C.3.

(400 top patterns instead of 1000) and we subsampled the dataset for computing the fitness during the GA (by randomly taking 1/10 prior to the running of the GA). We assume, that this is not a principial obstacle for finding the best learner, since the down-sampling (and lowering the precision of the pattern feature) should degrade the performance of the learners *systematically* — the ordering of fitnesses is expected to be more or less the same, though the fitness values surely differ. This is backed by the fact, that the best learner found by the GA scored very well when run on the full dataset. The run of genetic algorithm took on average approximately 2.5 hours per iteration, the machine specification is given in Appendix C.6.

The performance of the best ensemble found by the GA (on the full dataset) are given in Table 5.3 along with other learners to compare performances. The resulting learner (Appendix C.4) is fairly complex as Figure 5.2 shows. Evolution of the **RMSE** error in time is given in Figure 5.3.

5.1.4 Attribute Evaluation

We analysed different attributes for the best features from the previous section using the methodology presented in Section 4.5.2. We studied attributes, which were most strongly correlated with the strength of the player.

at 2.3GHz.

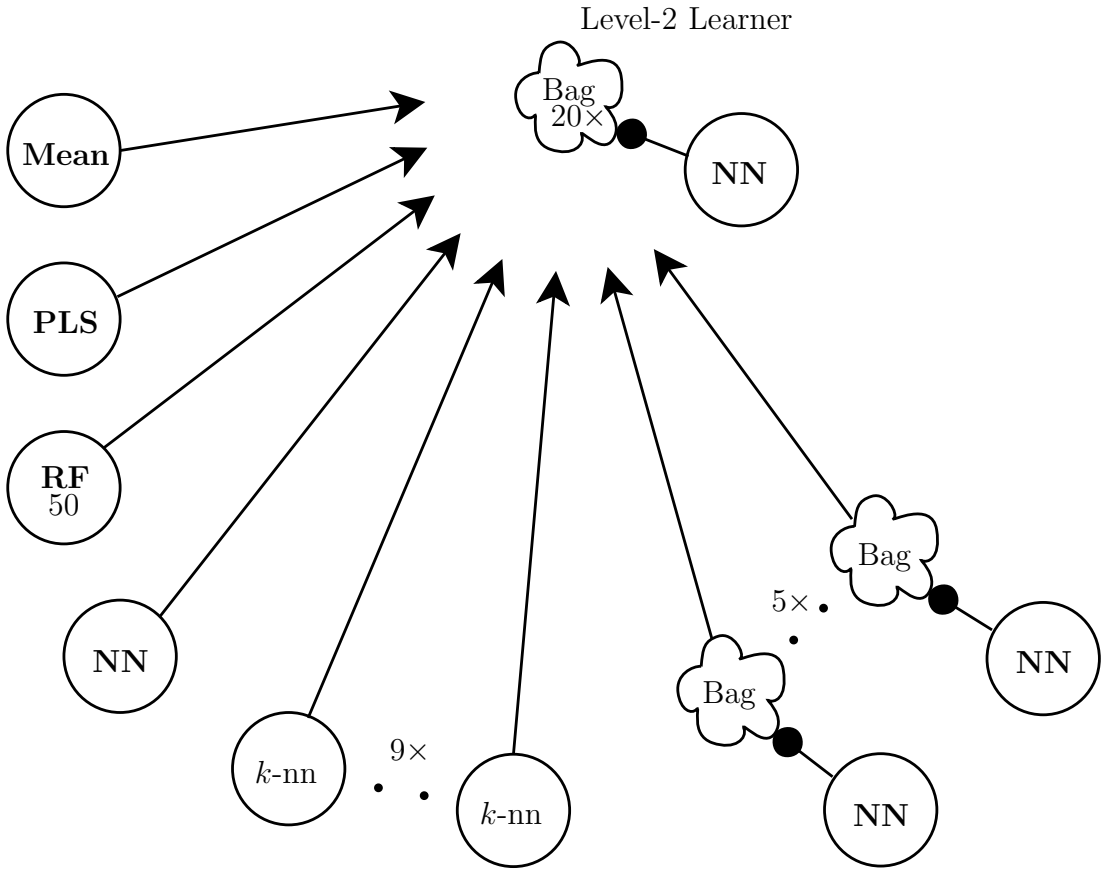


Figure 5.2: Structure of the best stacking ensemble found by the Genetic Algorithm. The circle marks a normal learner, with description within, the "cloud" denotes a bagging learner. The corresponding bagged learner is connected using the circle-ended arrow. Precise descriptions of the learners are given in Appendix C.4. **Mean** is the Mean regression, **PLS** Partial least squares regression, **RF** Random Forrests, **NN** various neural networks and k-nn is obviously the k-nearest neighbor learner.

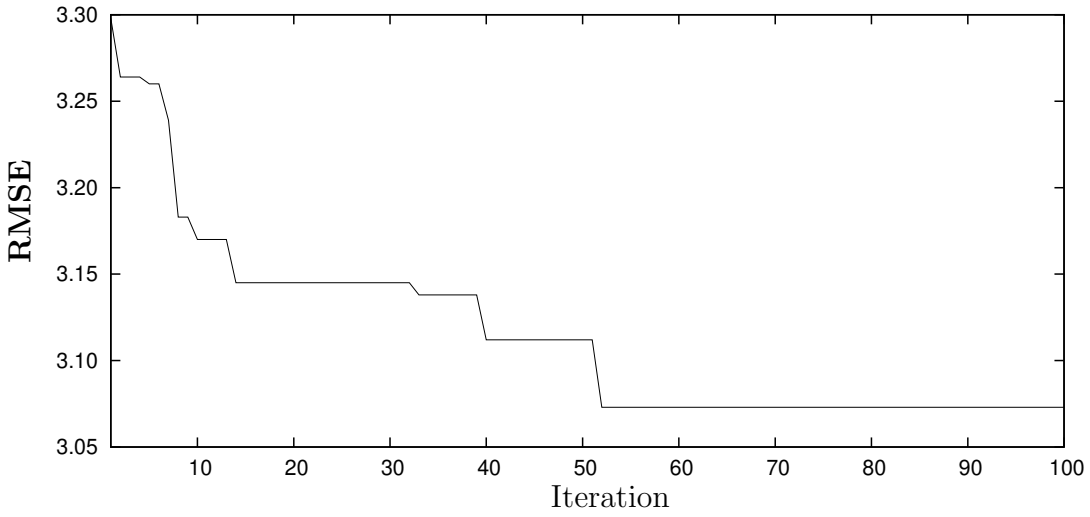


Figure 5.3: Evolution of **RMSE** error during the run of the genetic algorithm for finding an optimal stacking ensemble for the strength data.

In the following text, we present a few attributes with largest Pearson's correlation coefficient r along with general trends, more detailed list is given in Appendix D.

Negatively correlated attributes ("strong players' attributes")

The attributes most strongly correlated with the increasing strength of player ($r < 0$), are mainly a standard patterns with clear strategic meaning, or a part of *joseki sequence*[5].

The most strongly correlated attribute is the local sequence statistic, the average number of sente moves per game ($r = -0.512$). This backs the assumption that the concept of sente is indeed very important.

The second most strongly correlated attribute ($r = -0.480$), the pattern shown in Figure 5.4 (left), is a multi-purpose one-point jump. It can be found being played under different circumstances. For instance, Black might want to prevent possible invasion (which could be related to the white stone). The pattern also matches a common move of expanding one's territory, while preventing the opponent to block this expansion.

The third strongest attribute ($r = -0.457$), is the difference between number of sente and gote sequences. The strength of this attribute is probably caused by the fact that the number of sente sequences itself is very strong.

The next two attributes are the patterns shown in Figure 5.4 (middle and right). Both moves have a similar context — they are usually played on the boundaries of competing forces. Such moves are usually of crucial importance. For example, the horse move (*keima*) pattern (in the middle, $r = -0.455$) usually prevents White from foiling Black's future development by jumping into what probably is a potential Black's territory.

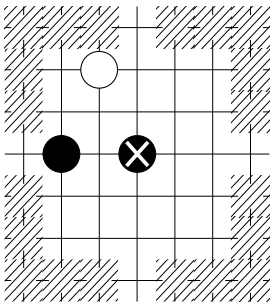
The pattern on the right ($r = -0.446$), shows a one-point jump which tries to get ahead of White and thus prevent White's possibility of shutting Black to the side (without the marked black stone, White could play at **a**, which would probably be unbearable for Black).

Positively correlated attributes ("weak players' attributes")

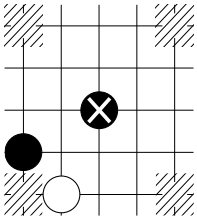
On the other hand, the most strongly positively correlated attributes ($r > 0$) are the patterns that exhibit defects or inefficiencies in shape. The best example of this is the empty triangle shape (Sensei's Library, 2013d), as shown in Figure 5.5 (left). The fact that the two strongest ($r = 0.437$ and $r = 0.402$) bad shapes are empty triangles backs the commonsense taught to beginners ("Do not play the empty triangle"). (The second strongest empty triangle pattern has almost the same configuration of stones as the first one, with the exception that the stone Ⓑ is not present.)

As expected, the beginners also tend to capture unnecessary stones — the third strongest "weak" attribute ($r = 0.377$) was the number of captured stones within first 60 moves. The most likely reason for this is that beginners are not able to discern between important and unimportant stones and they tend to capture because "they could" instead of because "it is the best move".

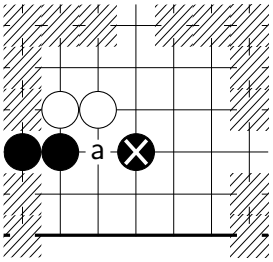
[5]The *joseki* are standard sequences of moves which ensure even result for both players. The joseki are mainly played in the opening and mainly regard corners and their approaches.



$r = -0.480$, a standard one-point jump, preventing White from capping.

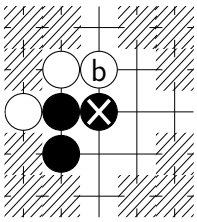


$r = -0.455$, a multi-purpose horse move.

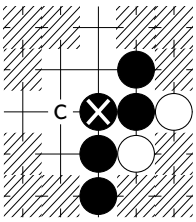


$r = -0.446$, Black is getting ahead; otw. White could play at **a**.

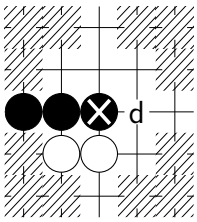
Figure 5.4: Top 3 negatively correlated patterns ("good moves").



$r = 0.437$, an empty triangle. The second pattern ($r = 0.402$) does not have (b) present.



$r = 0.351$, an overly solid connection.



$r = 0.325$, pushing from behind.

Figure 5.5: Typical positively correlated patterns ("bad moves").

The next attribute is a solid connection ($r = 0.351$) depicted in the middle of Figure 5.5. It looks like Black is trying to protect from the cut at ⊗. Were the cut really severe (and this is not sure), it would probably be a better idea to connect at **c** (so that Black has a better eye-space), but we cannot give further interpretation without seeing the rest of the board.

We should also mention another strongly correlated pattern which came up during the analysis and which is also very important. It is the *pushing from behind* pattern ($r = 0.325$) shown in Figure 5.5 (right). Generally, pushing from behind is bad because it allows the opponent to play at **d**, which is a very good move — it limits Black's future development, while extends White's. Instead of ⊗, Black could e.g. play the horse move (Figure 5.4, in the middle), or play somewhere else.

5.2 Style

Apart from the strength estimation, we also used the framework presented in this work to test prediction of playing styles for professional players. The organization of the text is similar to that of the strength. Firstly, we give details of the dataset (Section 5.2.1). Next, we investigate performance of different feature extractors (Section 5.2.2). Using the best features found, we explore possibilities for prediction of the styles (Section 5.2.3). Finally, we analyze the single attributes and their relation to different styles (Section 5.2.4).

5.2.1 Dataset

The collection of games in this dataset comes from the Games of Go on Disk (*GoGoD*) database by Hall – Fairbairn (winter 2011a). This database contains more than 70 000 games, spanning from the ancient times to the present.

We chose a small subset of well known players (mainly from the 20th century) and asked some experts (professional and strong amateur players) to evaluate these players using a questionnaire. [6] The experts (Alexander Dinerchtein 3-pro, Motoki Noguchi 7-dan, Vladimír Daněk 5-dan and Vít Brunner 4-dan) were asked to value the players on four scales, each ranging from 1 to 10.

Style	**1**	**10**
Territoriality	Moyo	Territory
Orthodoxity	Classic	Novel
Aggressivity	Calm	Fighting
Thickness	Safe	Shinogi

The scales try to reflect some of the traditionally perceived playing styles.[7] For example, the first scale (*territoriality*) stresses whether a player prefers safe, yet inherently smaller territory (number 10 on the scale), or roughly sketched large territory (*moyo*, 1 on the scale), which is however insecure (we describe the scales in more details below). Table 5.5 shows the data obtained from the questionnaire. The mean standard error of the answers is 1.164, which we regard as reasonably consistent. Table 5.4 shows mean values of answers (across all the players) for the styles along with standard deviation. On the right, the pairwise correlation of the styles from the questionnaire is given (measured by the Pearson's r).

—		**Pearson's r**			
Style	**Mean value**	Ter.	Orth.	Aggr.	Thick.
Territoriality	5.670 ± 2.390	1.000	-0.526	-0.602	0.566
Orthodoxity	5.861 ± 2.415		1.000	0.738	-0.072
Aggressivity	6.722 ± 2.176			1.000	0.124
Thickness	4.903 ± 1.667				1.000

Table 5.4: The mean values of styles (across all the answers) and the pairwise correlation between them.

[6] Part of the data was reused from our previous work (Baudiš – Moudřík, 2012).

[7] Refer to Fairbairn (winter 2011), or Sensei's Library (2013j) to grasp the concept deeper.

Player	Territoriality	Orthodoxity	Aggressivity	Thickness
Chen Yaoye	6.0 ± 1.0	4.0 ± 1.0	6.0 ± 1.0	5.5 ± 0.5
Cho Chikun	9.0 ± 0.7	6.2 ± 2.6	6.8 ± 1.1	9.0 ± 0.7
Cho U	7.2 ± 2.0	5.2 ± 1.5	6.0 ± 1.9	6.2 ± 1.5
Fujisawa Hideyuki	3.5 ± 0.5	9.0 ± 1.0	7.0 ± 0.0	4.0 ± 0.0
Go Seigen	6.0 ± 2.0	9.0 ± 1.0	8.0 ± 1.0	5.0 ± 1.0
Gu Li	6.2 ± 1.3	7.8 ± 1.5	9.2 ± 0.8	5.0 ± 1.9
Hane Naoki	7.5 ± 0.5	2.5 ± 0.5	4.0 ± 0.0	4.5 ± 1.5
Ishida Yoshio	8.5 ± 1.5	4.0 ± 2.1	3.0 ± 1.2	4.8 ± 1.1
Kato Masao	3.0 ± 0.8	3.7 ± 1.7	8.7 ± 1.2	5.7 ± 2.4
Kobayashi Koichi	9.3 ± 0.9	2.0 ± 0.8	2.7 ± 0.5	4.3 ± 1.7
Luo Xihe	7.3 ± 0.9	7.3 ± 2.5	7.7 ± 0.9	6.0 ± 1.4
Ma Xiaochun	8.2 ± 1.9	5.2 ± 1.9	5.2 ± 1.8	6.8 ± 2.3
Miyazawa Goro	1.5 ± 0.5	10.0 ± 0.0	9.5 ± 0.5	4.0 ± 1.0
O Meien	2.7 ± 1.2	9.7 ± 0.5	8.3 ± 1.7	3.7 ± 1.2
Otake Hideo	4.5 ± 0.5	2.5 ± 0.9	4.2 ± 1.3	3.2 ± 1.1
Rui Naiwei	5.5 ± 1.8	5.5 ± 0.5	9.0 ± 0.7	4.0 ± 1.6
Sakata Eio	8.0 ± 1.6	4.0 ± 1.2	7.8 ± 1.1	8.2 ± 1.5
Takao Shinji	5.0 ± 1.0	3.5 ± 0.5	5.5 ± 1.5	4.5 ± 0.5
Takemiya Masaki	1.5 ± 0.5	5.8 ± 2.0	7.2 ± 0.8	1.8 ± 0.8
Yamashita Keigo	2.0 ± 0.0	9.0 ± 1.0	9.5 ± 0.5	3.0 ± 1.0
Yi Ch'ang-ho	7.5 ± 1.8	5.2 ± 1.9	3.8 ± 1.8	3.5 ± 0.5
Yi Se-tol	6.0 ± 1.2	7.2 ± 2.3	9.2 ± 0.4	7.2 ± 1.5
Yoda Norimoto	7.0 ± 1.9	3.8 ± 2.0	4.0 ± 1.9	3.2 ± 1.1
Yuki Satoshi	3.0 ± 1.0	8.5 ± 0.5	9.0 ± 1.0	4.5 ± 0.5

Table 5.5: Expert-based evaluation of styles of selected Professionals, including standard deviation of their answers. Only the players that were evaluated by two or more experts out of four are included.

For each of the professional players, we took 192 of his games from the GoGoD database at random.[8] We divided these games (at random) into 12 colored sets GC of 16 games. For each player, we have one target variable y for each style — basically, we view the problem as 4 different regression problems which share the feature extraction.

[8]We chose this number because the database does not contain more games for some of the players.

Feature Extractor	RMSE	Parameters
None	2.403	(obtained by mean regression learner)
Pattern feature	1.558	$N = 600$, linear normalization, A randomly sampled as 20% of all the games.
Local sequences	2.267	$\omega = 5$
Border distance	1.663	$ByDist = \{\langle 1,2\rangle, \langle 3\rangle, \langle 4\rangle, \langle 5,\infty)\}$, $ByMoves = \{\langle 1,16\rangle, \langle 17,64\rangle, \langle 65,160\rangle, \langle 161,\infty)\}$
Captured stones	2.381	$ByMoves = \{\langle 1,40\rangle, \langle 41,160\rangle, \langle 161,\infty)\}$
Win/Loss statistics	2.362	—
Win/Loss points	2.415	— (see Footnote 9 on page 41)

Table 5.6: Comparison of the best feature extractors of each kind. The complete list of features evaluated is given in Appendix C.1. The learner used for evaluation is given in Appendix C.3. The results were computed using 5-fold validation by averaging $RMSE$ of all the styles. The last row shows performance of the mean regression learner and serves for comparison.

5.2.2 Feature Evaluation

We evaluated different features from Appendix C.1 similarly as we did with the strength data — we have used the same initial hand tuned learner (Appendix C.3) as the evaluation learner l_{ev}.[9] Because the style regression essentially consists of four different regression problems (one for each style) we could perform the analysis independently. Doing so would have several drawbacks — most importantly the features would not be easily comparable and the whole process of feature extraction would be slowed down four times.

We therefore decided to evaluate the features regarding *all* the styles. For a given feature, we used the l_{ev} learner to learn four style regression problems, the final $RMSE$ was computed as an average of the $RMSE$ errors from the subproblems. The $RMSE$ errors for subproblems were analyzed using 5-fold cross-validation with the same random seed (which means that the splits were the same for all the styles). The results are given in Table 5.6.

5.2.3 Regression

Using the concatenation of the best feature extractors from previous section, we processed the data. Then, we used the genetic algorithm to determine the best ensemble learner.

During the process, we have encountered over-fitting problems concerning the size of the dataset.

At first, we chose the parameters of the GA to be the same as in the problem of strength (Table 5.2), with the exception of the fitness function. The $RMSE$ error was computed in the same manner as in the style feature extraction (one learner for all the styles, the fitness of a learner is average $RMSE$ on the different styles). Similarly to the case of strength, it turned out that it was not possible

[9] Again with the exception of the Win/Loss points statistic, where we changed the number of components of PLS regression to 2 (from 3 in l_{ev}). This was needed because this feature has small dimension (2) which causes instability of the PLS with 3 latent variables.

to use cross-validation on the full dataset because of time constraints. We tried to workaround this by subsampling the data prior to the experiment, but due to relatively small size of the dataset, this resulted in over-fitting of the resulting ensemble model.

Secondly, we tried not to use the cross-validation, but to use proportional division scheme instead — the fitness is evaluated by randomly taking 70% of the dataset for training and the rest for testing; in each of the iterations, this is done anew to mitigate any effects caused by biased random split (dividing the dataset once prior to the run would cause over-fitting). Unfortunately, this too did not yield satisfactory results. Even though over-fitting was not the case, the genetic algorithm was not able to consistently improve the ensembles — the subsampled datasets in each of the iterations were too different to ensure that the best individiuals from one iteration would have good chances in the next one. This rendered the genetic algorithm unsucessful.

Consequently we concluded, that the robust cross-validation with the full dataset is necessary and that we thus need to compensate for the increased resource consumption differently. We did this by limiting the population size to 10 individuals and most importantly, by limiting the ensemble to contain at most 5 base learners. Technically, this is done by randomly removing excess number of base learners from each individual at the end of each iteration. Additionally, we decided to run the GA independently for each of the styles, instead of optimizing one ensemble learner for all the styles (as above). The parameters of the final genetic algorithm are listed in Table 5.7.

The performances of the best learners found are given in Table 5.8 and the learners are listed in Appendix C.5. Development of the **RMSE** error in time is given in Figure 5.6. Each run of the genetic algorithm (for different styles) took approximately one hour of CPU time per iteration, the machine specification is given in Appendix C.6.

Parameter	**Value**
Set of base learners BL	Is given in Appendix C.2.
Population size S	10
Elite size E	1
Number of iterations Max	100
Mutation probability Pm_I	0.2
Mutation probability Pm_v	0.5
Ensemble size limit	5
Fitness function	$1/RMSE$ of the resulting stacked learner. The $RMSE$ is computed using 5-fold cross-validation.

Table 5.7: The parameters of the genetic algorithm for the style dataset.

5.2.4 Attribute Evaluation

Following the same procedure we used in strength attribute evaluation, we scrutinized the dependences between the styles and different attributes. The results have some significant properties. Generally, the opening moves (which in sense form the shape of the following game) have high importance, as do some other

	RMSE			
Learner	Territoriality	Orthodoxity	Aggressivity	Thickness
Mean regression	2.403	2.421	2.179	1.682
Initial hand tuned l.	1.434	1.636	1.423	1.484
The best GA learner	1.394	1.506	1.398	1.432

Table 5.8: Regression performance of different learners on the full dataset. The feature set used is shown in Table 5.6. The results were computed using 5-fold cross-validation.

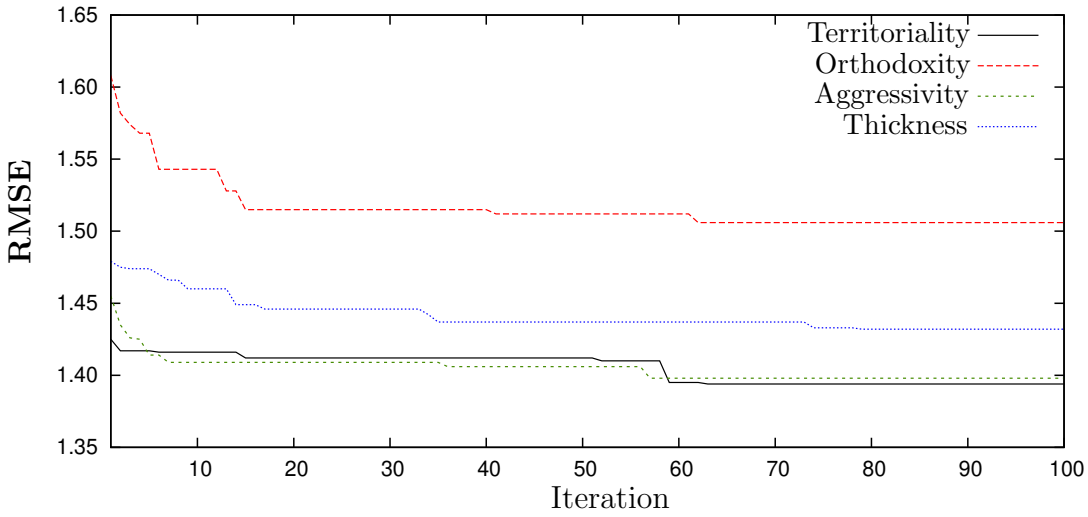


Figure 5.6: Evolution of **RMSE** error during the run of the genetic algorithm for finding an optimal stacking ensemble for the style data.

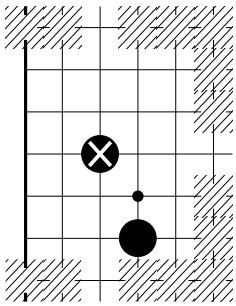
typical formations. Moreover, because the styles themselves are correlated with each other (relatively strongly – Table 5.4), the results for different styles are often related, as illustrated in the following text.

Territoriality

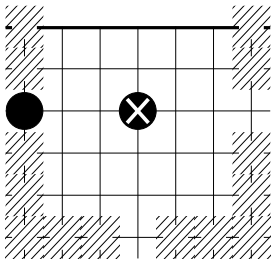
The scale of territoriality spans from the style which emphasizes moyo-based, influence style of game (number 1 on the scale) and a style which stresses safe territory on the other side (number 10 on the scale). Regarding correlations of the attributes, this corresponds to positive Pearson's $r > 0$ for the territorial style and $r < 0$ for players preferring moyos.

The attributes seem to capture this scale exceedingly well, the correlations are strong and have clear interpretations. In the opening (first 16 moves), increasing territoriality is revealed by preference of third line moves ($r = 0.621$, the border distance feature), while the moyo style is most strongly correlated ($r = -0.555$) by playing to 5th line or higher. In line with the common knowledge, playing on the fourth line in the opening (again, the border distance feature) also correlates with moyo-based style of the game ($r = -0.530$).

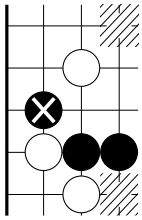
The pattern attributes seem to reveal the same information. The strongest territory focused pattern is the horse move extension (most probably a corner



$r = 0.537$, a horse move enclosure.



$r = 0.473$, a two space 3rd line extension.



$r = -0.317$, preventing White from connecting.

Figure 5.7: Territoriality. $r > 0$ for patterns correlated with increasing territoriality, $r < 0$ for patters which correlate with the moyo style of game.

enclosure) of $r = 0.537$ and a slightly weaker two space third line extension of $r = 0.473$. The strongest pattern with negative $r = -0.317$ is a move which prevents White from connecting underneath — allowing the enemy to connect is often bad, because connected group are stronger and the influence-preferring player wants to use his influence to fight. All these pattern attributes are shown in Figure 5.7.

Another interesting result is the fact, that the territoriality is also correlated with the difference between average number of sente and gote moves ($r = 0.347$) and the average number of sente moves ($r = 0.336$). Since the $\omega = 5$, which considers rather tightly local responses, we believe that this corresponds to moves which close the side in sente (e.g. from sliding under the stones, as also backed by a pattern attribute in Figure 5.8).

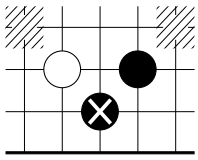


Figure 5.8: $r = 0.292$, securing the side.

Orthodoxity

The orthodoxity scale spans from 1 (players with a classic style, Pearson's $r < 0$) and 10, which corresponds to a novel style of play ($r > 0$).

The strongest attributes that correlate with classic style of game are related to that of territorial style from above. For example, the strongest "classic" attribute is the number of stones on third line played in the opening with $r = -0.479$; the second strongest is the horse move enclosure $r = -0.456$. The orthodox player also likes the formation shown in the middle of Figure 5.9, $r = -0.374$.

On the other hand, the novel player tends to play openings stressing influence by playing on the fourth line $r = 0.429$ or higher $r = 0.317$ (opening moves that are played higher than on the fourth line are quite uncommon, not studied as thoroughly and thus probably giving possibilities for innovative moves). Apart from the opening moves, attributes for novel players ($r > 0$) are generally weaker than those for

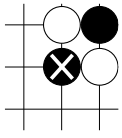
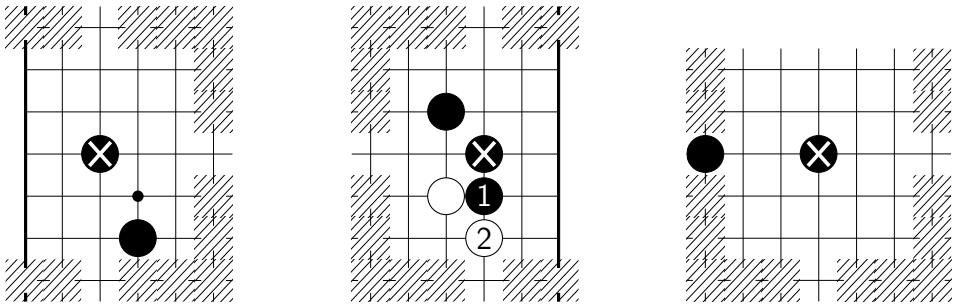


Figure 5.10: $r = 0.249$, a crosscut.



$r = -0.456$, a horse move enclosure.

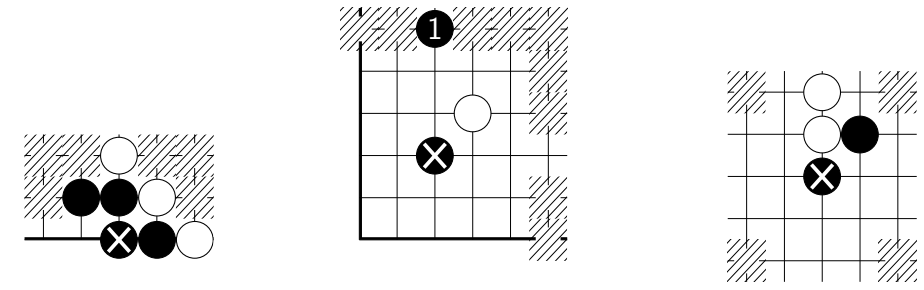
$r = -0.374$, pulling back after an attachment of Black **1**.

$r = -0.367$, a two space extension.

Figure 5.9: Orthodoxity. $r > 0$ for novel players, $r < 0$ denotes a classical style of game.

the classic player. This is not surprising, because novel players tend to come up with new and unique moves; thus trying to find a typical novel move might prove to be elusive. For example, the strongest pattern attribute with positive r, the crosscut shown in Figure 5.10, is not really a novel move. However, it often results in a complicated position, which could give opportunities for surprising innovative moves.

Aggressivity



$r = -0.306$, an endgame move.

$r = -0.265$, a corner invasion.

$r = 0.262$, a hane at the head of enemy stones.

Figure 5.11: Aggressivity. $r > 0$ for players who like to fight, $r < 0$ marks a calm player.

The aggressivity scale spans from the calm playing style (number 1, attributes correlating with decreasing aggressivity have $r < 0$), to very aggressive game style (number 10 on the scale, $r > 0$), where the player loves to fight.

Again, (with a few interesting exceptions) the calm player tends to play on the third line in the opening ($r = -0.428$), likes secure territory in the corner (second strongest attribute $r = -0.426$ is the same horse move enclosure from previous styles), and safe two space extension $r = -0.403$ (we omit these two patterns, since the reader is already familiar with them from above).

Moreover, he likes to play a 3-3 corner invasion $r = -0.265$ Figure 5.11 (middle). Most probably, this is a part of the sequence showed in Figure 5.12. ❶ approaches the white 4-4 stone, ② is a very aggressive response, a so-called pincer (Sensei's Library, 2013b). ❸ is a calm response, giving up the ❶ for now and taking the corner instead.

Finally, the calm player tends to win by points $r = -0.349$ more than the aggressive player. Interestingly, the third strongest pattern correlated with calmness was the endgame move ($r = -0.306$) on the left of Figure 5.11. This might mean that a strong calm player is aiming to win the game during the endgame, which usually requires mechanically precise reading and counting.

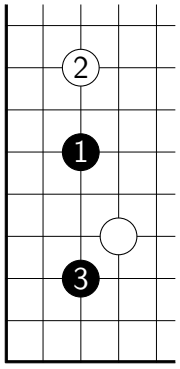
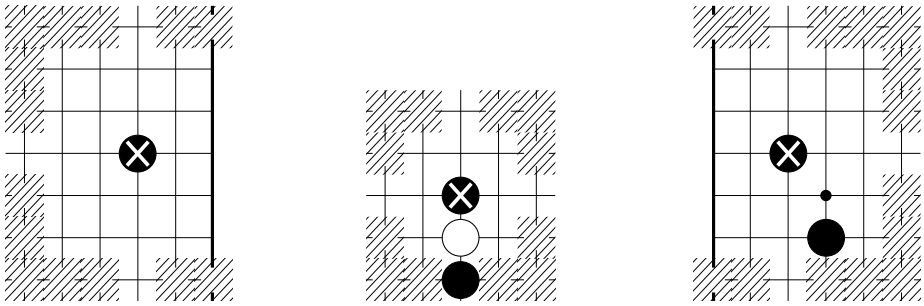


Figure 5.12: A standard joseki.

The fighting player likes to play on the fourth line in the opening ($r = 0.418$). The second strongest attribute ($r = 0.334$) of the fighting player is the number of moves played above the fourth line in the *early middle game* (moves 17 to 64) — these moves might be reductions of the opponents territory or preparations of the battlefield for future running fights. In line with our expectation, the fighting player also tends to win by resigns ($r = 0.234$).

Thickness



$r = 0.342$, a spacious move on a third line.

$r = 0.288$, a clamp, often a key move of a *tesuji*.

$r = 0.258$, yet another horse move enclosure.

Figure 5.13: Thickness. $r > 0$ for players who are skilled at shinogi, $r < 0$ for player whose formations are safe.

Finally, there is the thickness scale. It spans from safe style of game (1, $r < 0$) to shinogi style (10, $r > 0$). As discussed in Chapter 1.3.3, thick positions have little or no weaknesses. *Shinogi* is a term used when a player skillfully overcomes a crisis. The scale definition is not that clear-cut in comparison with the preceding scales, and also the variance of answers of the questionnaire is lower than the other scales.

The dependencies for the thickness scale are generally weaker than we have seen with the previous styles. Regarding the shinogi end of the scale, the strongest

attribute is the number of opening moves played on the third line ($r = 0.352$), followed by a third line pattern ($r = 0.342$) which could be a wide extension, or a move intended to disrupt enemy side formation in the opening (Figure 5.13, left). Interestingly, the third strongest attribute with positive correlation ($r = 0.288$) is a pattern attribute showing a contact move called clamp, which is often a part of *tesuji* (a cunning sequence which achieves something — e.g. saving a group, capturing enemy's key stones, etc.). The next strong pattern correlated with mastering the shinogi is the number of moves played on the first or second line within the early middle game $r = 0.272$. This could very well correspond to moves that are securing life in an enemy's sphere of influence, or placing stones that will be sacrificed to yield an advantage — playing this low this early neither builds a nice territory (third line is supreme in this), nor builds influence directly.

The thick part of this scale does not yield dependencies with clear interpretation. The two strongest attributes with $r < 0$ are the number of moves played on four line in the opening ($r = -0.312$), or above ($r = -0.261$). In line with this, the third is the number of stones played above the fourth line in the early middle game $r = -0.224$. These moves are not really moves characteristic for what we consider thick.

We believe that the main reason for this is the Takemiya Masaki, who is a strong outlier on this scale. Master Takemiya is known for building extreme moyos and prefer influence very strongly over safe territory. His games have often depended on whether the opponent lives inside the sketched moyo or not — so the score of 1.8 he received on the scale of thickness maybe does not reflect the fact that he is very thick and plays safely (which he does not), but the fact that his groups do not often need to shinogi, because it is the opponents who do. This probably implies, that the scale of thickness was not chosen very well. We treat this matter further in the discussion (Chapter 6).

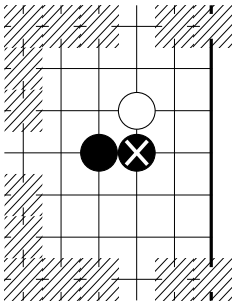


Figure 5.14: $r = -0.216$, A very thick move, called the iron pillar.

Only the fourth strongest negative attribute ($r = -0.216$) has clear interpretation as a thick move, it is the *iron pillar* block shown in Figure 5.14.

6. Discussion and Future Work

In this work, we dealt with many issues regarding the possibilities of inference of different variables (strength, styles) from collections of records of the game of Go. We found out that the inference of these variables is indeed possible, under few assumptions.

Most importantly, we need a set of robust features, that are general enough to capture information present in games, which is relevant to the target variable. Based on our knowledge of the game, we invented a number of features (Chapter 3) and tested their performance under different settings (Sections 5.1.2 and 5.2.2). In line with our expectations, different target variables are best captured by different features and attributes. A clear example of this was seen in the case of style: the histogram feature of distance from the board edge (in the opening) proved to capture the territoriality very well (Section 5.2.2). Generally, the pattern feature turned out to be the strongest for both the strength and the styles regression problems — this is because the pattern feature basically consists of statistics of many independent[1] patterns and the machine learning methods can "pick" the ones that are relevant. Analysis of the individual attributes (Sections 5.1.4 and 5.2.4) reveals, that all the features have useful attributes for some of the target variables — for example, the number of sente sequences correlates with strength, as did the number of captured stones in the early game. To mention one more example, the percentage of games won by resignation correlates with the aggressivity of a player.

Concerning the correlations, we need to say that except for the strongest correlation of a single attribute we have found, which had Pearson's r of 0.621[2], the dependencies were not very strong. The few best attributes for each variable have what would be better called a moderate dependency, on average spanning from $r = 0.25$ to $r = 0.5$. We conclude, that the feature space (defined by different features we have used) comprises many relatively weakly correlated attributes and only a few attributes that are correlated moderately. Overall, we should also note that we consider the fact that almost all of the most strongly correlated attributes were in line with common Go knowledge to be a sign of good "expressive" power of the features.

Secondly, to obtain the best performance, robust machine learning methods (Chapter 4) are needed. These need to be powerful enough to make use of the weak dependencies in the data. We have tested different algorithms — out of these, the method of stacked generalization turned out to have supreme performance. At first, we tried to hand-tune the ensemble (Appendix C.3), to find out that this is too cumbersome and time-consuming if best performance is sought. To solve the problem of finding the best combination of learners into the stacked ensemble, we designed a genetic algorithm, which showed an improvement[3] of 3% (for the strength data, Section 5.1.3) and of 6% (average improvement for the style data, Section 5.2.3) over the initial hand tuned ensemble, which we consider

[1] Strictly speaking, the attributes are truly independent only when *independent normalization* (Section 3.2) is used.

[2] The correlation marks dependency between the *territoriality* scale, and the number of moves played on the third line during the opening phase, Section 5.2.4.

[3] The baseline being the mean regression learner.

a good result. This might not seem like a lot, but remember that we compare the best GA learner with the best hand-tuned learner we found[4]. The ensembles evolved by the genetic algorithm are fairly complex (e.g. Figure 5.2) and would not be probably found by hand at all. It should be noticed, that the method of the genetic algorithm does however take considerable amounts of time. In cases where this would not be feasible, some worse (in terms of performance) yet faster method could be used. For instance, bagged neural network on itself performed pretty well (Table 5.3).

Finally, the scales need to be defined clearly and we need reliably tagged data. For example, the ranking system in the game of Go is very clearly defined by means of handicap stones (Section 1.2). Similarly, the scale of e.g. *territoriality* seems to be defined very well — judging by the results of the style regression (Section 5.2.3) and the fact that the strongest attributes seem to be in accordance with the traditional knowledge. On the other hand, the analysis of the strongest attributes correlated with the thick-playing end of *thickness* scale (Section 5.2.1) did not give patterns that really correlate with the traditional concept. The fact that the thickness scale is somewhat different from the other styles can also be guessed from the standard deviation $\sigma = 1.6$ of the answers on the thickness scale part of the questionnaire, which is much lower than that of the other styles (other styles have σ ranging from 2.176 to 2.415, see Table 5.4). Were the answers on the scale 1 to 10 distributed uniformly, the σ would be approximately 2.6.[5] This basically means that the scale of thickness is not "used well". This is also backed up by the comment of one of the interviewees, Vladimír Daněk, who noted that the concept of thickness is sometimes not in opposition to shinogi, but rather in coordination with it — when one plays thickly on one side of goban, he often plays shinogi on the rest. The interviewees might thus simply have understood this scale as how skilled the particular proffessional is in playing shinogi.

Regarding the styles, one more point should be made. We have seen, that the strongest attributes were often very similar. For example, the knight-move corner enclosure appears for the territory stressing player, the player with classic style and the player who plays safely. This is in line with the correlations of the scales themselves (Table 5.4). It might be however interesting to look for concepts that are *orthogonal*.

Future work and applications

Indeed, the methodology presented in this work could be extended. Regarding the prediction abilities, we believe that apart from increasing the dataset sizes[6] generally the only feasible way of further improvement is to add new features that capture different aspects of the gameplay (or refine the ones presented here). We made this conclusion after trying many different learners and other machine learning techniques. To name a few, we elaborated with the Support Vector Regression (Smola – Schölkopf, 2004), yet we were unable to get performances comparable

[4]Of course, the hand-tuned learner was tuned to the domain of strength, so the expected improvement is naturally bigger for the case of styles.

[5]From definition of standard deviation of random variable X from $U(a, b)$, $\sigma^2 = \mathbb{E}X^2 - (\mathbb{E}X)^2 = \ldots = (b-a)^2/12$.

[6]Which has clear computational limits.

to any other methods and the SVM took considerable time to learn.[7] We also tried to preprocess the data by the Principal Component Analysis (Jolliffe, 1986), a commonly used method to reduce dimension[8], but it did not yield any improvement whatsoever. Observing the performance of learners with the current set of features suggests that there is a clear upper bound on precision that can be achieved (see Table 5.3). We therefore believe that improving the features is the way to improve the performance further.

For example, we have devised a histogram feature which counts numbers of captured stones (Section 3.4.2). A straightforward idea would be to extend the counts to include the dead stones[9] as well. Deciding which stones are dead and which are alive is however in itself not an easy problem. Luckily, the status of a stone (or a group of them) can be estimated using methods of the Monte-Carlo tree search.[10] By looking at the owner of a stone at the end of each random simulation, the probable owner can be estimated. Moreover, this is a standard part of the MCTS bots, such as the Pachi (Baudiš et al., 2012) we already use to extract the patterns. We plan to extend the Pachi to output this information in the future, the new feature could simply extend the captured stones histogram from this work.

Adding new features and improving the prediction power is not important just for the sake of it, but also because of possible applications. For instance, we see some interesting potential in the attribute evaluations of the pattern feature.

Firstly, there is the educational potential — the attribute analysis of strength gives us a list of patterns (or other attributes), that are mainly played by weak players (Section 5.1.4). By simply pointing out the fact that a certain move is bad and why is it so, we can give any particular weak player a direct advice regarding his play. To test the idea, we have implemented this in the web application (Appendix A) for the top few bad patterns. Of course, this can be "reliably" done only for the most strongly correlated patterns, since the weaker dependencies are burdened by larger error.

Secondly, regarding the pattern attributes (and strength), the attribute evaluation could help to improve computer Go programs. For a given set of patterns[11], the method essentially gives weights of each pattern. This weighting might improve the random Monte-Carlo simulations, similarly as in (Coulom, 2007). Moreover, this could even be used to balance the level of the bot — making the bot do human-like bad moves could give more natural feel of the game for a weak human player. As far as we are aware, this is a novel idea.

The strength estimation could also be used to help to determine initial ranking of a player, both on the internet (where it often takes some time before the

[7] Apart from simple manual tuning, we also used (to no avail) the automatic parameter searching techniques present in the Orange datamining framework (see Appendix B, Implementation).

[8] We tried to reduce the dimension by taking the coefficients of projection to base of first N components, for different values of N.

[9] Remember from Chapter 1.1 that *dead* stones do not have two eyes and cannot be connected with stones that are *alive*.

[10] We mentioned the Monte-Carlo tree search in Chapter 1 on page 4, for instance, see Browne et al. (2012).

[11] We use the top N patterns from the data set (Chapter 3.2), but we could for example use all the spatial configurations up to a certain (reasonably small, e.g. 3 or 4) gridcular distance.

ranking algorithm converges to the real value) and in the real matches. In the near future, we plan to study how small could the number of games really be to still give a reliable estimate of the strength. From some initial experiments we have performed with the strength dataset from Section 5.1.1, it seems that the precision does not depend on the number of games, as far as the sample is larger than 10 games.

Similarly, precise prediction of style can serve as a tool for Go players — we can recommend professional games to review or point out some things to focus on to balance player's skillset. We also realized this as a part of the web application (Appendix A). Based on the user's predicted style, we compute the Euclidean distance to the professionals from the style questionnaire (Section 5.2.1) and present the user those who are relevant[12]. We are aware of only two tools, that do something alike, both of them are however based on a predefined questionnaire. The first one is the tool of Mr. Dinerchtein (2012) — the user answers 15 questions and based on the answers he gets one of predefined recommendations. The second tool is not available at the time of writing, but the discussion at (Sensei's Library, 2013l) suggests, that it computed distances to some pros based on user's answers to 20 questions regarding the style. We believe, that our approach should be more precise, because the evaluation takes into account many aspects of the games. On the other hand, since the style estimation in our work is trained on professional players' data, inadequate skill of the user is surely a source of errors[13]; it is however a question for a further discussion whether the concept of style as we defined it is even relevant for a beginner. Since the web application allows us to receive feedback on the style predicted, we plan to investigate this further in the future.

[12]We show the 4 professional players that are closest to the user and 4 that are farthest apart, based on the euclidean distance, see the web application for details.

[13]Which we unfortunately cannot even enumerate, since we have no style data for weaker players yet.

Conclusion

In this work, we extended the methodology for extracting evaluations of players from a sample of Go game records originally presented in (Baudiš – Moudřík, 2012). Firstly, we added more features and laid out a methodology for their comparison. Secondly, we developed a robust machine-learning framework, which is able to capture the dependencies between the evaluations and general target variable using ensemble meta-learning with a genetic algorithm.

We applied this framework to two domains, estimation of strength and styles. The results show that the inference of the target variables in both cases is viable and reasonably precise, except for the style scale of thickness which was not, however, defined well. Finally, we have presented a web application, which realizes the methodology, while presenting a prototype teaching aid for the Go players and gathering more data.

Overall, we hope that the findings of this work will be useful in deepening both human and computer understanding of the game of Go.

Bibliography

BAUDIŠ, P. – MOUDŘÍK, J. On Move Pattern Trends in a Large Go Games Corpus. *Arxiv, CoRR*. October 2012. Available at: `http://arxiv.org/abs/1209.5251`.

BAUDIŠ, P. et al. *Pachi — Simple Go/Baduk/Weiqi Bot* [online]. 2012. Available at: `http://repo.or.cz/w/pachi.git`.

BREIMAN, L. Random Forests. *Machine Learning*. October 2001, 45, 1, pages 5–32. ISSN 0885-6125. doi: 10.1023/A:1010933404324.

BREIMAN, L. Bagging predictors. *Mach. Learn.* August 1996, 24, 2, pages 123–140. ISSN 0885-6125. doi: 10.1023/A:1018054314350. Available at: `http://dx.doi.org/10.1023/A:1018054314350`.

BREIMAN, L. et al. *Classification and regression trees*. Monterey, CA : Wadsworth & Brooks/Cole Advanced Books & Software, 1984. ISBN 978-0-412-04841-8.

BROWNE, C. et al. A Survey of Monte Carlo Tree Search Methods. *Computational Intelligence and AI in Games, IEEE Transactions on*. march 2012, 4, 1, pages 1 –43. ISSN 1943-068X. doi: 10.1109/TCIAIG.2012.2186810. Available at: `http://www.doc.ic.ac.uk/~sgc/papers/browne_ieee12.pdf`.

Celery Project. *Celery Project* [online]. 2013. Available at: `http://www.celeryproject.org`.

COULOM, R. Computing Elo Ratings of Move Patterns in the Game of Go. In HERIK, H. J. et al. (Ed.) *Computer Games Workshop*, Amsterdam Pays-Bas, 2007. Available at: `http://hal.inria.fr/inria-00149859/en/`.

COVER, T. M. – HART, P. E. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*. 1967, 13, 1, pages 21–27. doi: 10.1109/TIT.1967.1053964.

CURK, T. et al. Microarray data mining with visual programming. *Bioinformatics*. February 2005, 21, pages 396–398. ISSN 1367-4803. Available at: `http://bioinformatics.oxfordjournals.org/content/21/3/396.full.pdf`.

GROOT, F. *Moyo Go Studio* [online]. 2005. Available at: `http://www.moyogo.com/`.

DINERCHTEIN, A. *What is your playing style?* [online]. 2012. [cit. 8. 4. 2013]. Available at: `http://style.baduk.com`.

FAIRBAIRN, J. *Go in Ancient China* [online]. 1995. [cit. 15. 1. 2013]. Available at: `http://www.pandanet.co.jp/English/essay/goancientchina.html`.

FAIRBAIRN, J. *Games of Go on Disk — GoGoD Encyclopaedia and Database, Go players' styles* [online]. winter 2011. Available at: `http://www.gogod.co.uk/`.

Go Census. [online]. 2002. [cit. 10. 2. 2013]. Available at: http://web.archive.org/web/20021217041220/http://www.fin.ne.jp/~igo/census.htm.

Google and community. *AngularJS* [online]. 2013. Available at: http://www.angularjs.org.

GÖRTZ, U. *Kombilo — a Go database program (version 0.7)* [online]. 2012. Available at: http://www.u-go.net/kombilo/.

HALL, T. M. – FAIRBAIRN, J. *Games of Go on Disk — GoGoD Encyclopaedia and Database* [online]. winter 2011a. Available at: http://www.gogod.co.uk/.

HALL, T. M. – FAIRBAIRN, J. *Games of Go on Disk — GoGoD Encyclopaedia and Database, Articles on Famous Players – Honinbo Dosaku* [online]. winter 2011b. Available at: http://www.gogod.co.uk/.

HAYKIN, S. *Neural Networks: A Comprehensive Foundation (2nd Edition)*. : Prentice Hall, 2 edition, jul 1998. Available at: http://www.worldcat.org/isbn/0132733501. ISBN 0132733501.

HOLLOSI, A. *SGF File Format* [online]. 2006. Available at: http://www.red-bean.com/sgf/.

JOLLIFFE, I. *Principal Component Analysis*. New York : Springer, 1986.

KOHAVI, R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. pages 1137–1143. Morgan Kaufmann, 1995.

MOUDŘÍK, J. *Orange Hacks* [online]. 2013. Available at: http://www.j2m.cz/~jm/orange_hacks/.

MOUDŘÍK, J. – BAUDIŠ, P. *GoStyle — Determine playing style in the game of Go* [online]. 2013. Available at: http://gostyle.j2m.cz/.

NISSEN, S. Implementation of a Fast Artificial Neural Network Library (fann). Technical report, Department of Computer Science University of Copenhagen (DIKU), 2003. http://fann.sf.net.

Python Software Foundation. *Python 2.7* [online]. November 2008. Available at: http://www.python.org/dev/peps/pep-0373/.

RIEDMILLER, M. – BRAUN, H. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591, 1993.

RODGERS, J. L. – NICEWANDER, W. A. Thirteen ways to look at the correlation coefficient. *The American Statistician*. Feb 1988, 42, 1, pages 59–66.

ROSIPAL, R. – KRäMER, N. Overview and recent advances in partial least squares. In *in 'Subspace, Latent Structure and Feature Selection Techniques', Lecture Notes in Computer Science*, pages 34–51. Springer, 2006. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.85.7735.

Sensei's Library. *Sensei's Library* [online]. 2013a. [cit. 15. 1. 2013]. Available at: http://senseis.xmp.net/.

Sensei's Library. *4-4 point low approach one-space low pincer* [online]. 2013b. [cit. 29. 3. 2013]. Available at: http://senseis.xmp.net/?44PointLowApproachOneSpaceLowPincer.

Sensei's Library. *Dan* [online]. 2013c. [cit. 8. 2. 2013]. Available at: http://senseis.xmp.net/?Dan.

Sensei's Library. *The empty triangle is bad* [online]. 2013d. [cit. 18. 3. 2013]. Available at: http://senseis.xmp.net/?TheEmptyTriangleIsBad.

Sensei's Library. *Go Proverbs* [online]. 2013e. [cit. 18. 2. 2013]. Available at: http://senseis.xmp.net/?GoProverbs.

Sensei's Library. *Rank - worldwide comparison* [online]. 2013f. [cit. 16. 1. 2013]. Available at: http://senseis.xmp.net/?RankWorldwideComparison.

Sensei's Library. *Rules of Go* [online]. 2013g. [cit. 16. 1. 2013]. Available at: http://senseis.xmp.net/?RulesOfGo.

Sensei's Library. *Scoring* [online]. 2013h. [cit. 16. 1. 2013]. Available at: http://senseis.xmp.net/?Scoring.

Sensei's Library. *Seki* [online]. 2013i. [cit. 16. 1. 2013]. Available at: http://senseis.xmp.net/?Seki.

Sensei's Library. *Professional Players' Go Styles* [online]. 2013j. [cit. 12. 3. 2013]. Available at: http://senseis.xmp.net/?ProfessionalPlayersGoStyles.

Sensei's Library. *Rules of Go - introductory* [online]. 2013k. [cit. 15. 1. 2013]. Available at: http://senseis.xmp.net/?BasicRulesOfGo.

Sensei's Library. *Which Pro Do You Most Play Like* [online]. 2013l. [cit. 8. 4. 2013]. Available at: http://senseis.xmp.net/?WhichProDoYouMostPlayLike.

SHUBERT, W. *KGS — Kiseido Go Server* [online]. 2013a. Available at: http://www.gokgs.com/.

SHUBERT, W. *KGS Archives — Kiseido Go Server* [online]. 2013b. Available at: http://www.gokgs.com/archives.jsp.

SMOLA, A. J. – SCHÖLKOPF, B. A tutorial on support vector regression. *Statistics and Computing*. aug 2004, 14, 3, pages 199–222. ISSN 0960-3174. doi: 10.1023/B:STCO.0000035301.49549.88. Available at: http://www.cmlab.csie.ntu.edu.tw/~cyy/learning/papers/SVR_Tutorial.pdf.

STERN, D. – HERBRICH, R. – GRAEPEL, T. Bayesian pattern ranking for move prediction in the game of Go. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 873–880, New York, NY, USA, 2006. ACM. doi: http://doi.acm.org/10.1145/1143844.1143954. ISBN 1-59593-383-2.

STEEN, J. *Gobase* [online]. 2013. Available at: http://www.gobase.org.

EES, T. The Slow Way West. In *Third International Conference on Baduk*, Myong-Ji University, Yongin City, Korea, October 2005. Available at: http://www.gogod.co.uk/. Available as a part of the GoGoD Encyclopaedia and Database, History.

WEDD, N. *Computer Go Info — Human-Computer Go Challenges* [online]. 2013. [cit. 8. 2. 2013]. Available at: http://www.computer-go.info/h-c/index.html.

WHITLEY, D. A genetic algorithm tutorial. *Statistics and computing*. 1994, 4, 2, pages 65–85.

WIKIPEDIA. *Computer Go — Wikipedia, The Free Encyclopedia* [online]. 2013. [cit. 17. 1. 2013]. Available at: http://en.wikipedia.org/w/index.php?title=Computer_Go&oldid=530803356.

WIKIPEDIA. *Go ranks and ratings — Wikipedia, The Free Encyclopedia* [online]. 2012. [cit. 18. 1. 2013]. Available at: http://en.wikipedia.org/w/index.php?title=Go_ranks_and_ratings&oldid=522963011.

WOLPERT, D. H. Stacked Generalization. *Neural Networks*. 1992, 5, pages 241–259. Available at: http://www.machine-learning.martinsewell.com/ensembles/stacking/Wolpert1992.pdf.

YEVSTYGNYEYEV, V. *Gokifu* [online]. 2013. Available at: http://www.gokifu.com.

List of Abbreviations

GA	Genetic Algorithm
GoGod	Games of Go on Disk
KGS	Kiseido Go Server
k-nn	k-nearest neighbors
MCTS	Monte Carlo Tree Search
MSE	Mean Square Error
NN	Neural Network
PLS	Partial Least Squares Regression
RMSE	Root Mean Square Error
SVM	Support Vector Machine

List of Algorithms

List of Figures

List of Tables

A. Web Application

The web application is published as a part of the GoStyle project, which we founded to study the possibilities of the computer analysis of Go game records by methods presented in this work. (Moudřík – Baudiš, 2013) The web of the project has two main parts:

1. An interactive questionnaire.[1]
2. The web application itself.[2]

Interactive questionnaire

The sole purpose of the interactive questionnaire[1] is to get preciser data about the styles of professionals. It serves to substitute our old method of gathering information from strong players by e-mail and copying and formating the information by hand. The information obtained is the same as in the questionnaire from Section 5.2.1. For a number of strong professional players (and also a few strong amateurs that are active on the Kiseido Go Server (Shubert, 2013a)), we ask the interviewee to evaluate style of these players. See Section 5.2.1 for details about the styles.

Web application

The web application[2] allows anyone to upload a sample of games (or specify a Kiseido Go Server nickname). Based on the sample, we do several things:

- We estimate the strength of the player (as in Section 5.1),
- we estimate the style of the player (Section 5.2),
- based on the estimated style, we reccomend a list of 4 professionals (from Section 5.2.1) whose styles are closest (by computing Euclidean distance between the styles) and 4 professionals whose styles are farthest.
- Also, we compare the feature vector computed by the strength regression with a linear model fitted for the top "weak players' attributes" and in case that the value of the attribute in the sample is corresponding to player who is weaker than 8-kyu, we warn the user.[3] This serves as a very simple teaching aid. Currently, this approach is limited by the fact, that the dependencies of single attributes are very weak and have large error, as discussed in Chapter 6.

[1] `http://gostyle.j2m.cz/questionare.html`

[2] `http://gostyle.j2m.cz/webapp.html`

[3] We warn the user about the empty triangles, stones captured in the opening, pushing from behind, number of sente and gote sequences.

- Finally, we let the user correct the strength and style (if he thinks the web application is mistaken). This feedback will allow us to improve the methods in the future.

The source code for both the server and client part is available online, as detailed in the implementation (Appendix B). Some screenshots of the web application follow.

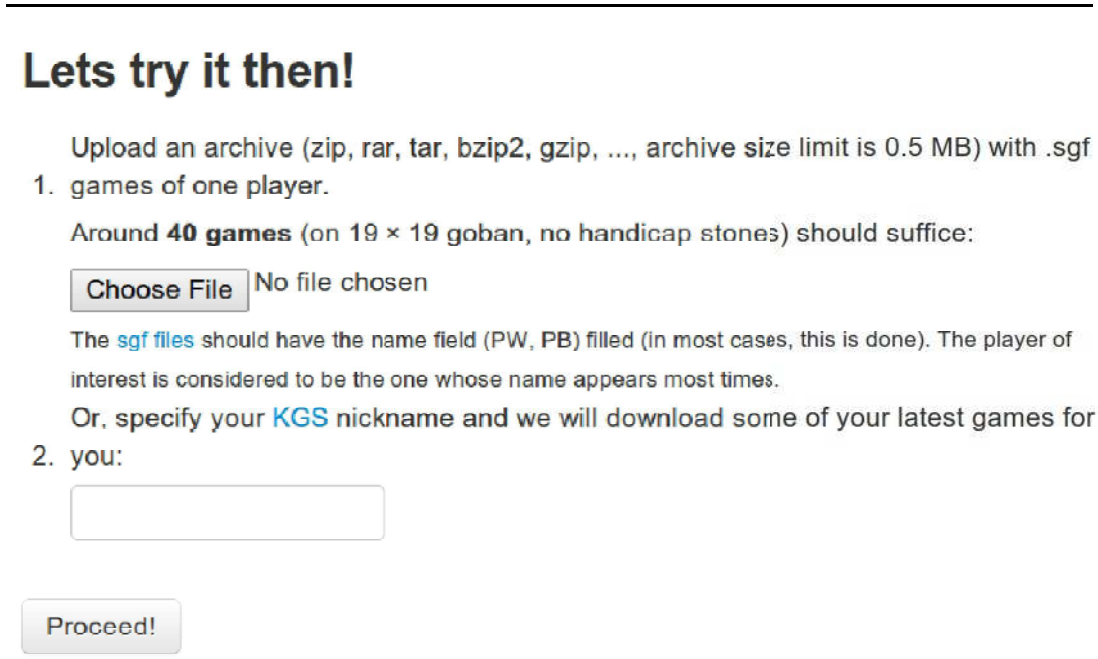


Figure A.1: A portion of the Web Application, showing how can the user upload data.

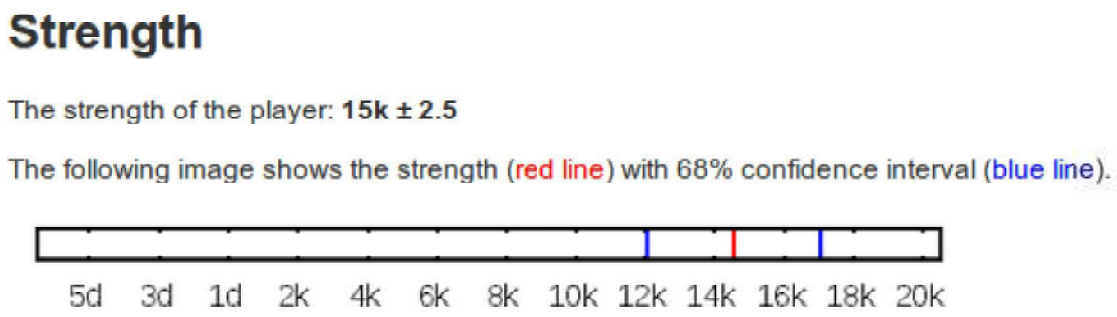


Figure A.2: A portion of the Web Application, showing results of the strength estimation for a weak player (whose real strength in this case is 13-kyu).

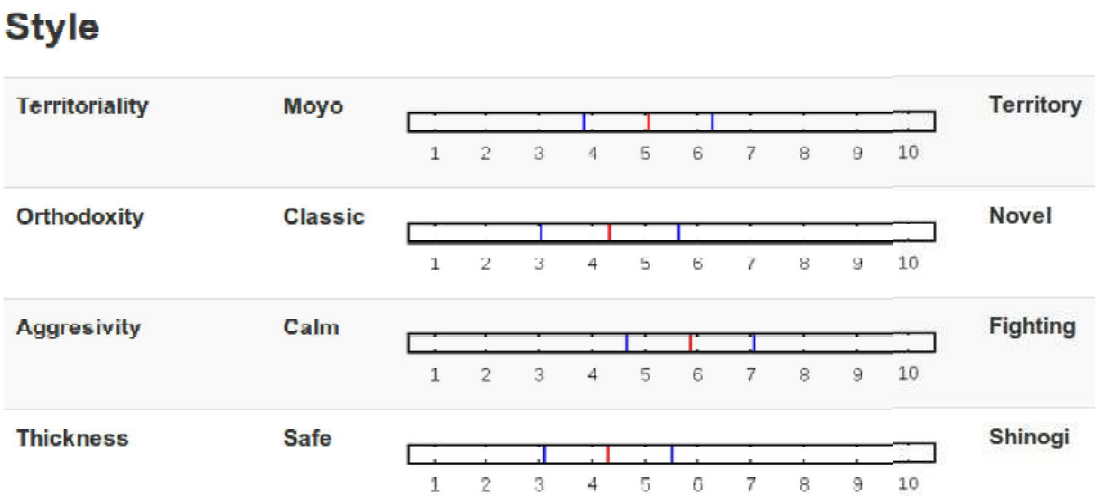


Figure A.3: A portion of the Web Application, showing results of style estimation and similar professionals for a weak player.

Distinctive patterns and tips

Bad things you play

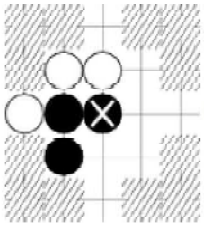
- You should concrentrate on the concept of sente and gote. Do you ask after every enemy move, Do I need to respond to this? and If so, *Do I have a possibility to respond on some unexpected place so that the threat is neutralized, but the opponent in turn has to respond?*
- We found out, that you really like to capture stones. This is not necessarily a bad thing, but are you always sure, that the stones you captured are important and the mere 6 points you get for capturing them is the best move on board? (E.g. in the opening, ...)
-  Huh, we think that you like the empty triangle. Please, remember, that this shape is usually terrible and try not to play it.. Are the stones you are trying to save necessary?

Figure A.4: A portion of the Web Application, showing some recommendations for a weak player.

B. Implementation

The code used in this thesis[1] is available on the attached CD, or released online as a part of GoStyle project (Moudřík – Baudiš, 2013). The majority of the source code is implemented in the Python programming language.[2]

The majority of the machine learning methods used were taken from the Orange Datamining suite (Curk et al., 2005), with the exception of the Fast Artificial Neural Network library FANN (Nissen, 2003) and our wrapper for this library (Moudřík, 2013).

We used the Pachi Go engine (Baudiš et al., 2012) for the raw pattern feature extraction.

Web Application

The server part of the web application is written in the Python programming language (Python Software Foundation, 2008), with aid of the Celery framework for asynchronous task processing (Celery Project, 2013).

The client part is a standard combination of HTML and Javascript and it uses the AngularJS framework (Google and community, 2013).

[1] `http://repo.or.cz/w/gostyle.git`

[2] (Python Software Foundation, 2008)

C. Parameters

C.1 Feature Extractors

Feature extractor	Settings
Pattern feature	Normalization $\in$ {independent, proportional, linear}, $N \in \{200, 400, 600, 800, 1000\}$ all combinations, A randomly sampled as 20% of all the games in the domain.
Local sequences	$\omega \in \{5, \ldots, 15\}$
Border distance	$ByDist = \{\langle 1, 2\rangle, \langle 3\rangle, \langle 4\rangle, \langle 5, \infty)\}$, $ByMoves = \{\langle 1, A\rangle, (A, B\rangle, (B, C\rangle, (C, \infty)\}$, $A \in \{10, 16\}, B \in \{44, 54, 64\}, C \in \{160, 200, 240\}$, all combinations.[1,2]
Captured stones	$ByMoves = \{\langle 1, A\rangle, (A, B\rangle, (B, \infty)\}$, $A \in \{40, 60, 80\}, B \in \{160, 200, 240\}$, all combinations.[1]
Win/Loss statistics	—
Win/Loss points	—

[1] The bounds for the parameters were partially limited by hand tuning prior to experiments.

[2] The motivation behind the relatively large number of boxes is to capture both the very early opening and late opening/very early middle game, which we expected to have a big importance. The last two intervals should correspond to late middle game and endgame.

C.2 Base Learners and their Settings

Base learner	Settings
Mean regression	—
PLS regression	$l \in \{2, \ldots, 10\}$
k-nearest neighbors	$k \in \{10, 20, \ldots, 60\}$, $\alpha \in \{10, 20\}$, $\delta \in \{\text{Manhattan}, \text{Euclidean}\}$, all combinations.
Random Forests	$N \in \{5, 10, 25, 50, 100, 200\}$
Neural network	Desired $\epsilon \in \{0.001, 0.005\}$, $max \in \{50, 100, 200, 500\}$ iterations, 1 hidden layer with number of neurons $\in \{10, 20\}$, all combinations. We used the symmetric sigmoid activation function.[1] [2]
Bagged Neural networks	For ensemble sizes of $\in \{20, 40, 60\}$, each Neural network (from right above) was tested.

[1] We have used a neural network with one hidden layer. The number of neurons in the input and output layers depends on the dimensions of data. Moreover, the range of activation function is $(-1, 1)$, while the range of domains of dependent variables in the work is larger (e.g. $\langle -5, 20 \rangle$ for strength data). Therefore, we had to scale the data. Given a training set $Tr = \{(x_i, y_i), ...\}$, we mapped the $min(y_i)$ to -1 and $max(y_i)$ to 1 and the values in between linearly. (Of course, the process is reversed when we predict the value, to give y's from the original range). The training data thus should not have smaller domain than the testing data, or the error is increased. With proper training/testing data sampling, we did not find this to be a problem.

[2] The bounds for the parameters were partially limited by hand tuning prior to experiments, because training the neural network is computationally costly.

C.3 Initial Hand-tuned Learner

This learner was found by hand-tuning for the strength data and we use it as a reference learner throughout the work.

Ensemble learner	Settings
Stacking	4 folds, level 2 learner: Neural network with desired $\epsilon = 0.005$, $max = 100$ iterations, 1 hidden layer with 10 neurons.
Base learners	**Settings**
Mean regression	—
PLS regression	$l = 3$
k-nearest neighbors	$k = 50$, $\alpha = 20$, $\delta =$ Manhattan.
Random Forests	$N = 50$
Bagged Neural network	$20 \times$ Bagged Neural network: desired $\epsilon = 0.001$, $max = 100$ iterations, 1 hidden layer with 10 neurons.

C.4 Strength: Best GA Stacking Ensemble

Ensemble learner	**Settings**
Stacking	6 folds, level 2 learner: Bagged (20×) Neural network with desired $\epsilon = 0.005$, $max = 500$ iterations, 1 hidden layer with 10 neurons.
Base learners	**Settings**
Mean regression	—
PLS regression	$l = 3$
Random Forests	$N = 50$
Neural network	Desired $\epsilon = 0.001$, $max = 200$ iterations, 1 hidden layer with 20 neurons.
k-nearest neighbors	$k = 20$, $\alpha = 20$, $\delta =$ Euclidean.
k-nearest neighbors	$k = 40$, $\alpha = 10$, $\delta =$ Manhattan.
k-nearest neighbors	$k = 40$, $\alpha = 10$, $\delta =$ Euclidean.
k-nearest neighbors	$k = 40$, $\alpha = 20$, $\delta =$ Euclidean.
k-nearest neighbors	$k = 50$, $\alpha = 10$, $\delta =$ Manhattan.
k-nearest neighbors	$k = 50$, $\alpha = 20$, $\delta =$ Manhattan.
k-nearest neighbors	$k = 50$, $\alpha = 20$, $\delta =$ Euclidean.
k-nearest neighbors	$k = 60$, $\alpha = 10$, $\delta =$ Euclidean.
k-nearest neighbors	$k = 60$, $\alpha = 20$, $\delta =$ Euclidean.
Bagged Neural network	20 × Bagged Neural network: desired $\epsilon = 0.001$, $max = 100$ iterations, 1 hidden layer with 10 neurons.
Bagged Neural network	40 × Bagged Neural network: desired $\epsilon = 0.005$, $max = 100$ iterations, 1 hidden layer with 10 neurons.
Bagged Neural network	40 × Bagged Neural network: desired $\epsilon = 0.001$, $max = 500$ iterations, 1 hidden layer with 20 neurons.
Bagged Neural network	20 × Bagged Neural network: desired $\epsilon = 0.005$, $max = 200$ iterations, 1 hidden layer with 20 neurons.
Bagged Neural network	40 × Bagged Neural network: desired $\epsilon = 0.005$, $max = 500$ iterations, 1 hidden layer with 20 neurons.

C.5 Style: Best GA Stacking Ensemble

C.5.1 Territoriality

Ensemble learner	**Settings**
Stacking	5 folds, level 2 learner: Bagged (40×) Neural network with desired $\epsilon = 0.001$, $max = 200$ iterations, 1 hidden layer with 20 neurons.
Base learners	**Settings**
PLS regression	$l = 3$
k-nearest neighbors	$k = 20$, $\alpha = 10$, $\delta =$ Euclidean.
Bagged Neural network	40 × Bagged Neural network: desired $\epsilon = 0.001$, $max = 100$ iterations, 1 hidden layer with 10 neurons.
Bagged Neural network	40 × Bagged Neural network: desired $\epsilon = 0.001$, $max = 200$ iterations, 1 hidden layer with 10 neurons.
Bagged Neural network	40 × Bagged Neural network: desired $\epsilon = 0.005$, $max = 50$ iterations, 1 hidden layer with 10 neurons.

C.5.2 Orthodoxity

Ensemble learner	**Settings**
Stacking	6 folds, level 2 learner: Neural network with desired $\epsilon = 0.001$, $max = 200$ iterations, 1 hidden layer with 20 neurons.
Base learners	**Settings**
PLS regression	$l = 3$
k-nearest neighbors	$k = 40$, $\alpha = 10$, $\delta =$ Manhattan.
k-nearest neighbors	$k = 40$, $\alpha = 20$, $\delta =$ Manhattan.
Neural network	Desired $\epsilon = 0.001$, $max = 50$ iterations, 1 hidden layer with 20 neurons.
Bagged Neural network	40 × Bagged Neural network: desired $\epsilon = 0.005$, $max = 200$ iterations, 1 hidden layer with 20 neurons.

C.5.3 Aggressivity

Ensemble learner	Settings
Stacking	6 folds, level 2 learner: Bagged (20×) Neural network with desired $\epsilon = 0.005$, $max = 500$ iterations, 1 hidden layer with 10 neurons.
Base learners	**Settings**
PLS regression	$l = 3$
k-nearest neighbors	$k = 10$, $\alpha = 10$, $\delta =$ Euclidean.
Neural network	Desired $\epsilon = 0.001$, $max = 500$ iterations, 1 hidden layer with 10 neurons.
Bagged Neural network	40 × Bagged Neural network: desired $\epsilon = 0.005$, $max = 100$ iterations, 1 hidden layer with 20 neurons.

C.5.4 Thickness

Ensemble learner	Settings
Stacking	2 folds, level 2 learner: Neural network with desired $\epsilon = 0.005$, $max = 500$ iterations, 1 hidden layer with 20 neurons.
Base learners	**Settings**
Mean regression	—
PLS regression	$l = 2$
Neural network	Desired $\epsilon = 0.005$, $max = 200$ iterations, 1 hidden layer with 10 neurons.
Neural network	Desired $\epsilon = 0.001$, $max = 500$ iterations, 1 hidden layer with 20 neurons.
Bagged Neural network	40 × Bagged Neural network: desired $\epsilon = 0.005$, $max = 500$ iterations, 1 hidden layer with 10 neurons.

C.6 Testing Machine Specification

The software was run on the GNU/Linux operating system with kernel of version 3.4.10. The machine is powered by the 4-core Intel i3 CPU at 2.3 GHz. The fitness evaluations for the Genetic algorithm were written to run in parallel.

D. Strength Attribute Evaluation

Table D.1 gives the first 30 most strongly correlated attributes. Figures D.1 to D.3 show the spatial configurations of the Pattern attributes. The Pearson's r coefficient is negative for attributes mostly played by strong players, and $r > 0$ for attributes that are played by weak players.

Because majority of the attributes in the Table D.1 are negatively correlated, we list some of the positively correlated attributes to complete the picture in Table D.2.

Feature name	**Pearson r**	**Attribute description**
Local sequences	-0.512	An average number of sente sequences, $\omega = 10$
Pattern	-0.480	A one-point jump
Local sequences	-0.457	An average difference between the number of sente and gote sequences, $\omega = 10$
Pattern	-0.455	A horse move
Pattern	-0.446	Jumping ahead
Pattern	-0.438	An attachment, part of a *joseki* sequence
Pattern	0.437	An empty triangle
Pattern	-0.424	A peep at a one-point jump
Pattern	-0.409	A general one-point approach move
Pattern	0.402	An empty triangle
Pattern	-0.398	A one-point approach move on the third line
Pattern	-0.391	An attachment in the corner
Pattern	-0.381	A connection to prevent a cut
Pattern	-0.381	A diagonal move, *kosumi*
Captured stones	0.377	An average number of stones captured within first 60 moves.
Pattern	-0.376	A one-point jump (probably to the center)
Pattern	-0.368	Pushing ahead
Pattern	-0.366	A horse move (*keima*) corner approach
Pattern	-0.364	A peep at a one-point jump
Pattern	-0.360	A horse move approach
Local sequences	-0.358	An average number of gote moves, $\omega = 10$
Pattern	0.351	A solid connection
Pattern	-0.349	A *hane* at one stone
Pattern	-0.348	An attachment to one stone
Pattern	-0.346	A horse move
Pattern	-0.346	Securing life in corner
Pattern	-0.344	Cutting through a horse move
Captured stones	0.343	Average number of stones captured by opponent within first 60 moves.
Win/Loss points	0.342	Average number of points for lost games.
Pattern	-0.341	A horse move approach

Table D.1: List of attributes most strongly correlated with strength.

Feature name	Pearson r	Attribute description
Pattern	0.333	A "weird" shape
Win/Loss stat	0.331	Average difference between number of games lost by points and lost by resignation.
Pattern	0.325	Pushing from behind
Captured stones	0.324	Average number of stones captured by opponent in the middle game (moves $\in \langle 61, 240 \rangle$)
Pattern	0.317	An endgame move after opponent's mistake

Table D.2: List of some other positively correlated attributes.

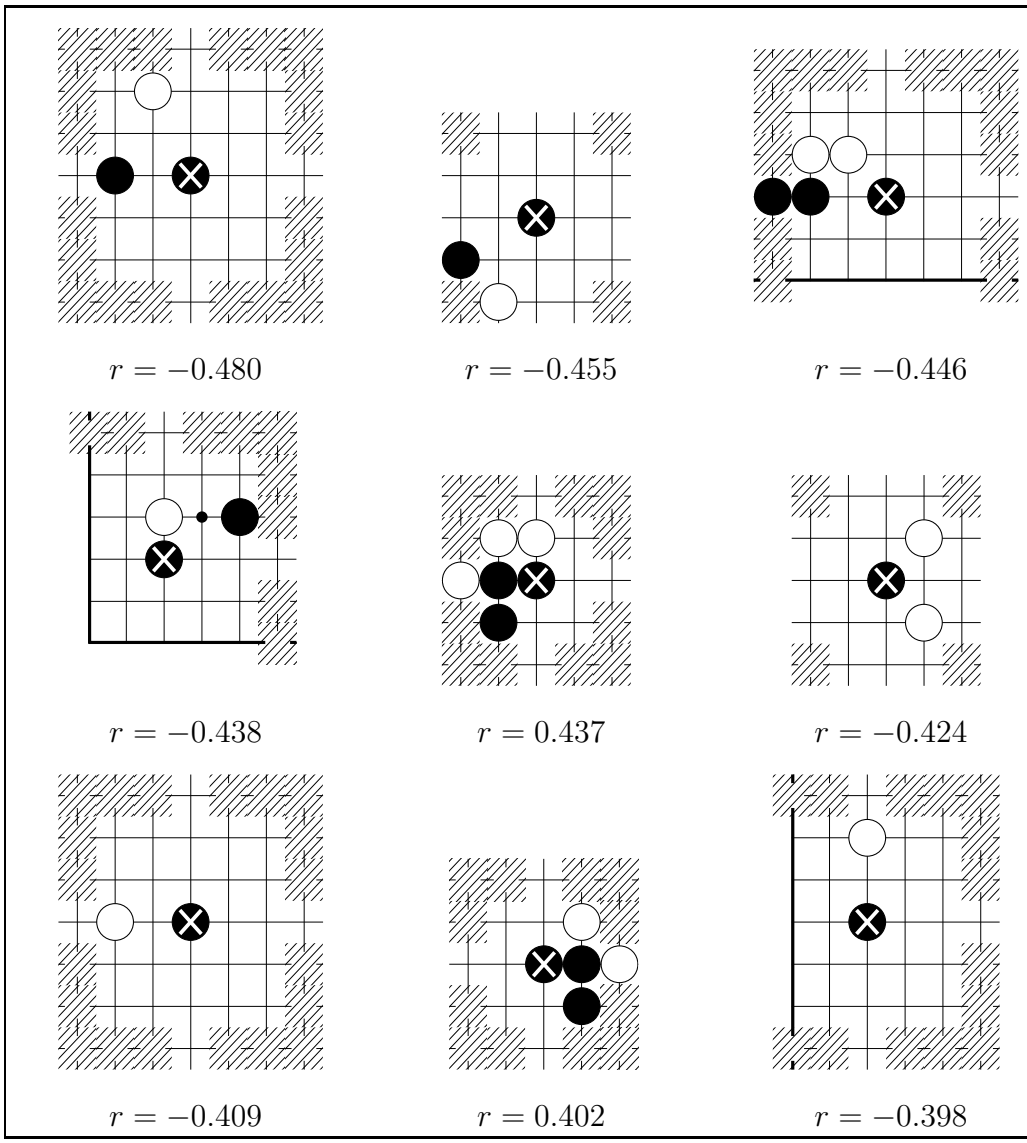


Figure D.1: First 9 most correlated pattern attributes, along with the Pearson's correlation coefficient r.

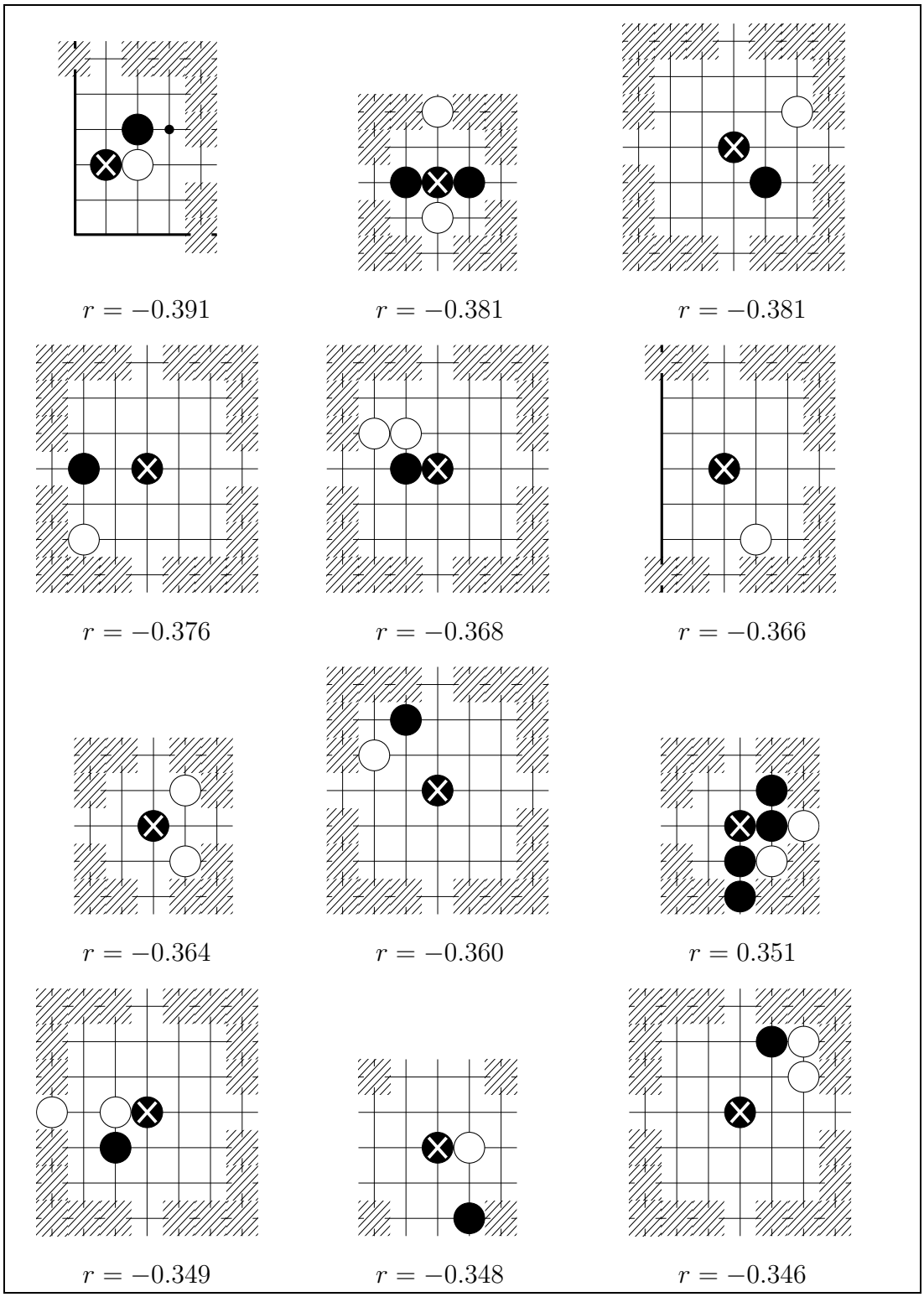


Figure D.2: Next 12 most correlated pattern attributes, along with the Pearson's correlation coefficient r.

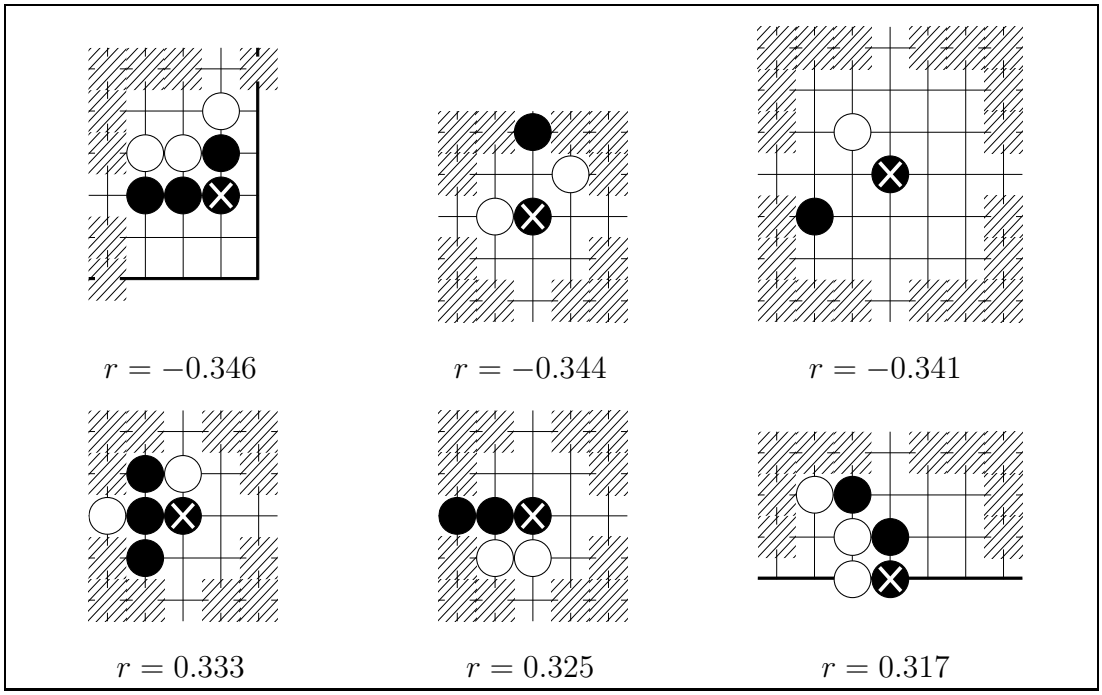


Figure D.3: Further strongly correlated pattern attributes and their Pearson's correlation coefficient r.