Evolving Non-linear Stacking Ensembles for Prediction of Go Player Attributes

Josef Moudřík
Charles University in Prague
Faculty of Mathematics and Physics
Malostranské náměstí 25
Prague, Czech Republic
email: j.moudrik@gmail.com

Roman Neruda
Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2
Prague, Czech Republic
email: roman@cs.cas.cz

***Abstract*—The paper presents an application of non-linear stacking ensembles for prediction of Go player attributes. An evolutionary algorithm is used to form a diverse ensemble of base learners, which are then aggregated by a stacking ensemble. This methodology allows for an efficient prediction of different attributes of Go players from sets of their games. These attributes can be fairly general, in this work, we used the strength and style of the players.**

I. Introduction

The field of computer Go is primarily focused on the problem of creating a program to play the game by finding the best move from a given board position [1]. We focus on analyzing existing game records with the aim of helping humans to play and understand the game better instead. In our previous work [2], we have presented a way to extract information rich features from sets of Go players' games.

This paper presents machine-learning methodology we have devised to fully utilize these extracted features. We have used stacking ensembles with non-linear second-level learner. Both the members of the ensemble and the second-level learner are chosen by a genetic algorithm. The resulting model performs better than any single base-learner (see Section III) on its own, and also better than the best hand-tuned ensemble we have been able to come up with.

As the methods of this work are mainly domain-independent, we only give a brief introduction to Go-related specifics. Go is a two-player full-information board game played on a square grid (usually 19×19 lines) with black and white stones; the goal of the game is to surround territory and capture enemy stones. Different players tend to choose different strategies to achieve this, in Go terminology this is captured by the notion of playing style. Obviously, the players also vary in their proficiency, to which we refer to as *strength*. All these player attributes (various axes of style, and strength) can be mapped on a subset of real numbers [3]. In this paper, we use thusly defined regression domains as black-box datasets.

This paper is organized as follows. Section II gives overview of related work. Section III presents the learners (Sections III-A and III-B) and the genetic algorithm (Section III-C). Section IV discusses performance measures used to compare various learners. Experiments and results are shown in Section V. Section VI discusses applications and future work.

II. Related Work

Ensemble approaches to machine-learning have been in researchers' attention for over two decades. During the time, various approaches appeared. Some of them train one model on differently sampled data [4], as is the case of bagging [5] and the related random forests algorithm [6]. The schemes for combining such models in a voting-like manner seems to be well understood [7]. For example, in neural networks, these ideas have also recently been re-introduced in the form of a dropout [8].

Another approach to combine different models is boosting [9], where a (presumably weak) model is iteratively trained to specialize on hard instances. Stacking [10] on the other hand, uses a two-layered approach, where model on the second level learns to correct for mistakes that first level learners make. For classification, various ways of forming the features from the first level prediction have been proposed ([11], [12]), multi-response linear regression has been found to work well for second level learner. For regression task such as ours, simple linear second level models have been proposed by Breiman [13]. We are not aware of any use of non-linear models for second level predictors like we use in this work.

Prediction of Go player attributes has until recently been limited to pre-defined questionnaires and simple methods [14], [15]. Universal approach to the problem has been introduced by our previous work [2] and [3].

III. Learners

This section presents the machine learning framework we have used. The basic methods are well-documented in literature, so we only give a very brief overview here.

Suppose we have a set of data

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

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.235

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

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 fitting; we call this the *inductive bias* (of the *learner* and the underlying model). 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 (according to some criterion).

Another approach, the one we use in this work, is not to choose the best, but rather try to combine the different models 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 understand this as *ensemble learning*.

A. Base Learners

1) Mean Regression: is a very simple method, which we use as a reference for comparing performances of other learners. It simply outputs the mean of the y's in the training set, and it is thus constant regardless of the input x.

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

2) Artificial Neural Networks: are a standard technique used for function approximation. The network is composed of simple computational units which are organized in a layered topology, as described e.g. in a monograph [16]. We have used a simple feedforward neural network with 20 hidden units in one hidden layer. The neurons have standard sigmoidal activation function and the network is trained using the RPROP algorithm [17] for at most 100 iterations (or until the error is smaller than 0.001). In both datasets used, the domain of the particular target variable (strength, style) was linearly rescaled to $\langle -1, 1 \rangle$ prior to learning. Similarly, predicted outputs were rescaled back by the inverse mapping.

3) k-Nearest Neighbor Regression: is another commonly used machine learning algorithm [18]. 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.*

TABLE I
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	$max \in \{50, 100, 200, 500\}$ iterations $\epsilon \in \{0.001, 0.005\}$, 1 layer with number of neurons $\in \{10, 20\}$, all combinations. Symmetric sigmoid activation function.
Bagged Neural Networks	For ensemble sizes of $\in \{20, 40, 60\}$, each Neural network (from right above) was tested.

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 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 α goes to infinity, the method essentially becomes one-nearest neighbor.

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 [19].

5) Random Forests: utilize an ensemble of tree learners to predict the dependent value [6], [20]. Although this model is an ensemble itself, we treat is as a black box in this work.

Each tree from the forest (of size N) is trained on an independently chosen subset of training data, exactly as the bagging in Section III-B1 does. See the Breiman's paper [6] for details.

B. Ensemble Learners

1) Bagging: is a simple ensemble method introduced in [5]. 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 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.

Paper [5] 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 (see Random Forests above).

2) Stacking: is a more sophisticated approach. The original idea was pioneered by [10] and extended by [12], [13]. 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 learners. The second level is a single learner which aggregates guesses from the 1st level models and outputs the final prediction. Figure 1 shows the topology.

The training dataset is divided into smaller parts (by cross-validation, see Section IV-A). The 1st level learners are trained on some of them and their *generalization* biases are measured by testing their performance on the rest. The 2nd level learner learns to correct these—it learns what the correct output is, given what the 1st level predictors output. Algorithm 1 describes the procedure in more detail. Usually, the 2nd level learner is a simple linear regression, in this work we use non-linear models as well.

Having diverse base learners (various models with different biases) often proves to increase the prediction performance. 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. Usually, all these parameters are hand-tuned; in this work, we use a genetic algorithm (Secion III-C).

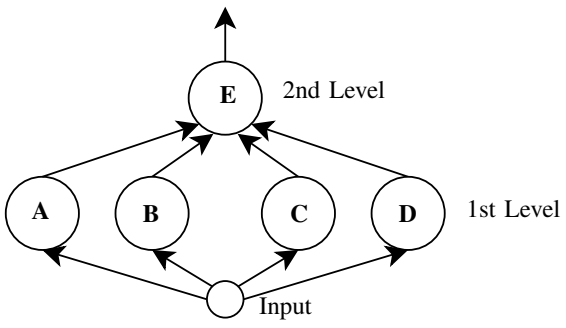

Fig. 1. The topology of the stacking ensemble method. **A**, **B**, **C** and **D** are the level 1 learners, **E** is the level 2 learner.

Algorithm 1: Stacking for Regression

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(`$L1$, $L2$`)`;

C. Evolving Stacking Ensembles

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 [21] 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 III-A—we use both differently parameterized base learners and bagged neural networks.

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 $1 \dots |BL|$, or we change the number of $Folds$ to $2 \dots 6$, (`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 2).

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 2: 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;
```

IV. 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 into parts that are only used for training (Tr) and testing (Ts).

A. 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 [22] for details.

B. Error Analysis

A commonly used performance measure is the mean square error (MSE) which estimates variance of the error distribution. We use its square root ($RMSE$) which is an estimate of standard deviation of the predictions,

$$RMSE = \sqrt{\frac{1}{|Ts|} \sum_{(ev,y) \in Ts} (predict(ev) - y)^2},$$

where the machine learning model $predict$ is trained on the training data Tr and Ts denotes the testing data.

V. Experiments and Results

A. Strength

One of the two major domains we have tested our framework on is the prediction of player strength. In the game of Go, strength (amateur) is measured by kyu (student) and dan (master) ranks. The kyu ranks decrease from about 20th kyu (absolute beginner) to 1st kyu (fairly strong player), the scale continues by dan ranks, 1 dan (somewhat stronger than 1 kyu), to 6 dan (a very strong player).

Dataset: We have collected a large sample of games from the public archives of the Kiseido Go server [23], the sample consists of over 100 000 records of games. The records were

TABLE II
THE PARAMETERS OF THE GENETIC ALGORITHM FOR THE STRENGTH DATASET.

Parameter	Value
Set of base learners BL	Is given in Table I.
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 III
STRENGTH: BEST GA STACKING ENSEMBLE.

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

divided by player's strength and preprocessed as is detailed in [2]. Here, it is enough to note that the dataset consists of 120 independent pairs (x, y) for each of the 26 ranks, where x is the feature vector described in [2] (dimension of x was 1040), and y is the target variable, which is one number describing the rank (1-26).

Results: In the process of finding the best learner, we started with a hand-tuned learner shown in Table IV. Using this learner (which we found to perform reasonably well, as shown in Table V-A) 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 III-C; 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 II.

Unfortunately, the time needed for a single iteration was very large in this setting (see Discussion for more detailed treatment of the time complexity). To speed up the process, we used a sub-sampled 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 principal obstacle

TABLE IV
THE HAND-TUNED LEARNER.

Ensemble learner	Settings
Stacking	4 folds, level 2 learner: NN, $\epsilon = 0.005$, $max = 100$ iter., 1 layer, 10 neurons.
Base learners	**Settings**
Mean regression	—
PLS regression	$l = 3$
k-NN	$k = 50$, $\alpha = 20$, $\delta =$ Manhattan.
Random Forests	$N = 50$
Bagged NN	20 × NN: $\epsilon = 0.001$, $max = 100$ iter., 1 layer, 10 neurons.

TABLE V
REGRESSION PERFORMANCE OF DIFFERENT LEARNERS ON THE FULL DATASET. THE RESULTS WERE COMPUTED USING 5-FOLD CROSS-VALIDATION. PARAMETERS OF THE BEST GA STACKING ENSEMBLE ARE GIVEN IN TABLE III, THE OTHER LEARNERS ARE TAKEN FROM THE INITIAL HAND-TUNED LEARNER FROM TABLE IV.

Learner	RMSE	Mean cmp
Mean regression	7.507	1.00
Random Forrest	3.869	1.94
PLS	3.176	2.36
Bagged NN	2.66	2.82
Hand-tuned learner	2.635	2.85
Best GA stacking ensemble	2.607	2.88

for finding the best learner, since the down-sampling 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. The run of genetic algorithm took on average approximately 2.5 hours per iteration on a 4-core commodity laptop in this setting.

The performance of the best ensemble found by the GA (on the full dataset) are given in Table V-A along with other learners to compare performances. The resulting learner (Table III) is fairly complex as Figure 2 shows. Evolution of the **RMSE** error in time is given in Figure 3.

B. Style

Apart from the strength estimation, we also tested the framework presented in this work to test prediction of playing styles of professional players. Playing style has different aspects, some of them are vaguely defined (for details, see [24]). Moreover, none of them has clear definition in a mathematical sense. To capture these notions at least approximately, we came up with four axes, whose ends correspond to opposing principles in traditional Go knowledge. Next, we used a questionnaire (submitted to domain experts) to find the style values for a set of well known professional players from the 20th century.

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

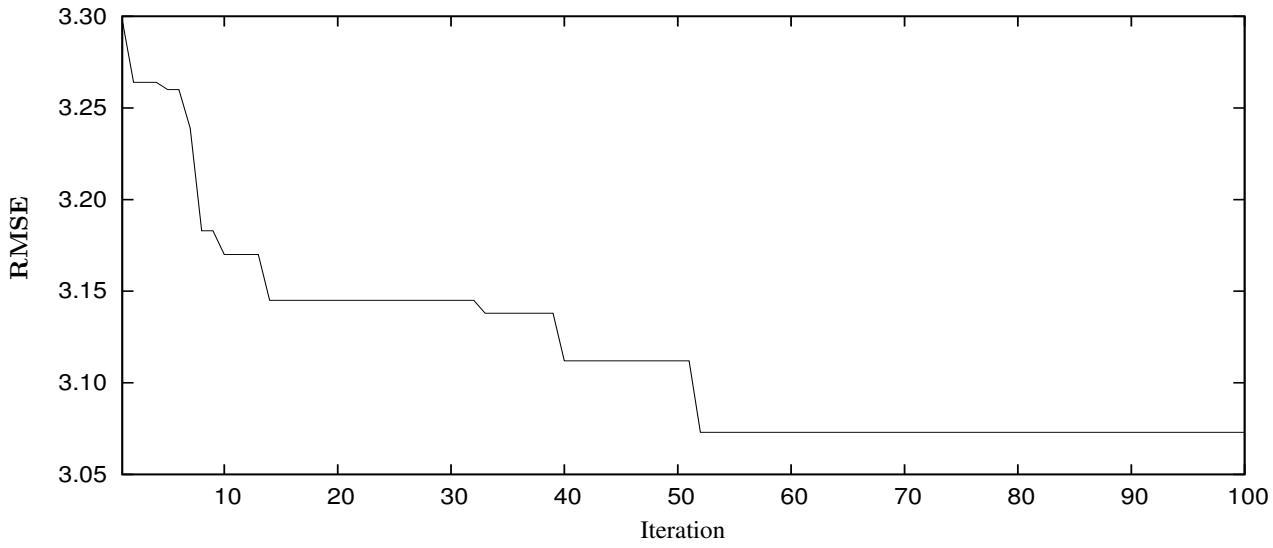

Fig. 3. Evolution of **RMSE** error during the run of the genetic algorithm for finding an optimal stacking ensemble for the (sub-sampled) strength data.

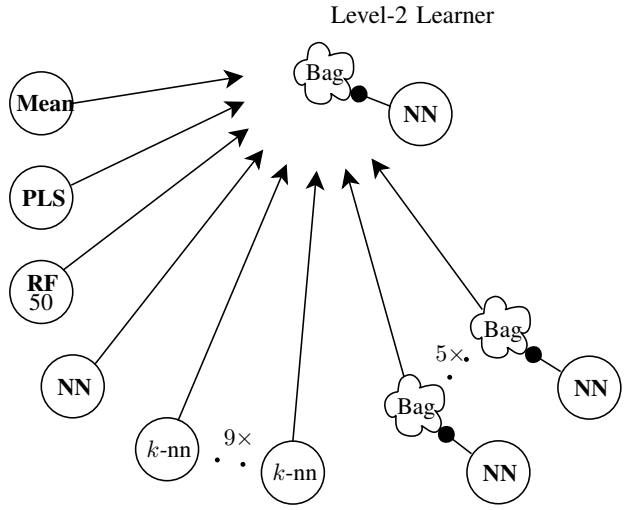

Fig. 2. Structure of the best stacking ensemble found by the Genetic Algorithm. The circle marks a normal learner, with description within, the "cloud" denotes a bagging learner. The corresponding bagged learner is connected using the circle-ended arrow. Precise descriptions of the learners are given in Table III. **Mean** is the Mean regression, **PLS** Partial least squares regression, **RF** Random Forests, **NN** various neural networks and k-nn is obviously the k-nearest neighbor learner.

Dataset: The collection of games in this dataset comes from the Games of Go on Disk (*GoGoD*) database by [25]. 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. [26] 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.

For each of 24 professional players, we obtained 12 pairs (x, y), where x is the feature vector obtained from the games as described in [2] (dimension of x was 640), and y is one of the 4 styles—basically, we view the problem as 4 different regression problems which share the same feature vectors.

Results: We used the genetic algorithm to determine the best ensemble learner. During the process, we have encountered over-fitting problems concerning the very small size of the dataset.

At first, we chose the parameters of the GA to be the same as in the problem of strength (Table II), 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 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 very 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 sub-sampled datasets in each of the iterations were too different to ensure that the best individuals from one iteration would have good chances in the next one. This rendered the genetic algorithm unsuccessful.

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. The parameters of the final genetic algorithm are listed in Table VI.

The performances of the best learners found are given in Table VII. The parameters of the models are omitted for brevity, see [3] for more. Development of the **RMSE** error in time is given in Figure 4. Each run of the genetic algorithm (for different styles) took approximately one hour of CPU time per iteration.

TABLE VI
THE PARAMETERS OF THE GENETIC ALGORITHM FOR THE STYLE DATASET.

Parameter	Value
Set of base learners BL	Is given in Table I.
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 VII
REGRESSION PERFORMANCE OF DIFFERENT LEARNERS ON THE FULL DATASET. THE RESULTS WERE COMPUTED USING 5-FOLD CROSS-VALIDATION.

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

VI. DISCUSSION

The results in both domains show that evolving stacking ensembles non-trivially improves upon the performance of the best hand tunes ensemble, circa by 1.5% for the case of strength and by 4% for the style domains (averaged over different styles). The resulting ensembles also show a lot of diversity.

The main drawback of the methods described is clearly the time consumed, which—even for such relatively small datasets—is in orders of hours per iteration. The main cause of this is the cross-validation performed at various levels, multiplying time complexity. At the outer level, it is run to obtain better error estimates. The cross-validation is also used in the inner loop during training Stacking ensembles, and moreover, in some of the base learners. In this extreme case, the complexity of training the base model is multiplied by number of $Folds^3$. With some effort, the large sequential time could also be exchanged for large number of machines, since it is easy to train different folds in parallel. Furthermore, the time-complexity can be lowered by limiting the number of learners in the ensemble, or by restricting the base learners used by the stacking ensemble to be simple and fast models (with possible loss of precision).

The prediction of player attributes as demonstrated in this work has been (together with the feature extraction presented in [2]) combined in an online web application[1], which evaluates games submitted by players and predicts their playing strength and style. The predicted strength is then used to recommend relevant literature and the playing style is utilized by recommending relevant professional players to review. So far, the web application has served thousands of players and it was generally very well received. 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 [14]—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 [15] suggests, that it computed distances to some professional players based on user's answers to 20 questions regarding the style. We believe that our approach is more precise, both because it takes into account many different aspects of the games [2], and because the methods presented in this paper are able to use the information well.

VII. CONCLUSION

The paper presents machine-learning algorithm which evolves non-linear stacking ensembles of learners. The algorithm is applied on two computer Go domains, prediction of player's strength and different aspects of his style. In both these domains, the algorithm outperforms other methods, with the disadvantage of taking relatively large amount of time; solutions to this problem are proposed.

VIII. IMPLEMENTATION

The code used in this work is released online as a part of GoStyle project [27]. The majority of the source code is implemented in the Python programming language [28].

The machine learning models were implemented and evaluated using the Orange Datamining suite [29] and the Fast Artificial Neural Network library FANN [30]. We used the Pachi Go engine [31] for the raw game processing.

Acknowledgment

This research has been partially supported by the Czech Science Foundation project no. 15-18108S. J. Moudřík has been supported by the Charles University Grant Agency project no. 364015 and by SVV project no. 260 224.

[1] http://gostyle.j2m.cz

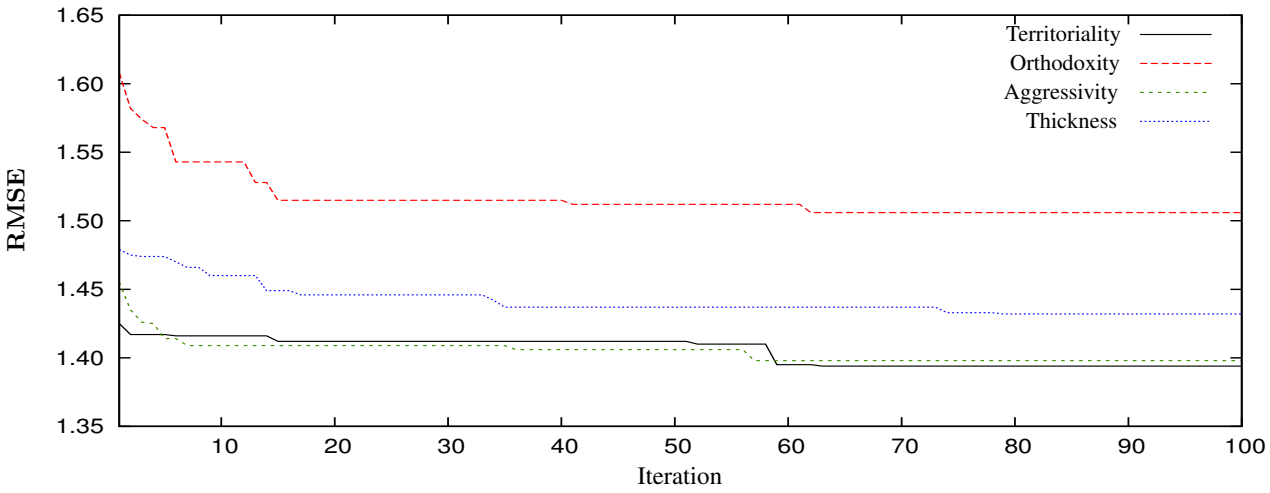

Fig. 4. Evolution of **RMSE** error during the run of the genetic algorithm for finding an optimal stacking ensemble for the style data.

REFERENCES

[1] S. Gelly and D. Silver, "Achieving master level play in 9x9 computer go," in *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*. AAAI Press, 2008, pp. 1537–1540.

[2] J. Moudřík, P. Baudiš, and R. Neruda, "Evaluating go game records for prediction of player attributes," in *IEEE Computational Intelligence in Games 2015*. IEEE, 2015, pp. 162–168, in Print.

[3] J. Moudřík, "Meta-learning methods for analyzing go playing trends," Master's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 2013. [Online]. Available: http://www.j2m.cz/~jm/master_thesis.pdf

[4] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 10, pp. 993–1001, 1990.

[5] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug. 1996. [Online]. Available: http://dx.doi.org/10.1023/A:1018054314350

[6] ——, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[7] J. Kittler, M. Hatef, R. P. Duin, and J. Matas, "On combining classifiers," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 3, pp. 226–239, 1998.

[8] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[9] Y. Freund and R. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *Computational Learning Theory*, ser. Lecture Notes in Computer Science, vol. 904. Springer Berlin Heidelberg, 1995, pp. 23–37. [Online]. Available: http://dx.doi.org/10.1007/3-540-59119-2_166

[10] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, pp. 241–259, 1992. [Online]. Available: http://www.machine-learning.martinsewell.com/ensembles/stacking/Wolpert1992.pdf

[11] K. M. Ting and I. H. Witten, "Issues in stacked generalization," 1999.

[12] P. K.-W. Chan, "An extensible meta-learning approach for scalable and accurate inductive learning," Ph.D. dissertation, Columbia University, 1996.

[13] L. Breiman, "Stacked regressions," *Machine Learning*, vol. 24, pp. 49–64, 1996. [Online]. Available: http://dx.doi.org/10.1007/BF00117832

[14] A. Dinerchtein. (2012) What is your playing style? [Online]. Available: http://style.baduk.com

[15] Sensei's Library. (2013) Which pro do you most play like. [Online]. Available: http://senseis.xmp.net/?WhichProDoYouMostPlayLike

[16] S. Haykin, *Neural Networks: A Comprehensive Foundation (2nd Edition)*, 2nd ed. Prentice Hall, jul 1998. [Online]. Available: http://www.worldcat.org/isbn/0132733501

[17] M. Riedmiller and H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," in *IEEE International Conference on Neural Networks*, 1993, pp. 586–591.

[18] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[19] R. Rosipal and N. Krmer, "Overview and recent advances in partial least squares," in *in Subspace, Latent Structure and Feature Selection Techniques, Lecture Notes in Computer Science*. Springer, 2006, pp. 34–51. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.85.7735

[20] L. Breiman, J. H. Friedman, O. R. A., and C. J. Stone, *Classification and regression trees*. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[21] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[22] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection." Morgan Kaufmann, 1995, pp. 1137–1143.

[23] W. Shubert. (2013) KGS archives — kiseido go server. [Online]. Available: http://www.gokgs.com/archives.jsp

[24] J. Fairbairn. (winter 2011) Games of Go on Disk — GoGoD Encyclopaedia and Database, Go players' styles. [Online]. Available: http://www.gogod.co.uk/

[25] T. M. Hall and J. Fairbairn. (winter 2011) Games of Go on Disk — GoGoD Encyclopaedia and Database. [Online]. Available: http://www.gogod.co.uk/

[26] P. Baudiš and J. Moudřík, "On move pattern trends in a large go games corpus," *Arxiv, CoRR*, October 2012. [Online]. Available: http://arxiv.org/abs/1209.5251

[27] J. Moudřík and P. Baudiš. (2013) GoStyle — Determine playing style in the game of Go. [Online]. Available: http://gostyle.j2m.cz/

[28] Python Software Foundation. (2008, November) Python 2.7. [Online]. Available: http://www.python.org/dev/peps/pep-0373/

[29] J. Demšar *et al.*, "Orange: Data mining toolbox in python," *Journal of Machine Learning Research*, vol. 14, pp. 2349–2353, 2013. [Online]. Available: http://jmlr.org/papers/v14/demsar13a.html

[30] S. Nissen, "Implementation of a fast artificial neural network library (fann)," Department of Computer Science University of Copenhagen (DIKU), Tech. Rep., 2003, http://fann.sf.net.

[31] P. Baudiš *et al.* (2012) Pachi — Simple Go/Baduk/Weiqi Bot. [Online]. Available: http://repo.or.cz/w/pachi.git