Mach Learn (2018) 107:177–207
https://doi.org/10.1007/s10994-017-5682-0

Discovering predictive ensembles for transfer learning and meta-learning

Pavel Kordík[1] · **Jan Černý**[1] · **Tomáš Frýda**[1]

Received: 10 May 2016 / Accepted: 4 October 2017 / Published online: 21 December 2017

Abstract Recent meta-learning approaches are oriented towards algorithm selection, optimization or recommendation of existing algorithms. In this article we show how data-tailored algorithms can be constructed from building blocks on small data sub-samples. Building blocks, typically weak learners, are optimized and evolved into data-tailored hierarchical ensembles. Good-performing algorithms discovered by evolutionary algorithm can be reused on data sets of comparable complexity. Furthermore, these algorithms can be scaled up to model large data sets. We demonstrate how one particular template (simple ensemble of fast sigmoidal regression models) outperforms state-of-the-art approaches on the Airline data set. Evolved hierarchical ensembles can therefore be beneficial as algorithmic building blocks in meta-learning, including meta-learning at scale.

Keywords Meta-learning · Ensemble learning · Evolutionary algorithms · Evolutionary programming · Combining classifiers · Regression models · Model blending · Automatic algorithm selection · Map reduce

1 Introduction

In predictive modeling, the algorithm selection step is often responsible for sub-optimal results and/or computational complexity of the modeling stage.

Many algorithm selection approaches (Kordík et al. 2011; Sutherland et al. 1993; Bensusan and Kalousis 2001; Botia et al. 2001) simply identify the best algorithm from a set of candidates. This set of candidate algorithms needs to be constructed first. The prevailing approach is to pick candidates manually from a set of available algorithms. However algorithm performance evaluation involves multiple runs of the algorithm (e.g. cross validation)

Editors: Pavel Brazdil and Christophe Giraud-Carrier.

✉ Pavel Kordík
pavel.kordik@fit.cvut.cz

[1] FIT CVUT, Thakurova 9, Praha 6, Czech Republic

and it is very time consuming. The number of candidate algorithms is high even when only default parameter settings for individual algorithms are considered.

In machine learning, parameters of algorithms are quite important, many of them having direct impact on plasticity of generated predictive models. Often, candidate algorithms are evaluated with their default parameter settings. More sophisticated algorithm selection approaches include parameter optimization as part of the selection process. Recent studies have showed the potential of Bayesian methods (Hutter et al. 2011) outperforming both random search (Bergstra et al. 2011) and grid search (Coope and Price 2001).

When ensembles of algorithms (Brown et al. 2006) are taken into account (and they should be considered because of their superb performance on many predictive tasks Hoch 2015; Stroud et al. 2012) the problem of algorithm selection becomes even more difficult. There is a potentially an infinite number of possible candidate algorithms, their parametrizations and ensembles to choose from. Furthermore, the generalization performance of algorithms is not the only quality criterion. For large scale machine learning tasks, the algorithm run-time is of great importance.

It is necessary to take into account the computational complexity of the algorithm selection process. It is not feasible to run all candidate algorithms on a new data set to select the best performing one, simply because there are an infinite number of available algorithms.

One approach to select a training algorithm for a new data set in a reasonable time is to use a meta-data collected during training on similar data sets. Meta-learning approaches (Kordík et al. 2011) utilizing meta-data have been studied intensively in the past few decades. They can predict performance of algorithms on new data sets and consistently is posible to select good performing algorithm among multiple candidates.

The majority of meta-learning approaches (Kordík et al. 2011; Sutherland et al. 1993; Bensusan and Kalousis 2001; Botia et al. 2001) simply select one from a set of few predefined fully specified data mining algorithms. Selected algorithm produces models with the best generalization performance for the given data. Later methods incorporate a further algorithm selection step on the top of recommendations (Sun and Pfahringer 2013).

More advanced meta-learning approaches combine algorithm selection and hyperparameter optimisation such as CASH (Thornton et al. 2013) elaborated within the *INFER project*. In Salvador et al. (2016a, b, c) data mining workflows are optimized including data cleaning and preprocessing steps, together with selected hyperpameters of modeling methods.

We focus on modeling stage only and optimize structure of algorithmic ensembles together with their hyperparameters as explained in Sect. 5. In this way, we can discover new algorithmic building blocks.

The hierarchical structure of algorithmic ensembles is represented by *Meta-learning algorithm templates* introduced in Sect. 5. Our templates indicate which learning algorithms are used and how their outputs are fed into other learning algorithms.

We use the genetic programming (Koza 2000) to evolve the structure of templates and a parameter optimization to adjust parameters of algorithms for specific data sets. Our approach allocates exponentially more time for evolution of above-average templates which is similar to the Hyperband approach (Li et al. 2016). In later stages of our algorithm, templates that survived from previous generation have more and more time to show their potential.

We show that evolved templates can be successfully used to generate models on similar data sets.

Furthermore, templates evolved (discovered) on small data sets can be reused as building blocks for large data sets.

Building predictive models on large data samples is a challenging task. Learning time of algorithms often grows fast with the number of training data samples and dimensionality of

a data set. Hyper-parameter optimization can help us to generate more precise models for given task, but it adds significant computational complexity to the training process. We show that templates evolved on small data subsamples can outperform state of the art algorithms including complex ensembles in terms of performance and scalability.

The next section discusses related work and shows how recent results in the field of meta-learning and automated machine learning are relevant for our research. Before we define the concept of meta-learning templates in Sect. 4, we need to describe building blocks of our templates (base algorithms and ensemble methods in Sect. 3). Introduction to templates is followed by a brief explanation of the evolutionary algorithm designed to evolve templates (Sect. 5). Experiments described in later sections aim to show that hierarchical templates can outperform standard ensembles (on standard benchmarking data samples in Sect. 6), they can be used for transfer learning and scaled up for large scale modeling (Sect. 9).

2 Related work

This contribution is tightly related to meta-learning. The definition of meta-learning is very broad. One of the early machine learning related definition (Vilalta and Drissi 2002) states that a meta-learning system must include a learning subsystem, which adapts with experience.

Another definition (Brazdil et al. 2009) requires meta-learning to start at a higher level and be concerned with accumulating experience over several applications.

Finally according to Vanschoren (2010) meta-learning monitors the automatic learning process itself and tries to adapt its behaviour to perform better.

2.1 Knowledge base meta-learning approaches and workflows

One of the main direction in meta-learning is constructing a meta-level system utilizing a knowledge repository (Vanschoren et al. 2012; Brazdil et al. 2009). The repository is intended to store a meta-data describing problem being solved and the performance of base learners.

Then for any new problem, one looks at problems with similar meta-data to select best performing algorithms (Kordík et al. 2011). *ESPRIT Statlog* (Sutherland et al. 1993) compared the performance of numerous classification algorithms on several real-world data sets. In this project, metadata (statistical features describing the data sets) were used for algorithm recommendation. The *MetaL project* (Bensusan and Kalousis 2001), built upon Statlogs outcomes, utilized landmarking (Pfahringer et al. 2000) metadata (results of fast algorithms, executed on a data set in order to determine its complexity). Ranking of algorithms can be obtained by fast pairwise comparisons (Leite and Brazdil 2010) just on the most useful cross-validation tests (Leite et al. 2012). Another project was *METALA* (Botia et al. 2001), an agent-based distributed data mining system, supported by meta-learning. Again, the goal was to select from among available data mining algorithms the one producing models with the best generalization performance for given data.

The problem with recommending a particular algorithm is that the portfolio of algorithms is potentially infinite. Especially "Frankenstein" ensembles winning Kaggle competitions (Puurula et al. 2014) are good example how complex the topology of machine learning ensembles can be. In Bonissone (2012) so called lazy meta-learning is applied to create customized ensembles on demand. Individual models, ensembles and combination of ensembles in time series forecasting can be selected adaptively (Lemke and Gabrys 2010) by meta-learning.

Recommendation and optimization of data mining workflows (Grabczewski and Jankowski 2007; Jankowski 2013; Sun et al. 2013) is another important research direction aiming at

automation in data science. In this article, we optimize hierarchical modeling templates that are more general than simple ensembles but still narrow enough when compared to universal data mining templates including data preparation. Planning and optimization of full data mining workflows is also elaborated in Nguyen et al. (2014); Kietz et al. (2012), where meta model and AI planer are combined. On the contrary, we focus on predictive modeling stage only and we extend the search to the domain of hierarchical ensembles of predictive algorithms.

2.2 Ensembling as meta-learning

Even simple model ensembling methods such as Boosting (Schapire 1990), Stacking (Wolpert 1992) or Cascade generalization (Gama and Brazdil 2000) can be considered meta-learning methods with respect to the above definitions of the meta-learning. They all use information from previous learning steps to improve the learning process itself. There are many more ensembling approaches and these can be even further combined in a hierarchical manner resembling structures in the human brain as we show in this article.

Theoretical derivation and experimental confirmation that hierarchical ensembles are the best performing option for some classification problems can be found in Ruta and Gabrys (2002, 2005).

Aggregation or hierarchical combination of ensembles has been studied (Analoui et al. 2007; Costa et al. 2008; Sung et al. 2009) intensively not only in predictive modeling. In particular, gradient boosting (Friedman 2000) and multi-level stacking of neural networks (Bao et al. 2009) were parts of the winning solution in the Netflix competition (Töscher and Jahrer 2009; Bennett et al. 2007).

These hierarchical ensembles are single purpose architectures often tailored to one particular problem (data set), where they exhibit excellent performance, but very likely fail with different data. The prevailing approach to constructing these ensembles is manual trial-and-error combined with extensive hyper-parameter optimization.

2.3 Growing ensembles and their optimization

One of the first growing ensembles introduced was the GMDH MIA approach (Mueller et al. 1998) that can be also considered as adaptive layered stacking of models. Our GAME neural networks (Kordík 2009) grow inductively from data to match the complexity of given task and maximize the generalization performance.

Another growing ensemble of neurons (or network) called NEAT (Stanley and Miikkulainen 2001) was primary designed for reinforcement learning controllers. Evolutionary approaches are used to optimize topology and parameters of these ensembles.

When it comes to optimization of ensembles, genetic programming was also used to evolve trees of ensemble models, as suggested in Hengpraprohm and Chongstitvatana (2008), but only to a limited degree with only one type of ensemble, and the article deals with the Cancer data only.

Interesting approach to ensemble building (Caruana et al. 2004) is to prepare ensembles from libraries of models generated using different learning algorithms and parameter settings of algorithms.

The Neural Network ensembling method (GEMS), proposed in Ulf Johansson (2006) trains models independently, then combines them using genetic programming into trivial hierarchical ensemble using weighted average. Weights are evolved by means of genetic programming rather than derived from model performance as in Boosting for instance.

Multi-component, hierarchical predictive systems can be constructed by a grammar-driven genetic programming in Tsakonas and Gabrys (2012) an approach very similar to ours. They used very limited ensembling templates, trivial base models and focus on maintaining diversity during evolution. We focus more on time efficiency.

Some of the modern scalable neural networks (Buk et al. 2009; Smithson et al. 2016; Fernando et al. 2016) can be constructed using indirect encoding. You can evolve structures at macro level (Real et al. 2017) optimizing large building blocks or at micro level (Zoph and Le 2016) optimizing internal structure of neuron cells. Neuroevolution of deep and recurrent networks (Miikkulainen et al. 2017; Rawal and Miikkulainen 2016) is computationally expensive but results looks very promising.

In predictive modeling and supervised learning, it is often more efficient to optimize continuous parameters of algorithms independently of the topology (in contrast to TWEANN approach Stanley and Miikkulainen 2001). Most popular approach for continuous hyperparameter optimization is a simple brute force grid search or random search (Bergstra and Bengio 2012). More sophisticated approaches are based on Bayesian methods (Salvador et al. 2016b). Recently introduced Bandit based method HyperBand (Li et al. 2016) uses performance of base learners to speed up the learning process and can be therefore considered a meta-learning approach. When learning of models can be prematurely terminated, we can save significant amount of resources, speeding up learning by giving more resources to promising learners. Disadvantage of this approach can be that complex models need more time to adapt and it is hard to estimate their final performance in early stages of learning.

The CASH approach in Auto-WEKA (Thornton et al. 2013) combines algorithm selection and hyperparameter optimization (Hutter et al. 2011) in the classification domain. In our approach we optimize the topology of ensembles as well.

2.4 Scalable meta-learning

The proper topology for a given problem is particularly important when machine learning models are evaluated by multiple criteria. The most important criterion is the generalization performance, but often also time of model training/recall should be taken into consideration (Chan and Stolfo 1997; Sonnenburg et al. 2008).

Our results on the Airline data set suggests that simple ensemble of sigmoid models can significantly outperform deep learning models (Arora et al. 2015) when it comes to scalability and learning efficiency.

A recent paper on large scale evolution of image classifiers (Real et al. 2017) is another example of time sensitive approach, where one can trade-off generalization performance for learning/recall speed.

Anytime learning (Grefenstette and Ramsey 2014) aims at building algorithms capable of returning best possible solution given a training time. Anytime ensembling methods such as Speedboost (Grubb 2014) can not only generate approximate models rapidly from weak learners, but they are capable of using extra time resources, when available, to further improve their performance. In this manner, we developed evolutionary search in Sect. 5.

Before discussing anytime optimization of our ensembles, we describe the building blocks and ensembling mechanisms used.

3 Base algorithms and ensembling strategies

We build ensembles from fast weak learners (Duffy and Helmbold 1999). Many of our base models resemble neurons with different activation functions. We can use base models to construct both classification and regression ensembles. In this article, we focus on classification tasks only, however regression models can also be present in classification ensembles.

The classification task itself can be decomposed into regression subproblems by separation of single classes from the others. These binary class separation problems can be approximated by regression models—by estimating continuous class probabilities. The maximum probability class is then considered as output value. The classifier consisting of regression models is further referred to as ***ClassifierModel***.

3.1 Base algorithms

Training regression models (as components of probabilistic classifiers) is fast and straightforward. We use several activation functions in simple perceptrons, namely ***Sigmoid, SigmoidNorm, Sine, Polynomial, Gaussian, Exponential*** and ***Linear***.

To train coefficients of linear or polynomial models, the General Least Squares method (Marquardt 1963) is applied. For models that are non-linear in their coefficients, an iterative optimization process is needed. We compute analytic gradients of error for all fast regression models and employ quasi-Newton method (Shanno 1970) to optimize their parameters.

The ***LocalPolynomial*** base model as well as ***Neural Network (NN), Support Vector Machine (SVM), Naive Bayes classifier (NB), Decision Tree (DT), K-Nearest Neighbor (KNN)*** were adopted from the RapidMiner environment (RapidMiner).

3.2 Ensembling algorithms

The performance of models can often be further increased by combining or ensembling (Brazdil et al. 2009; Kuncheva 2004; Wolpert 1992; Schapire 1990; Woods et al. 1997; Holeňa et al. 2009) base algorithms, particularly in cases where base algorithms produce models of insufficient plasticity or models overfitted to training data (Brown et al. 2006).

A detailed description of the large variety of ensemble algorithms can be found in Brazdil et al. (2009). We briefly describe the ensembling algorithms that are used in our experiments. ***Bagging*** (Breiman 1996) is the simplest one; it selects instances for base models randomly with repetition and combines models with simple average. ***Boosting*** (Schapire 1990) specializes models on instances incorrectly handled by previous models and combines them with a weighted average. ***Stacking*** (Wolpert 1992) uses a meta model, which is learned from the outputs of all base models, to combine them. Another ensemble utilizing meta models is the ***Cascade Generalization*** (Gama and Brazdil 2000), where every model except the first one uses a data set extended by the output of all preceding models. ***Delegating*** (Ferri et al. 2004) and ***Cascading*** (Alpaydin and Kaynak 1998; Kaynak and Alpaydin 2000) both use a similar principle: they operate with certainty of model output. The latter model is specialized not only in instances that are classified incorrectly by previous models, but also in instances that are classified correctly, but previous models are not certain in terms of their output. Cascading only modifies the probability of selecting given instances for the learning set of the next model. ***Arbitrating*** (Ortega et al. 2001) uses a meta-model called referee for each model. The purpose of this meta-model is to predict the probability of correct output. All methods used in this study were implemented within the FAKE GAME open source project (Fake Game).

4 Meta-learning templates

The meta-learning template (Kordík et al. 2011) is a prescription how to build hierarchical supervised models. In the most complex case, it can be a collection of ensembling algorithms and base algorithms combined in a hierarchical manner, where base algorithms are leaf nodes connected by ensembling nodes. Regression models or classifiers deeper in the hierarchy can be more specialized to a particular subset of data samples or attributes. This scheme decomposes the prediction problem into subproblems and combines the final solution (model) from subsolutions. The procedure of problem decomposition depends on ensembling methods. Typically, it distributes data to member models and when all outputs are available, they are combined to the ensemble output.

Note that meta-learning templates are not data mining models, but algorithms. Models are produced when templates are executed.

Figure 1 shows an example of a meta-learning template. When executed, the full training data set is passed to a top level Bagging that generates 4 bootstrap training data sets for members of the ensemble. The second bootstrap training data set is used to train a KNN classifier by Boosting and samples, where this classifier demonstrates high error, are more likely to be used in the training set for the second member model of the Boosting: the stacking of NN and DT classifiers. Bottom level NN and DT are evaluated on the training data and upon their responses a SVM meta-model is trained. The Stacking is evaluated and a weight is assigned to its output in Boosting. The output of Boosting is averaged with the other three top level base models and the whole classifier is finished (see the left-hand tree in Fig. 2).

The resulting classifier is depicted in Fig. 2. The tree in the center shows how the input attributes are presented to the model. The propagation of input vector is straightforward in this example, but some ensembles (e.g. Cascading) involve evaluation of member models (their outputs are added to input vectors of subsequent models). The right-hand tree shows how outputs of base models are blended to produce the final output.

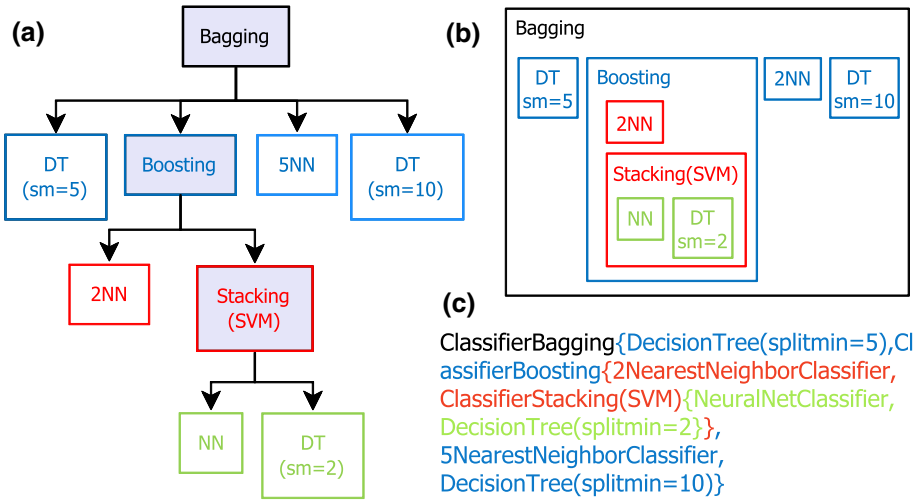

Fig. 1 An example of hierarchical combination of algorithms. Using this meta-learning template, a classifier can be produced (see Fig. 2). The template can be represented by **a** a tree, **b** embedded boxes or **c** by text

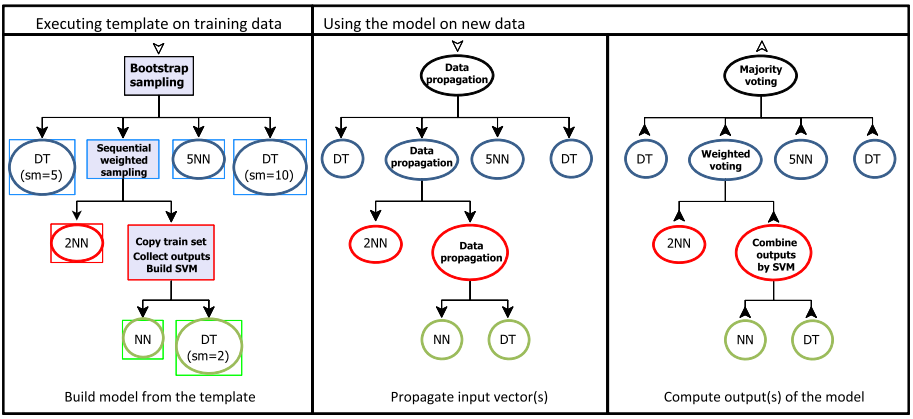

Fig. 2 An ensemble classifier can be produced by the hierarchical combination of algorithms depicted in Fig. 1. Executing the template will distribute data to leaf base models according to procedures specified by ensembling algorithms. Base models and ensembles are constructed until the root ensemble (base model) is finished. Using the model involves propagating and presenting an input vector to leaf models and combining their outputs by ensembling procedures

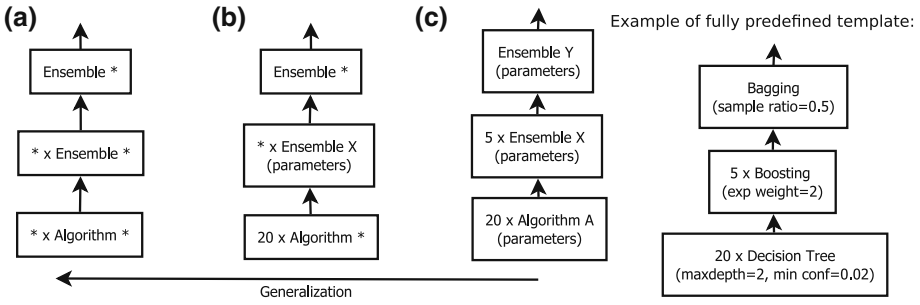

Fig. 3 Nested ensembles can be represented by a template. Using wildcards, specific (or predefined in other words) template can be generalized to represent set of templates

Whereas data mining workflows are directed acyclic graphs, meta-learning templates are hierarchical structures. Inner nodes in our templates are ensembling algorithms and leaf nodes are base algorithms. Fully predefined templates are algorithm configurations containing parameters of both ensembles and base algorithms. Templates can be generalized using wildcards (see Fig. 3) to represent a subspace of the search space of topologies and parametrizations of hierarchical ensembles.

Similarly to the Holland's schema theorem (Holland 1975), we can define fitness of a template as average/maximum fitness of individual algorithms represented by this particular template. Wildcards here are used just as placeholders for random decisions on type of ensembles or base algorithms and their parameters. On the contrary, in rooted tree schema theory (Rosca 1997) wildcards represent sub-trees.

5 Discovering templates

The meta-learning template can be designed manually using an expert knowledge (for example, bagging boosted decision trees showed good results on several problems) so it is likely to perform well on a new data set. This is however not guaranteed.

In our approach we optimize templates on data sub-samples using a genetic programming (Koza 2000). In this way, we can search the space of possible architectures of hierarchical ensembles and optimize their parameters simultaneously.

5.1 Evolving templates by genetic programming

Applying genetic programming or grammatical evolution involves resolving (a) representation of individual, (b) design of genetic operators and evolution, (c) fitness function formulation and (d) construction of initial population.

Algorithm 1 Evolve Meta-learning Template

```
Function EvolveTemplate(maxTime, useMetaDB)
    Data: Dataset data
    Result: Meta-learning template for data
    if Big (data) then
        data_sample = Reduce (data, size)
    if useMetaDB then
        Generate initial population using the Metadatabase
    else
        Initialize population randomly from a minimal form
    while (time < maxTime ) do
        generation = 1
        while (generation < max_generations ) do
            Evaluate fitness of individuals on data_sample
            if stagnation then
                generation = max_generations
            Select individuals in tournament
            Apply mutations
            Prepare new generation
        if size < getSize(data) then
            size = 2 * size
            data_sample = Reduce (data, size)
    return best_template
```

5.1.1 Encoding templates to chromosomes

Encoding is straightforward, because in genetic programming (GP) individuals are represented as trees. Each specific template has ensembles in inner nodes and base algorithms in leaf nodes whereas their parameters are associated with corresponding nodes (not encoded as individual nodes as in Koza's representation). Generalized templates contain wildcards in their chromosomes. Wildcards are represented as lists of genes. One of these genes is randomly selected when an individual should be produced from a template. Meaning when 20 base models should be generated, the heuristics selects randomly twenty times from list of available algorithms.

5.1.2 Adaptive control for anytime learning

The pseudo-code of Algorithm 1 shows how to evolve meta-learning templates. There are two parameters, a time limit for the algorithm and an attribute that decides whether a metadatabase should be used to streamline the evolution. Later we discuss advantages and disadvantages of using the metadatabase.

The algorithm has several internal parameters and many of them are adaptive. Time limit influence most of internal parameters, because only fast templates on small data samples can be evaluated for small time allocations. With more time available the search for best performing meta-learning templates can intensify and explore bigger part of the search space.

The algorithm receives a data set as an input. When the data set has more than 200 dimensions or 500 instances (constants experimentally chosen based on results on several data sets), a sample is generated using random subsampling or stratified sampling in case of small or imbalanced data. We sample both instances and attributes when constrains are violated to get a representative data subset.

5.1.3 Initial population and subsequent evolutions

An initial population of the first evolution (generalized templates) is generated from a minimal form, similarly to Stanley and Miikkulainen (2001); Mueller et al. (1998). In case that the metadatabase is not used, base models form the population. The advantage is that each type of base model is considered before ensembles are taken into account. Also the population grows from a minimal form. With the metadatabase, the initial population is filled by best individuals from most similar meta data (pairwise similarities of attributes statistics). For subsequent evolutions, we use population from the last epoch of the previous evolution.

While time is available, we run a sequence of evolutions that are gradually exploring the state space of possible templates. The first evolution runs on a small data sample (200×500 maximum) and after maximum of hundred generations (or when a stagnation is detected), data is doubled (both dimensionality and numerosity if possible) and next evolution follows. In each subsequent evolution, templates are more specific and the percentage of wildcards decrease. Also, ranges of explored parameters increase as templates get more precise and specific.

This is quite similar to the Hyperband approach (Li et al. 2016), when exponential more time is given to perspective learners. Here, many template topologies are eliminated on a small data subset. Just the most successful templates are examined on larger sets and their parameters are extensively finetuned.

The optimization process is designed to be time-constrained. For each algorithm, we estimated its scalability so that we can predict the run-time given size of a data set and parameter settings. Parameters like maximum template depth, maximum allowed computational complexity of template, intervals of base algorithm parameters, size of data samples are then increased adaptively set based on time available. Similarly as in Li et al. (2016), we give exponential allocation to search in promising parts of the state space. For particular details see [Software: Fake game, data mining software (https://fakegame.sourceforge.net/)], (Kordík et al. 2014) or our open source implementation (Kordík 2006).

5.1.4 Fitness evaluation

Fitness evaluation is also time-effective. It is estimated by a multiple crossvalidation (CV) (Browne 2000; Kordík et al. 2014). The fitness of a template is proportional to the average performance of models generated on training folds and evaluated on testing folds, while the data is divided into folds multiple times. We need a reliable estimate of a generalization performance of models/templates even when time allocated for the optimization is very short (e.g. 1 min). For short time allocations, data samples are small (up to 300 instances) and repeated CV runs are necessary to reduce variance of cross-validation estimates. With more

time available, our fitness estimates are refined with additional fitness evaluations starting with the most promising estimates with high variation of recent evaluations. An approach similar to Moore and Lee (1994) or (Li et al. 2016) helping us to allocate additional resources for promising candidates.

After fitness evaluation, the selection is implemented by a tournament. We do not use crossover, just mutations similar to the approach used in a standard GP (Koza 2000). Mutations grow/modify both topology and parameters of templates.

Structural mutations are realized using the context free grammar (Whigham 1995) rules shown in Table 1 defining how templates can grow from simple base classifiers to large hierarchical ensembles often containing regression ensemble sub-trees.

Parameters of a node are mutated by applying Gaussian noise to the current value. The mutation probabilities and distribution of noise are controlled by adaptive parameters for anytime learning.

Exploration versus exploitation capabilities of evolutions are ale influenced by adaptive mutation probabilities and intervals. For exact parameter settings and adaptation strategies please consult (Fake Game; Kordík et al. 2014).

Table 1 Context free grammar rules

template → **classify**
classify → **baseClassification**|**metaClass**
metaClass → $ClassBagging$(**classify**^{n})|$ClassBoosting$(**classify**^{n})
metaClass → $ClassCascadeGen$(**classify**^{n})|$ClassStacking$(**classify**$^{n+1}$)
metaClass → $ClassArbitrating$(**classify**^{n}); where n is number of classifiers in ensemble
baseClassification → **classifierModel**|$NeuralNetClassifier$|$DecTree$| KNN|SVM|$NaiveBayes$|$Polynomial$|$Sine$
classifierModel → **regression**^{c} ; where c is the number of classes
regression → **baseRegression**|**metaReg**
metaReg → $RegBagging$(**regression**^{n})|$RegBoostingRT$(**regression**^{n})
metaReg → $RegStacking$(**regression**, **regression**^{n}) $RegCascadeGen$(**regression**^{n})|$RegDelegating$(**regression**^{n})
metaReg → $RegDivide$(**regression**^{n}); where n is number of models in ensemble
baseRegression → $Sigmoid$|$SigmoidNorm$|$Exponential$|$Gaussian$|$Linear$| $Polynomial$|$Sine$|$LocalPolynomial$|$NeuralNet$

5.1.5 Metadatabase

When a metadatabase is enabled, the population of general templates can be then seeded from this metadatabase. The probability that a template is selected for seeding the population is inversely proportional to the squared distance of meta data vectors and proportional to a *robust performance* of the template. The robust performance is defined as average rank of template performance on similar data sets. Then, as the algorithm runs, templates consisting of one base algorithm are evaluated on the data set and stored into the metadatabase. Their performance is used as landmarking attribute (Pfahringer et al. 2000) and together with data statistics make up meta-features. The meta-features vector is then compared to other vectors stored in the metadatabase and the most similar records are returned. The records contain a list of best templates which are inserted into the initial population. The fitness of each template is updated during evolutions and when the optimization terminates, winning templates are saved as a new record into the metadatabase or corresponding records are updated with the new templates.

Section 7.5 provides experimental results showing that using the templates from a metadatabase is beneficial for most of the data sets. On the other hand, a metadatabase can lead to templates overfiting and one should avoid seeding the initial population with templates evolved on data that has been already used, as we show later in the experimental part.

5.2 Exploring models produced by templates

The final template is comprehensively tested and the generalization performance of models generated by this template should be the highest among candidate templates. The quality of the selected template can also be observed in the shape and consistency of decision boundaries of models produced from selected template.

As an example, we ran the evolution on the Two Intertwined Spirals data set (Juille and Pollack 1996) (10 min on a standard PC). The template that was finally selected can be written as: *ClassifierCascadeGenProb*{4× *KNN(k=2,vote=true, measure=ManhattanDistance)*}. We used our RapidMiner plugin [Software: Fake game, data mining software (https://fakegame.sourceforge.net/)] to visualize the structure and behavior of the classifier produced when this template was executed. The template contains the *ClassifierCascadeGenProb* ensemble of three 2NN classifiers. In the Cascade Generalization (Gama and Brazdil 2000) ensemble, every model except the first one uses a data set extended by the output of all previous models. In this particular case, the first 2NN classifier is produced on the Spiral data set, the input of the second 2NN classifier is enriched by two outputs of the first classifier (probabilities of membership in one of the two intertwined spirals). The third classifier receives two original 'spiral' inputs plus four output probabilities from the already generated classifiers, etc.

This behavior can be observed in Fig. 4a. As can be seen in the thumbnail images, where the background color should match the color of data points for the perfect classifier, the first KNN algorithm is capable of making a nearly perfect model, except for small regions with absent learning data. The other classifiers specialize in these regions, so the final cascade ensemble classifies the Spiral data even better. Figure 4b shows the decision boundaries of a recently evolved template that outperformed the Cascade generalization of KNN classifiers: *ClassifierModel*{*outputs*× *LocalPolynomialModel*}}. The LocalPolynomialModel was added to our base algorithm recently and it apparently performs better than KNN on this problem. The evolved algorithm works as follows. It builds a lazy model based on the LocalPolynomial regression to model probability of each class (spiral) given the input coordinates. Final output

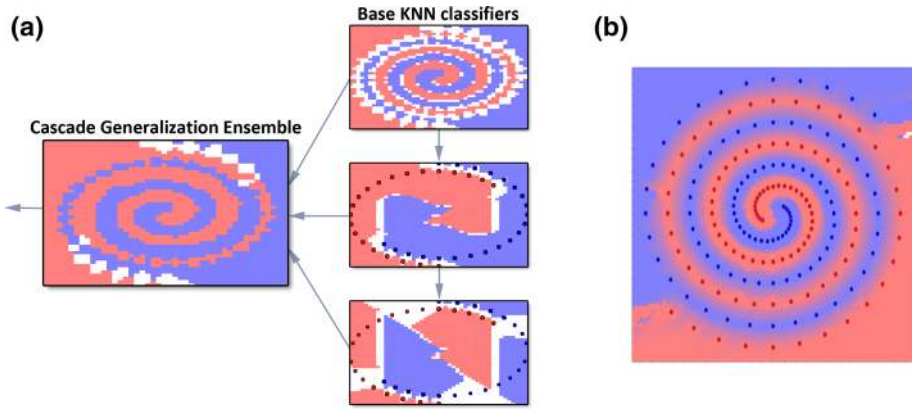

Fig. 4 The template performing cascade generalization ensemble of three 2NN classifiers was discovered by the evolution on the Spiral problem. The thumbnail images show the response of classifiers to the change of their two most relevant inputs

is decided by ClassifierModel choosing higher probability returned by two LocalPolynomial models.

Templates evolved on the Spiral data should also produce good models for similar problems, e.g. for any other complex separation problem in two dimensions. Experiments described in the sections are to reveal the universality of discovered templates.

6 Small data sets for evaluation

To evaluate transfer learning capabilities (Pan and Yang 2010) of meta-learning templates, it is necessary to experiment with a wide range of data sets. First of all, we use small data sets of different complexity.

Table 2 lists the data sets used as well as their size, dimensionality and number of classification classes (outputs). Most of the data sets are taken from the UCI repository (Frank and Asuncion 2010). Other data sets (mostly artificial) are tailored to evaluate data separation capabilities of algorithms for low dimensional problems. The Spirals data set was used in the previous section and was designed as a benchmark for global approximation methods.

Spread is a two-dimensional artificial data set, which was created with an evolutionary algorithm to be unsolvable by the basic classification algorithms available in the RapidMiner. The fitness function was inversely proportional to the performance of the best classifier and the chromosomes contained parameters of a data set generator.

Data sets (*Texture1* and *Texture2*) come from a generator of images for pattern recognition (Texture 2008). Four features were extracted from these images, one using the local binary pattern and the other three with a 5×5 convolution matrix for each color component (rgb) We generated balanced data sets with 250 instances for each class (segment). Texture1 was formed by three segments (750 instances) and Texture2 by ten segments (2500 instances).

Splitting data into learning and testing sets to avoid overfitting is a well known principle. In the process of evolution of templates it is necessary to estimate the quality of templates and to balance well the data used for learning and for evaluation. Also, when some testing data is used to select best performing template, we should not use it for testing any more, because the error estimate might be biased.

Table 2 Data sets are obtained mostly from the UCI repository and are small to medium-sized

data set	Origin	Inputs	Classes	Instances
Balance	UCI	5	3	624
Breast	UCI	9	2	698
Diabetes	UCI	8	2	767
Ecoli	UCI	8	7	335
Glass	UCI	9	7	213
Heart	UCI	13	5	269
Texture1	TSB	4	3	750
Texture2	TSB	4	10	2500
Ionosphere	UCI	34	2	350
Spirals	TIS	2	2	192
Vehicle	UCI	18	8	845
Wine	UCI	13	3	177
Segment	UCI	19	7	2309
Fourier	UCI	76	10	1999
Spread	EVO	2	19	2500

7 Examining properties of templates

First of all, we designed an experiment to verify if hierarchical ensembles can outperform simple ensembles and base model. For this experiment, we selected the Glass data set, because it has quite complex decision boundaries and hierarchical ensembles can therefore reveal their potential.

7.1 Hierarchical ensemble

We compare three configurations of the optimization process with the same time allocation of 2 h. In the first configuration, we restrict the search to trivial templates with base algorithms only (depth 0). The second configuration allows the evolution to consider also ensembles of base algorithms (depth 1). The third configuration extends the search to the second level hierarchies–ensembles of ensembles (depth 2). There was no additional restriction on type of base models or ensembles. We used 20 fold cross validation to increase size of the train sets.

Table 3 shows that the generalization performance of templates increases with depth. It is interesting that all levels were dominated by the same base model (Decision Tree). We can conclude that for this particular problem (Glass data) the hierarchical ensemble represents significant improvement over base models or regular ensembles.

Of course, ensembles and hierarchical ensembles are not always beneficial (for example when problems are linearly separable, there is no need for ensembles, because most of base algorithms are capable of solving the task perfectly alone). That is also the motivation for our optimization procedure to explore trivial templates first and then gradually extend the search space to more complex ensembles and hierarchies.

Table 3 A trivial template (maximal template depth limited to 0) was evolved on the Glass data set for given time (this is equivalent to the selection among base algorithms with optimized parameters), then we run same experiment with maximal depth 1 (simple ensembles of base algorithms allowed) for the same amount of time and so on

Depth limit	Max Acc	Avg Acc	The best meta-learning template found in the search space
0	0.67	0.64	DecisionTree(depth=9,conf=0.04,alt=7)
1	0.76	0.71	ClassifierBagging{40x Decision-Tree(depth=46,conf=0.494,alt=3)}
2	0.78	0.74	ClassifierBoosting{9x ClassifierBoosting{8x DecisionTree(depth=12,conf=0.5,alt=2)}}

Results are averaged from 50 runs for each depth limit. Table shows the best template found for given depth limit and their maximal and average classification accuracies on test data sets

7.2 Template overfitting

The next experiment is to examine the sensitivity of meta-learning templates to data overfitting. A the same time, we will explore the robustness of our approach in terms of generating stable solutions for very similar problems.

The experimental setup is rather complicated so we use Fig. 5 to illustrate it. The Ecoli data set was divided into two folds of equal size (training and testing). The training fold was subsequently divided into learn and validation folds multiple times, with division ratio iterating from 0.1 (10% learning, 90% validation) to 1 (100% learning, 0% validation). Learning sets of increasing size were used to evolve meta-learning templates and to produce models by executing templates on the same data. These models were evaluated on the training set and on the testing set producing the validation and test errors. Whereas the test errors are unbiased estimates of model performances, the validation errors gradually translate to the training errors (possibly biased) as the size of the learn set increases. Note that for 100% learn data fraction, full training set is used to evolve templates and build models so the validation error becomes the training error.

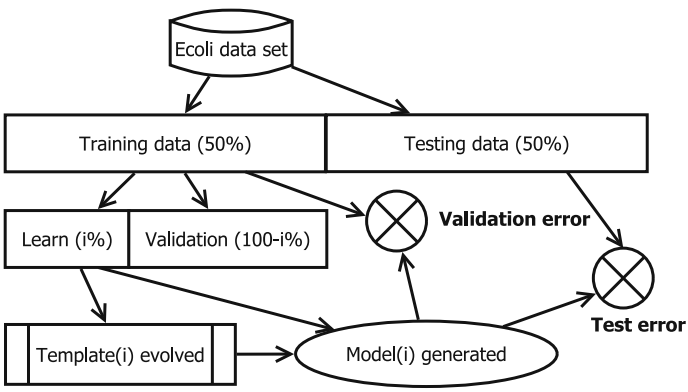

Fig. 5 The workflow evaluating sensitivity to overfitting and stability of solution. Note that validation error is not computed just from the validation set but also from learn set, because the multiple CV is performed

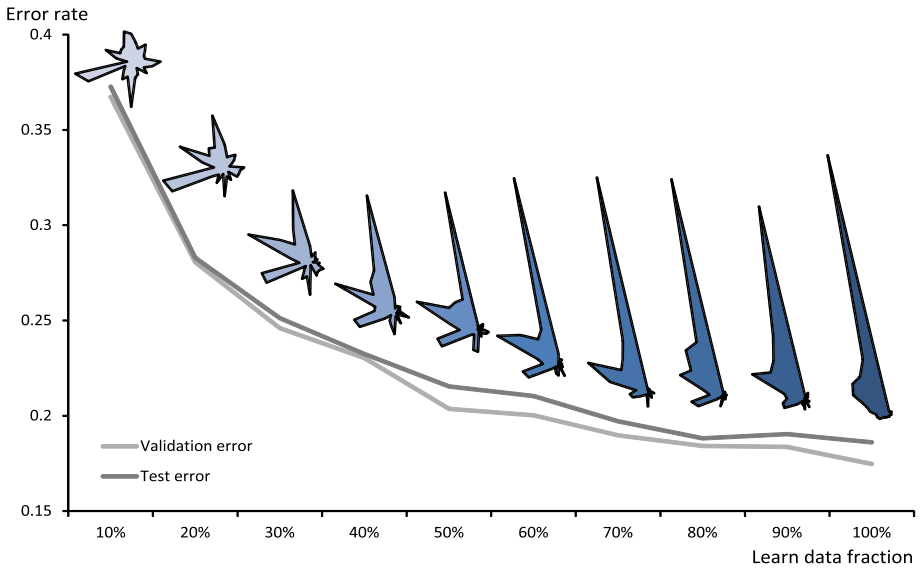

Fig. 6 The difference between test and validation errors is not significant. Glyphs indicate percentages of base algorithms and meta-algorithms in winning templates

We averaged results from 20 repetitions of this setup and plotted the development of errors (see Fig. 6). The level of data overfitting is reasonably low. Even when the same (training) data set is used to evolve the template, build the model and estimate its error, the error is not significantly different from the unbiased estimate computed on independent Testing set. This is mainly due to the fitness function used in the evolution of templates, which favors templates generating models performing well on unseen data. Glyphs summarize numbers of base algorithms and meta-algorithms appearing in evolved templates for each division ratio. For tiny learning data sets (ratio below 0.3) diverse templates were evolved in each repetition, whereas for ratios above 0.8, evolved templates were almost identical.

Note that this behavior is demonstrated on the Ecoli data set, but can be observed also on other data sets. Below, we experiment with diverse portfolio of small data sets to examine which templates are discovered an if they can be reused on other data sets.

7.3 Templates evolved for various data sets

Templates evolved on data sets (see Table 5) were serialized to a text description representing their internal structure. As you can see in the Table 4, for some data sets trivial templates were evolved (for example the KNN algorithm for Heart and Pendigits), for other data sets a regular ensemble performed best (for example the Boosting of Decision Trees for Segment) and hierarchical templates were the best solution for Vehicle or Texture2, Wine or Breast data sets and others. Note that depicted templates are representatives of final templates selected in multiple runs on benchmarking data sets. In each independent (no metadatabase) run of the evolution on a single data set, final template can differ. Diversity of the final templates may be minimal for some data set and significant for other, however they are very similar in terms of functionality and complexity.

The occurrence of individual algorithms can be counted for evolved templates. Almost 40% of solutions were hierarchical templates, the same percentage contained the *Clas-*

Table 4 Templates evolved on individual data sets serialized into text description

Data	Meta-learning templates evolved on benchmarking datasets
Glass	StackingProbabilities{4x KNN(k=2,vote=true,measure=ManhattanDistance)}
Balance	Boosting{57x ClassifierModel{outputs × PolynomialModel(degree=4)}}
Breast	ClassifierModel{outputs × CascadeGenModel{7x CascadeGenModel{5x GaussianModel}}}
Diabetes	SVM(kernel=dot)
Ecoli	ClassifierArbitrating{2x ClassifierBagging{3x SVM(kernel=anova)}}
Heart	KNN(k=15,vote=false,measure=CosineSimilarity)
Texture1	ClassifierArbitrating{4x ClassifierModel{outputs × PolynomialModel(degree=2)}}
Texture2	CascadeGenProb{8x Boosting{2x ClassifierModel{outputs × ExpModel}}}
Ionospher	DecisionTree(maxdepth=20,conf=0.25,alt=10)
Spirals	CascadeGenProb{8x ClassifierArbitrating {4xKNN(k=3,vote=false,measure=MixedEuclideanDistance)}}
Pendigits	KNN(k=3,vote=false,measure=CosineSimilarity)
Vehicle	ClassifierArbitrating{6x ClassifierModel{outputs × DivideModel(mult=6.68){7x PolynomialModel(degree=3)}}}
Wine	CascadeGenProb{9x ClassifierModel{outputs × BoostingRTModel(tr=0.1) {8x GaussianModel}}}
Spambase	ClassifierModel{outputs × CascadeGenModel{9x SigmoidModel}}
Segment	Boosting{17x DecisionTree(maxdepth=24,conf=0.082,alt=0)}
Fourier	NeuralNetClassifier(net=-1x0,epsilon=0.00001,learn=0.3,momentum=0.2)
Spirals+3	KNN(k=3,vote=true,measure=EuclideanDistance)
Spread	CascadeGenProb{5x CascadeGenProb{3x KNN(k=9,vote=true,measure=CosineSimilarity)}}

sifierModel decomposing the classification problem into N regression problems of class probability estimation. It is surprising that regression models are present so often in final classification templates. One possible explanation is that our optimization algorithms for predictive modeling are very efficient and fast. Therefore the evolution can explore many more variants in given time than in case of KNN, Neural nets or other classification algorithms that tend to be slower. It is apparent that ensembles and particularly their hierarchical variants significantly outperform optimized base algorithms for several data sets.

7.4 Similarity and substitutability of templates

The aim of this experiment is to evaluate performance of evolved templates on other data sets to see how universal each template is.

In our contribution (Kordík et al. 2012) we analyzed the similarity of templates in terms of performance on individual data sets. We executed each template on all data sets and measured

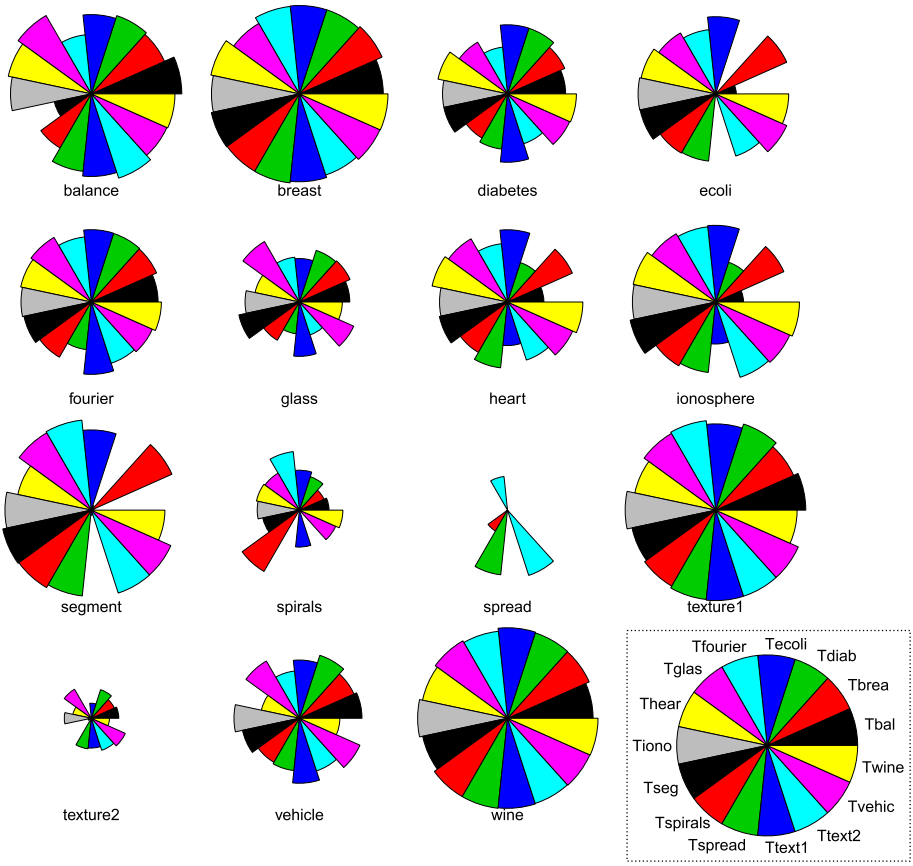

Fig. 7 Performances of meta-learning templates on individual data sets visualized as a starplot matrix. Labels of templates in the legend are derived from data sets used to evolve them. For Balance data, the template evolved on Segment data (Tseg) performs worst whereas the template evolved on Balance data (Tbal) performs best—expected behavior

the performance of generated classifiers. We split each data set randomly into two folds, one is used for learning and the second for evaluating the classifier, then the folds are exchanged. Due to the noise in results, we repeated this procedure 25 times so that each template was evaluated using 25× two fold cross-validation on all data sets.

The results summarized in Fig. 7 show that three data sets (Breast, Wine and Texture1) are very easy to classify, no matter which template (algorithm) is used. The set of evolved templates was slightly different than that listed in this paper (Fig. 4). Although we have added some base algorithms and improved global heuristics of the evolution since the last experiments published in Kordík et al. (2012), the winning templates are quite consistent.

There is a group of four data sets (Ecoli, Heart, Ionosphere, Segment), that can be solved by most of the templates except those based on Polynomial models. These models are trained by the Least squares algorithm (Kordík et al. 2010). For certain data (noisy with binary inputs and overlapping instances) the algorithm fails to deliver a solution due to a non-invertible matrix, the parameters of polynomials are set randomly and the result is poor. There are also two complex data sets (Spirals and Spread) that can be solved almost exclusively by their

templates and one complex noisy data set (Texture2) where only ensemble (or hierarchical ensemble) of algorithms can deliver satisfactory results. You have probably noticed that for a number data sets a template derived from another data set performs better than the one derived for this specific data set. The differences are not significant and they are caused by noise in the process of selection of template and evaluation of template on the other data set.

Based on these experiments we can conclude that hierarchical templates evolved on particular complex problems (data sets) have often capacity to solve other complex problems very well. This is often the case for complex general-purpose templates containing universal algorithms such as neural nets. On the other hand, some problems (Spirals, Spread) require specific algorithms (KNN, Local polynomial regression). Note that in our previous work (Kordík et al. 2012) the template *CascadeGenProb*{9× *ClassifierModel*{*outputs*× *ExpModel*}} was evolved for the Spread problem and it failed to produce good classifiers on the Spiral data set.

When the performance of templates on individual data sets is averaged, we get the "universality" of templates. Templates based on polynomial models are least universal (with 60% average performance). On the other hand, the most universal is the Texture2 template (double stacking of neural nets). With an average performance over 80% on all data sets, the top three templates (Ttexture2, Tspread, Tspirals) contain hierarchical ensembles.

7.5 Evaluation of metadatabase

The motivation of this experiment is to evaluate when the metadatabase should and should not be used. The content and usage of metadatabase is described in Sect. 5.1.5. We run experiments on selected data sets in two configurations (a) metadatabase disabled—initial population generated from minimal form (Base classifiers) and (b) metadatabase enabled.

The positive influence of seeding the initial population with templates from a metadatabase is demonstrated in Fig. 8. The best solution (template) found in the initial population is far better when the metadatabase is used for all tested data sets. The improvement is bigger for complex tasks, such as the Spiral problem, where hierarchical templates have to be discovered and the evolution of such templates from randomly initialized population takes many generations.

We have to note that the metadata and templates of the data sets tested were excluded form the metadatabase with one exception. For Spirals, there was very similar data set (Spirals + 3 irrelevant attributes) contained and that is why the performance went up rapidly after seeding templates from this data set. This observation motivated us to investigate, how much

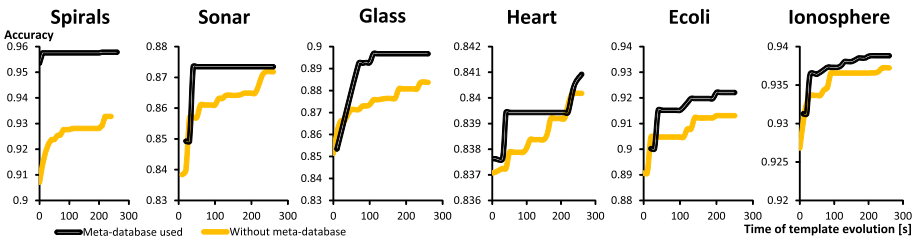

Fig. 8 The improvement in the convergence of the evolution can be observed for all tested data sets when the seeding from the meta database is used. There is a danger of overfitting because the same data sets were already used to evolve very similar data. The bias however should not be high, because of low number of parameters used in the template

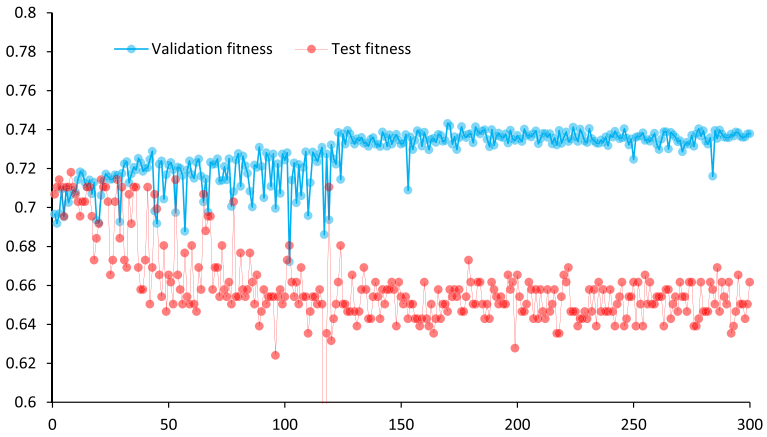

Fig. 9 Templates can overfit the data when the metadatabase is used and the same data set is presented to the system over and over again. The data set here is Borelia (reproduced with permission from Motl (2013))

templates are prone to data overfitting. We stored templates evolved on single data set and then reused it again with the same data. We performed 300 evolutions initialized from the metadatabase (after each evolution the metadatabase was updated when better solution was found).

Apparently, there is a danger of data overfitting when the metadatabase is used. By storing the best template, the data set used to evolve or select the template should not be further used for validation or testing. When the same data set is presented over and over again and each time the best template from the metadatabase is inserted into the initial population of templates, overfitting occurs after dozen of runs as shown in Fig. 9 despite a template does have just a fraction of parameters of a model generated from it. We recommend using metadatabase just in scenarios where we can guarantee enough data to prevent reusing data samples.

8 Benchmarking meta-learning templates on small data

In this section, we evaluate the performance of meta learning templates evolved using the proposed algorithm and compare it to the standard algorithms and ensembles on small data sets described in Sect. 6. We decided to perform experiments in our environment, because we can easily control run-time of algorithms and compare them to RapidMiner classifiers.

The methodology was the following. The training set was used to evolve templates on each data set by means of procedures described above. Then the final template selected for each data set was evaluated on the corresponding test data set by fifty repetitions of ten-fold crossvalidation. The performance of meta-learning templates was compared to the most popular algorithms contained in the RapidMiner software. For each data set, we measured performances of all algorithms with default settings on training data and evaluated the best performing algorithm on the testing set (also 50x tenfold CV).

To improve results of RapidMiner algorithms with default settings, their parameters were optimized on training set with the same evolutionary framework described above (ensembling was disabled). Again, the most successful algorithm with optimized parameters was selected on training data and was evaluated on the testing set.

Table 5 The column Templates lists averaged test performances (classification accuracies in percents) of individual templates, evolved on corresponding training data sets

Data set	Templates	Base-default	Base-optimized	Simple ensembles (LOOCV)
Balance	**99.5**	93.4	97.6	90.7
Breast	**97.7**	96.3	96.3	96.1
Diabetes	79.0	77.1	78.8	77.0
Ecoli	**90.0**	86.3	87.4	86.6
Glass	**83.1**	69.2	74.9	79.0
Heart	86.5	83.9	86.5	82.4
Texture 1	**99.4**	98.5	98.7	98.0
Texture 2	**65.6**	59.8	61.2	57.9
Ionosphere	95.8	92.6	95.8	93.4
Spirals	**95.6**	78.7	80.2	68.2
Pendigits	99.5	99.3	99.4	99.5
Vehicle	**78.5**	75.5	75.9	76.1
Wine	100	98.9	98.9	100
Spambase	93.4	92.0	92.1	**95.3**
Segment	97.6	96.9	97.3	**98.5**
Fourier	83.3	82.4	83.4	83.9
Average	**89.7**	85.5	87.0	85.5

The Base-default column lists performances of the best single classification algorithm with default parameters. The Base-optimized column shows performances of the best single algorithms with parameters optimized (e.g. K=3 for KNN). Templates significantly outperform base classifiers for several data sets (bold)

Results of experiments (Table 5) confirm that classifiers generated by meta-learning templates should be of the same or better quality than base classifiers trained by standard single algorithms from RapidMiner. This is due to the fact that evolution of templates starts from the minimal form and all base algorithms (including RapidMiner base models) are examined in the beginning and survive in the population unless significantly better solutions (e.g. hierarchical ensembles) drive them out.

Although the evolution of the best template itself took about hundred times longer to complete (the approx. time was in minutes, compared to learning process of base models, which is measured in miliseconds), the additional computing time given to the base algorithms will not increase their capacity to generate more precise models. Computing time of optimized base models in RapidMiner is comparable to the time spent for template evolution—especially when grid search is used.

Additionally, we have computed the performance of base models in default settings and selected ensembles using one-leave-out crossvalidation. Classification accuracies in Table 6 shows that most of the results were dominated by meta-learning templates despite the fact that in the one-leave-out crossvalidation setup, models can benefit from bigger training sets than in case of 10fold crossvalidation used to obtain results in the Table 5. On average (we use the geometric average which is more robust to outliers), templates outperformed selected simple ensembles in spite of smaller training sets (10 fold CV versus LOOCV).

In our experiments we were not able to compare templates to all possible ensembles of base algorithms. The number of such combinations is so high that heuristic search is needed—and this feature is not available in the RapidMiner. Also we have many base models that are not

Table 6 One-leave-out validation performance of selected base models and ensembles on benchmarking data sets

Data	Boosting DecTree-50	Boosting Sigmoid-10	Bagging DecTree-50	Bagging Sigmoid-10	Gaussian	Sigmoid	Linear	Polynomial	Exp	DecTree
Balance	74.6	90.7	80.6	87.4	86.1	86.6	84.5	**92.2**	86.6	77.8
Breast	**96.1**	52.4	96.6	64.5	65.5	65.2	65.5	65.5	65.1	95.1
Diabetes	73.2	71.5	75.5	77.0	**77.6**	77.1	77.5	75.8	77.1	69.0
Ecoli	86.3	83.6	**86.6**	87.5	87.5	83.0	84.5	15.5	82.7	78.0
Glass	77.6	64.0	**79.0**	61.7	62.1	63.6	60.7	64.0	56.5	66.8
Heart	80.2	81.5	81.7	82.4	82.8	**83.2**	44.7	48.2	82.0	74.8
Texture1	97.9	98.0	97.3	97.7	96.0	96.1	96.7	**98.8**	96.8	96.5
Texture2	56.0	52.8	**57.9**	49.8	38.3	46.3	37.4	42.0	35.9	48.8
Ionosphere	**93.4**	89.5	93.2	90.0	92.0	88.9	35.9	50.4	87.2	88.9
Spirals	**68.2**	49.0	62.5	45.8	49.5	47.9	46.9	45.8	50.0	65.1
Pendigits	**99.5**	96.7	99.2	91.6	90.0	90.3	86.5	92.4	88.7	95.8
Vehicle	76.1	73.2	75.1	75.1	72.2	73.5	73.5	**76.5**	70.6	71.4
Wine	97.8	98.3	97.8	**100.0**	99.4	97.8	98.9	98.9	98.3	92.7
Spambase	94.7	92.5	**95.3**	92.6	88.8	92.3	88.6	92.2	89.6	91.4
Segment	**98.5**	94.8	98.2	93.1	89.2	90.6	14.3	12.9	90.9	95.8
Fourier	81.9	81.3	**83.9**	81.6	80.3	78.6	78.2	82.2	72.2	66.8

Decision tree was adapted from RapidMiner, other base models are $ClassifierModels$ described in Sect. 3.1

available in other environments. Meta-learning templates are performing so well also due to ability to build classifier from subtrees of regression models.

9 Templates at scale

The recent rise of big data modeling challenges scalability of predictive modeling algorithms and tools. One obvious approach is to reduce dimensionality and numerosity of data (Borovicka et al. 2012). This approach works in most of the cases because big data often includes similar cases that are redundant. However for some data sets, the performance of predictors increase significantly with growing number of instances used for training. For such data, scalable algorithms (Basilico et al. 2011) and tools (Arora et al. 2015; Meng et al. 2016) have been developed.

Most of these approaches are based on a map-reduce technique (Chu et al. 2007).

In this section, we show, that meta-learning can be also used at scale. Our approach is inspired by van Rijn et al. (2015), where classifier selected on sub-samples work reasonably well on larger data sets. We evolve templates on a subset of 3000 randomly selected instances. Then, evolved template can be executed on full data. When we do not have enough time for the meta-learning template evolution, it is also possible to generate the subset just for computing meta-features. Then we can use a best performing template for the data set with most similar meta-features.

For the template execution we split large data into multiple disjoint subsets and then use the map-reduce paradigm to train multiple instances of the template. Prediction is made by reducing (majority voting) of models generated from templates.

This approach is very similar to bagging except that we do not use the bootstrap sampling.

9.1 Experiments

We have conducted experiments to get an insight into the scalability of several machine learning algorithms from h2o as well as our parallel training of templates. Our motivation is to show that proper algorithm selection is important especially for large data sets and can be often done using a fraction of the data set.

We have chosen two public data sets—HIGGS (Baldi et al. 2014) and Airline Delays which is available through H2O (H2O 2015). Those data sets are used for binomial classification of selected output attributes.

We benchmark our paralelized templates to models available in H2O.ai implemented using the map reduce approach. ***Generalized Linear Model*** (Hussami et al. 2015) is using logistic regression to deal with classification problems. ***Naive Bayes*** classifier assumes Independence of input attributes and classifies based on conditional probabilities obtained from training data. ***Deep learning*** (Arora et al. 2015) is a feedforward neural network with various activation functions in neurons. ***Distributed Random Forest*** and ***Gradient Boosted Machine*** (Click et al. 2016) are ensembles based on decision trees. ***H2O Ensemble*** is an ensemble classifier called Super Learner by LeDell (2016).

The following experiments use 1,000,000 randomly selected rows from each data set. Then 50% rows are randomly selected as a test set and the rest is then sampled to subsets of growing size to examine scalability of algorithms. This sampled data is randomly split to training set (80%) and validation set (20%).

At first, we examined scalability of algorithms on the Higgs data set. Figure 10 shows learning time and performance of individual algorithms executed on subsets of growing

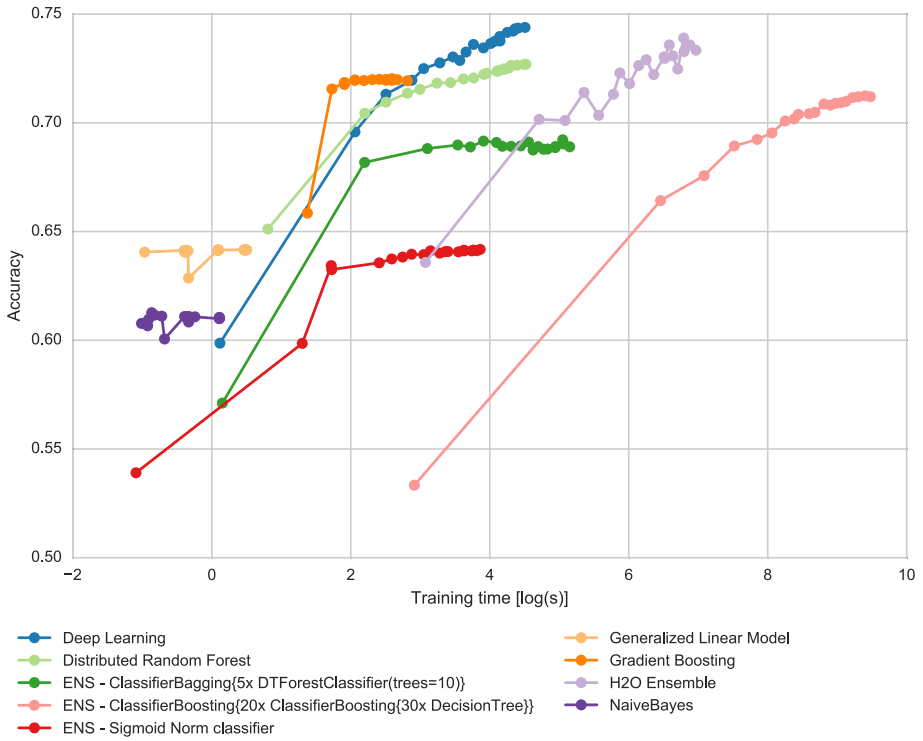

Fig. 10 Comparison of several machine learning algorithms in H2O.ai trained on samples with various sizes from Higgs (Baldi et al. 2014) data set

size. The best performance was achieved by Deep Learning which was also reasonably fast. Gradient Boosting is faster, but it does not have capacity to improve with bigger data subsets. Distributed Random Forest is also reasonably accurate and fast, but it is dominated by Deep Learning on Higgs. Ensembles produced from templates are not very competitive on this data set. Only complex hierarchical ensemble of decision trees is approaching the performance of Distributed Random Forest, but it is much slower. Our implementation is not optimized for H2O.ai.

Looking at the Fig. 11, where arrival delay is predicted on the Airlines data set, results are completely different. Our ensembles are both more accurate and faster. The difference is so big, that we decided to analyze these results further.

We even simplified the prediction task by predicting the departure time without removing the DepTime attribute.

The prediction problem then becomes quite trivial, because you can obtain the target (is departure delayed?) by comparing DepTime and CRSDepTime attribute. It is quite surprising that most of the classifiers are mislead by other attributes and fail to discover this simple relationship.

Figure 12 shows that again our simple ensembles based on Sigmoidal model are able to learn fast and solve the problem even on small subsets. H2O Ensemble and Deep Learning discovered the relationship on 500 thousand instances and their learning time was significantly higher.

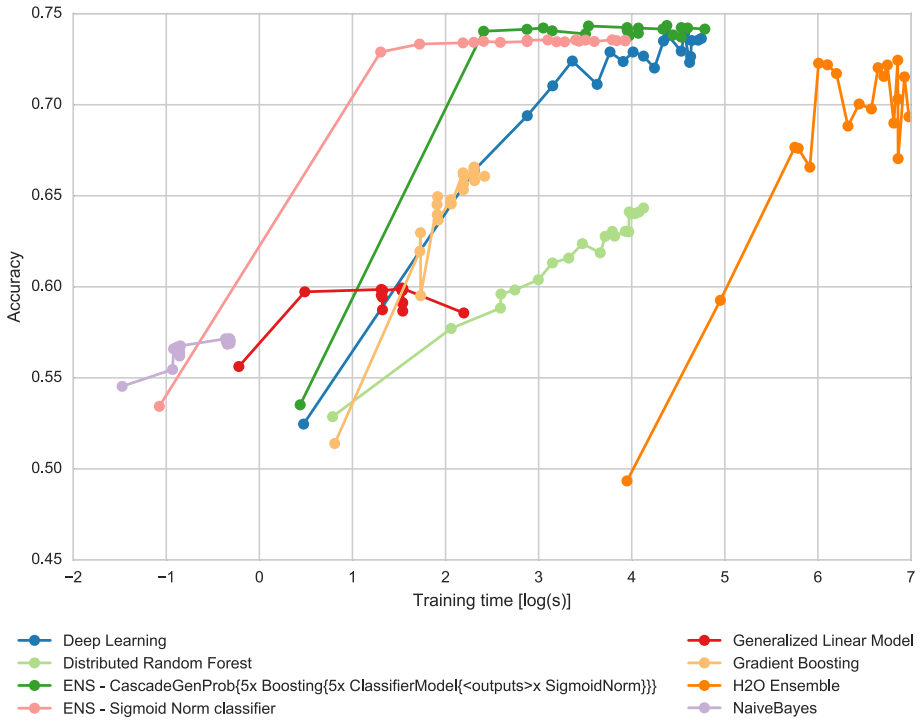

Fig. 11 Predicting IsArrDelayed on Airline data set: comparison of algorithms in H2O.ai trained on subsamples of increasing size

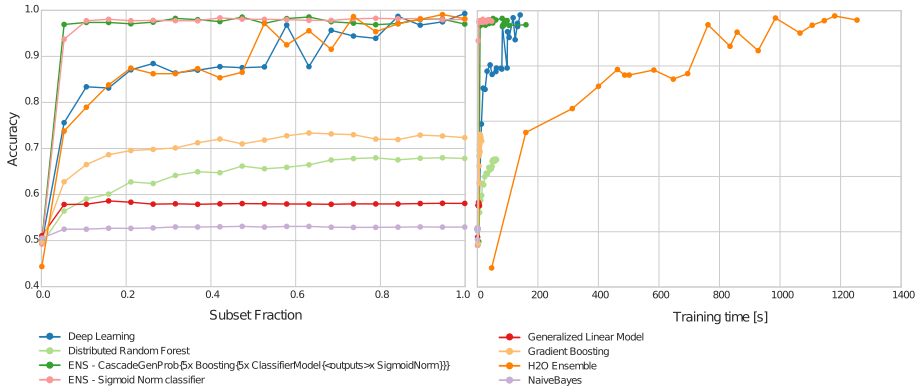

Fig. 12 Predicting IsDepDelayed on Airline data set: comparison of algorithms in H2O.ai trained on subsamples of increasing size. Left subfigure shows that the performance of our templates for small data samples is significantly higher than that of other algorithms. Interesting observation is that deep learning needs almost 100k training instances to match the performance of simple template trained on 10k dataset. Also, when it comes to training times, differences among algorithms are huge (right subfigure). It takes almost 20 min to train H2O Ensemble on this task, whereas Sigmoid template is trained in few seconds

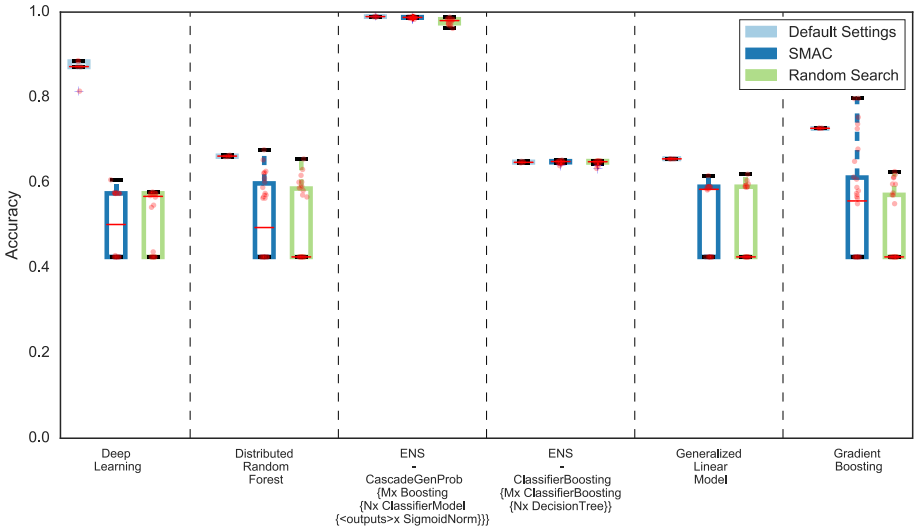

Fig. 13 Hyperparameter search using neither Random Search nor SMAC is beneficial for H2O models. In most of the cases, the performance is worse than for algorithms in default settings

To ensure that the problem is not caused by improper parameter settings, we run optimization of parameters on a subset of 100 thousand instances. The list of parameters and their ranges are available [Software: Algorithmic templates for h2o.ai (https://github.com/kordikp)]. Figure 13 shows that most of the H2O algorithms are very sensitive to improper parameter settings. Deep learning was able to converge in default parameter setting only, our assumption is that parameters are controlled adaptively by default. Similarly, negative impact was observed for Generalized Linear Model. For Gradient Boosting and Distributed Random Forest, optimization discovered better performing configuration, however the difference was not significant. We also optimized number of models in our hierarchical ensembles but apparently it had almost no effect on performance. The Decision Tree based ensemble was unable to solve the task in any configuration which is consistent with poor performance of DT based ensembles from H2O. On the other hand the Sigmoid based ensemble was able to discover the relationship even with minimal number of models in the ensemble which is consistent with previous experiments. From boxplots and distribution of individual results (red dots) the Bayesian Optimization (SMAC) method outperformed the Random search.

Plots of class probabilities and decision boundaries helped us to reveal the reason of poor performance of decision tree based ensembles. Figure 14 shows that successful classifiers (ensemble of sigmoid models, Deep Learning) were able to identify simple relation of two input attributes to departure delay prediction. The relationship (decision boundary) is hard for decision trees to model with their orthogonal decisions. It is also impossible to solve for Naive Bayes classifier assuming independence of input attributes.

Apparently, we were able to discover very efficient template for this trivial problem. We believe that our approach can contribute to evolve (discover) templates for diverse data sets and predictive tasks. Building library of algorithmic templates can improve capacity of predictive modeling systems to solve diverse tasks efficiently.

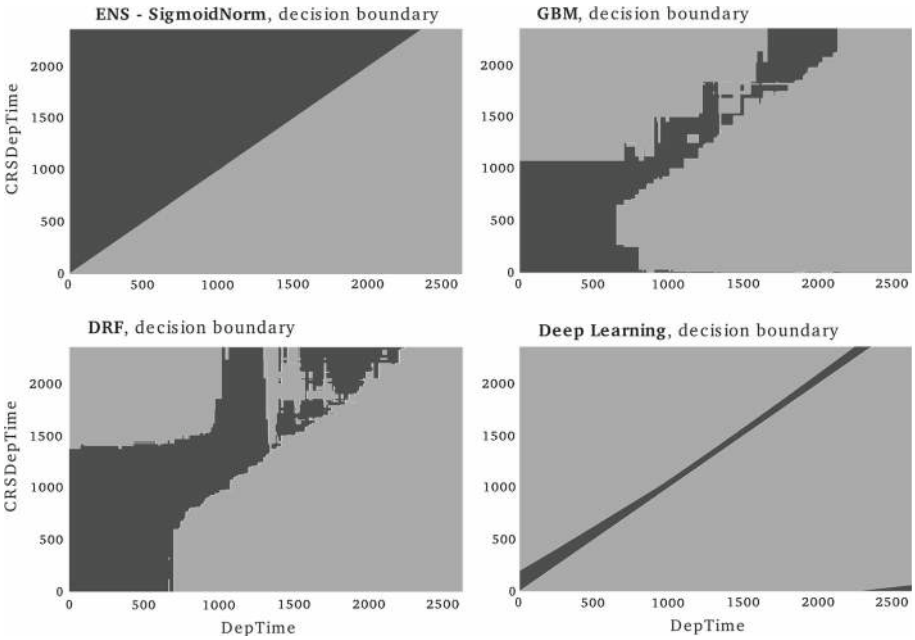

Fig. 14 Decision boundaries of algorithms on problem of predicting aircraft departure delay. Simple ensemble of sigmoid classifiers was able to generalize the relationship well, whereas decision tree based ensembles overfitted the data. Deep Learning discovered the relationship only on large data samples. Note that plots are showing the behaviour of classifiers just in two dimensional plane of the multidimensional input space. Attributes DepTime and CRSDepTime were however the most important dimensions

10 Conclusions

In this article, we propose to optimize topology and parameters of algorithmic ensembles by an evolutionary algorithm. We show that useful algorithms can be discovered on small data sub-samples and later applied to large data sets.

We use meta-learning templates to describe set of algorithmic ensembles and examine their performance on several benchmarking problems.

The generalization accuracy of classifiers generated using these templates are capable of outperforming classifiers produced by the most popular data mining algorithms.

We found out that templates are prone to data overfitting in spite of very low number of their parameters. One needs to be aware of this issue especially when a metadatabase is employed. Data samples that were used to select best template in previous runs cannot be reused for unbiased estimate of the generalization performance. Metadatabase can be however very useful in speeding up the convergence when enough data is available for independent model validation.

We show how templates can be scaled up for large data sets modeling using the map-reduce approach. Benchmarks revealed that our approach is able to produce algorithms competitive with state of the art approaches for large scale predictive modeling. Ensembles of simple regression models can outperform popular algorithms in both generalization ability and scalability as demonstrated on the Airlines data set.

Acknowledgements This research was partially supported by the Modern data-mining methods for advanced extraction of information from data ($SGS17/210/OHK3/3T/18$) grant of the Czech Technical University in Prague. Thanks to reviewers for their valuable time and comprehensive feedback.

References

Alpaydin, E., & Kaynak, C. (1998). Cascading classifiers. *Kybernetika*, *34*, 369–374.

Analoui, M., Bidgoli, B. M., & Rezvani, M. H. (2007). Hierarchical classifier combination and its application in networks intrusion detection. In *International conference on data mining workshops*, pp. 533–538.

Arora, A., Candel, A., Lanford, J., LeDell, E., & Parmar, V. (2015). Deep learning with h2o.

Baldi, P., Sadowski, P., & Whiteson, D. (2014). Searching for exotic particles in high-energy physics with deep learning. *Nature Communications* 5. https://doi.org/10.1038/ncomms5308

Bao, X., Bergman, L., & Thompson, R. (2009). Stacking recommendation engines with additional meta-features. In *RecSys '09: Proceedings of the third ACM conference on recommender systems* (pp. 109–116). New York: ACM.

Basilico, J. D., Munson, M. A., Kolda, T. G., Dixon, K. R., & Kegelmeyer, W. P. (2011). Comet: A recipe for learning and using large ensembles on massive data. In *2011 IEEE 11th international conference on data mining* (pp. 41–50).

Bennett, J., Lanning, S., & Netflix, N. (2007). The netflix prize. In *In KDD cup and workshop in conjunction with KDD*.

Bensusan, H., & Kalousis, A. (2001). Estimating the predictive accuracy of a classifier. In *Proceedings of the 12th European conference on machine learning*. Springer.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems (NIPS)* pp. 2546–2554.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*, 281–305.

Bonissone, P. P. (2012). Lazy meta-learning: Creating customized model ensembles on demand. In *IEEE world congress on computational intelligence* (pp. 1–23). Springer.

Borovicka, T., Jirina, M, Jr., Kordik, P., & Jirina, M. (2012). Selecting representative data sets. In *Advances in data mining knowledge discovery and applications*. Intech.

Botia, J. A., Gomez-Skarmeta, A. F., Valdes, M., & Padilla, A.: METALA (2001). A meta-learning architecture. In: *Proceedings of the international conference, seventh fuzzy days on computational intelligence, theory and applications*.

Brazdil, P., Giraud-Carrier, C., Soares, C., & Vilalta, R. (2009). *Metalearning: Applications to data mining. Cognitive technologies*. New York: Springer.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Browne, M. W. (2000). Cross-validation methods. *Journal of Mathematical Psychology*, *44*(1), 108–132.

Brown, G., Wyatt, J., & Tino, P. (2006). Managing diversity in regression ensembles. *Journal of Machine Learning Research*, *6*, 1621–1650.

Buk, Z., Koutník, J., Šnorek, M. (2009). Neat in hyperneat substituted with genetic programming. In *International conference on adaptive and natural computing algorithms* (pp. 243–252). Springer.

Caruana, R., Niculescu-Mizil, A., Crew, G., & Ksikes, A. (2004). Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference machine learning* (ICML 2004), Banff, Alberta.

Chan, P. K., & Stolfo, S. J. (1997). On the accuracy of meta-learning for scalable data mining. *Journal of Intelligent Information Systems*, *8*(1), 5–28.

Chu, C., Kim, S. K., Lin, Y. A., Yu, Y., Bradski, G., Ng, A. Y., et al. (2007). Map-reduce for machine learning on multicore. *Advances in Neural Information Processing Systems*, *19*, 281.

Click, C., Malohlava, M., Candel, A., Roark, H., & Parmar, V. (2016). Gradient boosting machine with h2o.

Coope, I. D., & Price, C. J. (2001). On the convergence of grid-based methods for unconstrained optimization. *SIAM Journal on Optimization*, *11*(4), 859–869.

Costa, E. P., Lorena, A. C., Carvalho, A. C., & Freitas, A. A. (2008). Top-down hierarchical ensembles of classifiers for predicting g-protein-coupled-receptor functions. In *Proceedings of the 3rd Brazilian symposium on bioinformatics*BSB '08 (pp. 35–46). Springer: Berlin.

Duffy, N., & Helmbold, D. (1999). A geometric approach to leveraging weak learners. In *European conference on computational learning theory* (pp. 18–33). Springer

Fake Game, data mining software. http://fakegame.sourceforge.net/.

Fernando, C., Banarse, D., Reynolds, M., Besse, F., Pfau, D., Jaderberg, M., Lanctot, M., & Wierstra, D.(2016). Convolution by evolution: Differentiable pattern producing networks. In: *Proceedings of the 2016 on genetic and evolutionary computation conference* (pp. 109–116). ACM

Ferri, C., Flach, P., & Hernández-Orallo, J. (2004). Delegating classifiers. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, Vol. 37. New York, NY: ACM

Frank, A., & Asuncion, A. (2010). UCI machine learning repository.

Friedman, J. H. (2000). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, *29*, 1189–1232.

Gama, J., & Brazdil, P. (2000). Cascade generalization. *Machine Learning*, *41*(3), 315–343.

Grabczewski, K., & Jankowski, N. (2007). Versatile and efficient meta-learning architecture: Knowledge representation and management in computational intelligence. In *IEEE symposium on computational intelligence and data mining*, 2007. CIDM 2007 (pp. 51–58).

Grefenstette, J. J., & Ramsey, C. L. (2014). An approach to anytime learning. In *Proceedings of the ninth international conference machine learning* (pp. 189–195).

Grubb, A. (2014). *Anytime prediction: Efficient ensemble methods for any computational budget*. DTIC Document: Technical report.

H2O.ai: H2O: Scalable Machine Learning. (2015).

Hengpraprohm, S., & Chongstitvatana, P. (2008). A genetic programming ensemble approach to cancer microarray data classification. In *The 3rd intetnational conference on innovative computing information and control*.

Hoch, T. (2015). An ensemble learning approach for the kaggle taxi travel time prediction challenge. ECML-PKDD-DCs.

Holeňa, M., Linke, D., & Steinfeldt, N. (2009). Boosted neural networks in evolutionary computation. In *Neural information processing. Lecture notes in computer science* (Vol. 5864, pp. 131–140). Berlin: Springer.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. Ann Arbor: Michigan Press.

Hussami, N., Kraljevic, T., Lanford, J., Nykodym, T., Rao, A., & Wang, A. (2015). Generalized linear modeling with h2o.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Lecture notes in computer science* pp. 507–523.

Jankowski, N. (2013). Meta-learning and new ways in model construction for classification problems. *Journal of Network and Information Security*, *4*(4), 275–284.

Juille, H., & Pollack, J. B. (1996). *Co-evolving Intertwined Spirals* (pp. 461–467). Cambridge: MIT Press.

Kaynak, C., & Alpaydin, E. (2000). Multistage cascading of multiple classifiers: One man's noise is another man's data. In *ICML '00: Proceedings of the seventeenth international conference on machine learning* (pp. 455–462). San Francisco: Morgan Kaufmann Publishers Inc.

Kietz, J.U., Serban, F., Bernstein, A., & Fischer, S. (2012). Designing kdd-workflows via htn-planning. In *Proceedings of the 20th European conference on artificial intelligence* (pp. 1011–1012). IOS Press.

Kordík, P. (2009). Game–hybrid self-organizing modeling system based on gmdh. In *Hybrid self-organizing modeling systems* (pp. 233–280). Springer.

Kordík, P. (2006). Fully automated knowledge extraction using group of adaptive models evolution. PhD thesis, Czech Technical University in Prague, FEE, Department of Computer Science and Computers, FEE, CTU Prague, Czech Republic.

Kordík, P., & Černý, J. (2011). Self-organization of supervised models. In N. Jankowski, W. Duch, & K. Grabczewski (Eds.), *Meta-learning in computational intelligence. Studies in computational intelligence* (Vol. 358, pp. 179–223). Berlin: Springer.

Kordík, P., & Černý, J. (2012). On performance of meta-learning templates on different datasets. In: IJCNN, IEEE, pp. 1–7.

Kordík, P., & Černý, J. (2014). Building predictive models in two stages with meta-learning templates optimized by genetic programming. In *IEEE symposium on computational intelligence in ensemble learning (CIEL)*, pp. 1–8.

Kordík, P., Koutník, J., Drchal, J., Kovářík, O., Čepek, M., & Šnorek, M. (2010). 2010 special issue: Meta-learning approach to neural network optimization. *Neural Networks*, *23*(4), 568–582.

Koza, J. R. (2000). Genetic programming. *IEEE Intelligent Systems*, *14*(4), 135–84.

Kuncheva, L. (2004). *Combining pattern classifiers: Methods and algorithms*. New York: Wiley.

LeDell, E. (2016). Scalable super learning. Handbook of Big Data 339.

Leite, R., Brazdil, P., & Vanschoren, J. (2012). Selecting classification algorithms with active testing. In *International workshop on machine learning and data mining in pattern recognition* (pp. 117–131). Springer.

Leite, R., & Brazdil, P. (2010). Active testing strategy to predict the best classification algorithm via sampling and metalearning. *ECA*, *I*, 309–314.
Lemke, C., & Gabrys, B. (2010). Meta-learning for time series forecasting and forecast combination. *Neurocomputing*, *73*(10), 2006–2016.
Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2016). Efficient hyperparameter optimization and infinitely many armed bandits. arXiv preprint.
Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, *11*(2), 431–441.
Meng, X., Bradley, J., Yuvaz, B., Sparks, E., Venkataraman, S., Liu, D., et al. (2016). Mllib: Machine learning in apache spark. *JMLR*, *17*(34), 1–7.
Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Navruzyan, A., Duffy, N., & Hodjat, B. (2017). Evolving deep neural networks. arXiv preprint arXiv:1703.00548.
Moore, A. W., & Lee, M. S. (1994). Efficient algorithms for minimizing cross validation error. In *ICML*, pp. 190–198.
Motl, J. (2013). Supporting the diagnosis of borreliosis by machine learning methods. Master's thesis, CTU in Prague.
Mueller, J. A., Ivachnenko, A., & Lemke, F. (1998). Gmdh algorithms for complex systems modelling. *Mathematical and Computer Modelling of Dynamical Systems*, *4*(4), 275–316.
Nguyen, P., Hilario, M., & Kalousis, A. (2014). Using meta-mining to support data mining workflow planning and optimization. *Journal of Artificial Intelligence Research*, *51*, 605–644.
Ortega, J., Koppel, M., & Argamon, S. (2001). Arbitrating among competing classifiers using learned referees. *Knowledge and Information Systems*, *3*(4), 470–490.
Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, *22*(10), 1345–1359.
Pfahringer, B., Bensusan, H., & Giraud-Carrier, C. (2000). Meta-learning by landmarking various learning algorithms. In *Proceedings of the 17th international conference on machine learning*.
Puurula, A., Read, J., & Bifet, A. (2014). Kaggle lshtc4 winning solution. arXiv preprint arXiv:1405.0546
RapidMiner, data mining software. https://rapid-i.com/.
Rawal, A., & Miikkulainen, R. (2016). Evolving deep lstm-based memory networks using an information maximization objective. In *Proceedings of the 2016 on genetic and evolutionary computation conference* (pp. 501–508). ACM
Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Le, Q., & Kurakin, A. (2017). Large-scale evolution of image classifiers. arXiv preprint arXiv:1703.01041.
Rosca, J. P. (1997). Analysis of complexity drift in genetic programming. *Genetic Programming* pp. 286–294.
Ruta, D., & Gabrys, B. (2002). A theoretical analysis of the limits of majority voting errors for multiple classifier systems. *Pattern Analysis and Applications*, *5*(4), 333–350.
Ruta, D., & Gabrys, B. (2005). Classifier selection for majority voting. *Information Fusion*, *6*(1), 63–81.
Salvador, M. M., Budka, M., & Gabrys, B. (2016a). Adapting multicomponent predictive systems using hybrid adaptation strategies with auto-weka in process industry. In *International conference on machine learning. AutoML workshop*.
Salvador, M. M., Budka, M., & Gabrys, B. (2016b). Towards automatic composition of multicomponent predictive systems. In *International conference on hybrid artificial intelligence systems* (pp. 27–39). Springer.
Salvador, M. M., Budka, M., & Gabrys, B. (2016c). Automatic composition and optimisation of multicomponent predictive systems. arXiv preprint arXiv:1612.08789
Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, *5*(2), 197–227.
Shanno, D. F. (1970). Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, *24*(111), 647–656.
Smithson, S. C., Yang, G., Gross, W. J., & Meyer, B. H. (2016). Neural networks designing neural networks: Multi-objective hyper-parameter optimization. arXiv preprint arXiv:1611.02120.
Software: Algorithmic templates for h2o.ai (https://github.com/kordikp).
Software: Fake game, data mining software (https://fakegame.sourceforge.net/).
Software: Rapid miner, data mining (https://rapid-i.com/).
Sonnenburg, S., Franc, V., Yom-Tov, E., & Sebag, M. (2008). Pascal large scale learning challenge. In *25th international conference on machine learning (ICML2008) workshop* (Vol. 10, pp. 1937–1953). https://largescale.first.fraunhofer.de.J.Mach.Learn.Res.
Stanley, K. O., & Miikkulainen, R. (2001). Evolving neural networks through augmenting topologies. Technical report, University of Texas at Austin, Austin, TX, USA.
Stroud, J., Enverga, I., Silverstein, T., Song, B., & Rogers, T. (2012). *Ensemble learning and the heritage health prize (iCAMP 2012)*. University of California, Irvine.

Sun, Q., Pfahringer, B., & Mayo, M. (2013). Towards a framework for designing full model selection and optimization systems. In *Proceedings of 11th international workshop multiple classifier systems*, MCS 2013, Nanjing, China, pp. 259–270.

Sung, Y. H., kyun Kim, T., & Kee, S. C. (2009). Hierarchical combination of face/non-face classifiers based on gabor wavelet and support vector machines.

Sun, Q., & Pfahringer, B. (2013). Pairwise meta-rules for better meta-learning-based algorithm ranking. *Machine Learning*, *93*(1), 141–161.

Sutherland, A., Henery, R., Molina, R., Taylor, C. C., & King, R. (1993). *StatLog: Comparison of classification algorithms on large real-world problems*. Berlin: Springer.

Texture Segmentation Benchmark. In *Proceedings of the 19th international conference on pattern recognition, ICPR 2008, Los Alamitos, IEEE computer society* (December 2008).

Thornton, C., Hutter, F., & Hoos, H. H. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 847–855.

Töscher, A., & Jahrer, M. (2009). *The bigchaos solution to the netflix grand prize*. Technical report, commendo research & consulting.

Tsakonas, A., & Gabrys, B. (2012). Gradient: Grammar-driven genetic programming framework for building multi-component, hierarchical predictive systems. *Expert Systems with Applications*, *39*(18), 13253–13266.

Ulf Johansson, Tuve Lfstrm, R. K., & Niklasson, L. (2006). Building neural network ensembles using genetic programming. In *International joint conference on neural networks*.

van Rijn, J. N., Abdulrahman, S. M., Brazdil, P., & Vanschoren, J. (2015). Fast algorithm selection using learning curves. In *International symposium on intelligent data analysis*, (pp. 298–309). Springer.

Vanschoren, J. (2010). Understanding machine learning performance with experiment databases. Ph.D. thesis, Katholieke Universiteit Leuven.

Vanschoren, J., Blockeel, H., Pfahringer, B., & Holmes, G. (2012). Experiment databases. *Machine Learning*, *87*(2), 127–158.

Vilalta, R., & Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artificial Intelligence Review*, *18*(2), 7795.

Whigham, P. A., et al. (1995). Grammatically-based genetic programming. In *Proceedings of the workshop on genetic programming: From theory to real-world applications*, Vol. 16, pp 33–41.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, *5*, 241–259.

Woods, K., Kegelmeyer, W., & Bowyer, K. (1997). Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*, 405–410.

Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578.