A Note on Generalization Loss When Evolving Adaptive Pattern Recognition Systems

Christian Igel, *Senior Member, IEEE*

***Abstract*—Evolutionary computing provides powerful methods for designing pattern recognition systems. This design process is typically based on finite sample data and therefore bears the risk of overfitting. This paper aims at raising the awareness of various types of overfitting and at providing guidelines for how to deal with them. We restrict our considerations to the predominant scenario in which fitness computations are based on point estimates. Three different sources of losing generalization performance when evolving learning machines, namely overfitting to training, test, and final selection data, are identified, discussed, and experimentally demonstrated. The importance of a pristine hold-out data set for the selection of the final result from the evolved candidates is highlighted. It is shown that it may be beneficial to restrict this last selection process to a subset of the evolved candidates.**

***Index Terms*—Evolutionary learning, machine learning, model selection, overfitting, pattern recognition.**

I. Introduction

EVOLUTIONARY algorithms are frequently and successfully used to design adaptive pattern recognition systems, in particular, neural networks [1]–[3] and, more recently, kernel-based learning machines [4]–[7]. The predominant design goal is to create learning systems that generalize well, that is, accurately recognize patterns not used in the design process.

In this paper, reasons for the loss of generalization performance when evolving adaptive pattern recognition systems due to overfitting are discussed. These considerations help improve the design of adaptive systems and reveal that many estimates of the generalization performance of evolved systems reported in the literature are overoptimistic and may even be misleading.

Overfitting occurs when a hypothesis faithfully reflects aspects of the data used in the design process to such an extent that idiosyncrasies of these data, rather than merely of the underlying distribution, shape the hypothesis [8]. Three sources of overfitting are of special interest when evolving learning systems:

1) overfitting to training data;
2) overfitting to evaluation data;
3) overfitting to final selection data.

Manuscript received October 14, 2011; revised January 30, 2012 and April 13, 2012; accepted April 16, 2012. Date of publication May 2, 2012; date of current version May 24, 2013. This work was supported by the European Commission through Project AKMI (PCIG10-GA-2011-303655).

The author is with the Department of Computer Science, University of Copenhagen, Copenhagen 2100, Denmark (e-mail: igel@diku.dk).

Digital Object Identifier 10.1109/TEVC.2012.2197214

The first is well known and will only be discussed briefly, for completeness. It describes the effect of getting a bad estimate of system performance if the data for training are the same as for performance evaluation. The second refers to overfitting to data used in the selection process, for example, a fixed hold-out data set on which the fitness is based. Because of this second type of overfitting, it is argued that it is beneficial to consider an additional data set for the final hypothesis selection, that is, to be used to pick the final result from the evolved learning machines. The third type of overfitting refers to overfitting to this data set and ideas for reducing this type of overfitting will be presented.

Section II introduces the notation and some basic concepts. Section III will discuss three types of overfitting and recommendations for reducing their effects. Then, experiments will be presented to demonstrate the different sources of overfitting and to show that the proposed countermeasures are indeed helpful.

II. Background

First, a formal definition of the supervised learning problem will be given. We will focus on classification, but these considerations apply, as well, to other learning tasks, such as ranking, regression, and density estimation. Then, a canonical evolutionary algorithm for evolving learning machines will be sketched.

A. Learning Problem

A supervised pattern classification problem can be described by an input space X, a set of classes Y, and a joint probability distribution P on $X \times Y$. The goal of a pattern recognition algorithm is to generate a hypothesis $h : X \to Y$ that minimizes the generalization error

$$\mathrm{er}_P(h) = \int L(y, h(x)) \mathrm{d}P(x, y) \tag{1}$$

given some loss function $L : Y \times Y \to \mathbb{R}$ with $\forall y \in Y : L(y, y) = 0$. For a pattern x belonging to class y the loss $L(y, h(x))$ quantifies the error when the hypothesis predicts $h(x)$ given x. In the following, we assume the canonical 0-1-loss, which is zero if $h(x) = y$ and 1 otherwise.

Learning is driven by a finite sample of training data $S = \{(x_1, y_1), \ldots, (x_\ell, y_\ell)\}$ drawn i.i.d. according to P. The empirical error of a hypothesis h on data set S is given by

$$\mathrm{er}_S(h) = \frac{1}{\ell} \sum_{i=1}^{\ell} L(y_i, h(x_i)). \tag{2}$$

A learning machine A, given sample data, generates a hypothesis. Formally, $A : \{((x_1, y_1), \ldots, (x_\ell, y_\ell)) \mid 1 \leq i \leq \ell < \infty \text{ and } x_i \in X, y_i \in Y\} \to \mathcal{H}$, where $\mathcal{H}$ is a class of hypotheses. We refer to this data-driven generation of a hypothesis as model building or learning.

In the following sections, we will draw a basic bound on generalization performance for binary classification to illustrate some of the arguments. Let $\mathcal{H}$ be a finite set of hypotheses mapping elements of X to $Y = \{-1, 1\}$. Then for any $\ell \in \mathbb{N}^+$, any probability distribution P on $X \times \{-1, 1\}$, an $S = \{(x_1, y_1), \ldots, (x_\ell, y_\ell)\}$ drawn i.i.d. according to P, and any $\epsilon \in \mathbb{R}^+$, we have

$$\Pr\{\max_{h \in \mathcal{H}} |\operatorname{er}_P(h) - \operatorname{er}_S(h)| \geq \epsilon\} \leq 2|\mathcal{H}|e^{-2\epsilon^2\ell}. \quad (3)$$

A proof can be found, for example, in [9]. The more data there is (i.e., the larger ℓ), the more concentrated is the estimate. But also the fewer hypothesis to choose from (i.e., the smaller $|\mathcal{H}|$), the more concentrated is the estimate.

In the following, our primary interest is finding a good hypothesis h minimizing $\operatorname{er}_P(h)$. However, to select candidates for solutions we need reliable estimates of their performance, that is, we want $|\operatorname{er}_P(h) - \operatorname{er}_S(h)|$ to be small in order to make proper decisions about whether to select or discard h. Note that a small / large $\operatorname{er}_P(h)$ does not imply a small / large $|\operatorname{er}_P(h) - \operatorname{er}_S(h)|$ nor vice versa.

B. Evolving Learning Machines

When evolving adaptive pattern recognition systems the goal is to find an appropriate learning machine for a given problem or a class of problems. The solution is chosen from a predefined set of learning machines. Each element a of the search space $\mathcal{G}$ usually encodes one of these machines, which is denoted by A_a.

Many evolutionary algorithms for the design of adaptive systems more or less fit into the canonical scheme sketched in Algorithm 1. Of course, they fundamentally differ in what kinds of learning machines are considered and what is encoded in the genome of individuals. For example, the genotypes may encode hyperparameters of the learning algorithm (e.g., regularization parameters or learning rates), the topology of a neural network or graphical model, or may control the data preprocessing (e.g., in evolutionary feature selection).

In Algorithm 1, S denotes the data set available for the whole design process. Typically, the algorithm has no access to the underlying distribution P and cannot obtain new samples. In each generation g, subsets of these data S_{train} and S_{evolve} are used to generate and evaluate hypotheses, respectively. In this canonical algorithm, it is possible but not necessary that the current parents are reevaluated and considered in the selection process. The selection of the final result a^* can be based on additional evaluations or can correspond to simply choosing the fittest individual found so far.

Algorithm 1 is stated to illustrate where the (not necessarily disjunct) data sets $S_{\text{train}}, S_{\text{evolve}}, S_{\text{final}} \subseteq S$ are used in an evolutionary design process. However, the following considerations also hold for algorithms that only fit roughly into this scheme.

Algorithm 1 only captures evolutionary systems in which the fitness value of an individual is based on a point estimate.

Algorithm 1: Simple canonical evolutionary algorithm for the design of an adaptive system. The parent population and offspring population at iteration g are denoted by $P^{(g)}$ and $O^{(g)}$ with size μ and λ, respectively. The search space $\mathcal{G}$ contains descriptions of learning algorithms that generate hypotheses given sample data S

Data: sample data S
$g = 0$, initialize $P^{(g)} \in \mathcal{G}^\mu$
repeat
  create $O^{(g)} = \left(a_i^{(g+1)} \mid i = 1, \ldots, \lambda\right) \in \mathcal{G}^\lambda$ from $P^{(g)}$
  for $a \in O^{(g)} \cup P^{(g)}$ **do**
    build model using $S_{\text{train}} \subseteq S$ and compute fitness $\Phi(a)$ using $S_{\text{evolve}} \subseteq S$
  end
  select $P^{(g+1)} \in \mathcal{G}^\mu$ from $O^{(g)}$ and $P^{(g)}$ based on fitness
  $g \leftarrow g + 1$
until *some stopping criterion is met*
select final $a^* \in \left\{a_i^{(g)} \mid i = 1, \ldots, \lambda \wedge g = 1, \ldots\right\}$ based on $S_{\text{final}} \subseteq S$
Output: a^*

It is assumed that each individual corresponds to a single model, assigned a scalar performance value. Therefore, the formal framework does not cover algorithms in which each individual represents a probability distribution over models or rather model parameters, as in full Bayesian approaches, or interval estimates of these parameters (e.g., confidence intervals). It does, however, include probabilistic approaches in which probabilistic inference is used by the learning machine to infer a single hypothesis. In particular, the learning machine could map to the hypothesis corresponding to a maximum likelihood estimate or a maximum *a posteriori* estimate of the model parameters.

III. Three Ways of Overfitting

Three particular sources of overfitting when evolving adaptive pattern recognition systems will be identified and recommendations will be given for how to deal with them.

A. Overfitting to Training Data

The first source of overfitting is the well-known fact that the resubstitution method, in which the data for training the model are the same as for estimating its performance, yields a bad estimate of the performance of the system when a complex predictor is built based on too few examples.

Let us consider evolutionary optimization of neural networks as an example. If $S_{\text{train}} = S_{\text{evolve}}$, that is, if the fitness of each neural network is the error on the training set, then the evolutionary process will clearly overfit to $S_{\text{train}} = S_{\text{evolve}}$. For completeness, let us state this well-known problem and its remedies.

Problem 1 (Overfitting to Training Data): Given a learning algorithm choosing from a rich class of hypotheses, the training error of the produced hypothesis is an

over-optimistically biased estimate of its generalization performance.

Now, what does richness (or complexity or capacity) of a hypothesis class mean in this context? It can be measured in different ways, for example, in terms of the VC dimension [10] or the Rademacher complexity [11], or, for a finite class, simply by its cardinality. These measures of the richness of a hypothesis class $\mathcal{H}$ have an intuitive interpretation in terms of the possibility of finding a function in $\mathcal{H}$ fitting arbitrarily labeled data. Furthermore, they have the desirable property that they allow bounding the maximum difference between the true and empirical loss for functions in $\mathcal{H}$. An example of such a bound is (3), which is only applicable to finite $\mathcal{H}$ and uses the cardinality $|\mathcal{H}|$ as a measure of complexity.

Recommendation 1: Use as much data as possible in the design process. Using more data for training a classifier will in general improve its performance. Furthermore, the more data points used to measure the performance of a classifier, the more accurate this performance estimate can be expected to be.

The latter can be seen, for instance, by looking at (3). Most people follow this obvious recommendation. However, in practice the amount of data that can be used in the design process is typically limited. First, often just a small amount of proper training data is available. Second, there is an upper limit on the number of data points used for either training and/or evaluating a model due to time (and perhaps storage) complexity problems. If the training time or evaluation time of a model scales unfavorably with the number of data points, then this imposes constraints on the sizes of the data sets (e.g., consider batch training of nonlinear support vector machines, which scales at least quadratically with the number of training data points [12]). Thus, we need additional measures to avoid overfitting, beyond that of using more and more data in the design process.

Problem 1 is called traditional overfitting by Langford in [13]. He also summarizes the standard ways to reduce traditional overfitting (including the first recommendation).

Recommendation 2 [13]: The effects of overfitting to training data can be avoided by using independent data for assessing the performance of the classifier, considering a less rich class of predictors, increasing the number of training patterns, and/or integrating over many predictors.

B. Overfitting to Evaluation Data

The first problem and its corresponding recommendations are very widely accepted. However, Problem 2, which occurs when evolving an adaptive pattern recognition system, is almost as fundamental, but often ignored.

Problem 2 (Overfitting to Evaluation Data): When optimizing (the parameters of) a learning algorithm to find a proper hypothesis in a rich class of functions, the error on data used to evaluate different (parameters of) algorithms may be an over-optimistically biased estimate of the generalization performance.

That is, samples repeatedly used for fitness evaluation will not give reliable estimates of generalization performance, regardless of whether they are used for training the machine or not. This holds for all types of evaluation based on a *fixed* data set such as the hold-out method, k-fold cross-validation (rotation method) including the leave-one-out procedure, and bootstrapping [14]. For instance, although cross-validation makes clever use of the available data within a single generation, it does not per se circumvent the problem of the same data points being used in subsequent generations.

Recommendation 3: Overfitting to evaluation data can be avoided if the data used for evaluating the models in one generation g are independent of the data used for evaluating the models in all other generations $g' \neq g$. In the following, it is assumed that S_{train} is independent of S_{evolve}.

This could be achieved if we could sample the patterns in S_{evolve} anew in each iteration independently from the previous iterations. This is, however, either not possible or too expensive in most application scenarios (but a reasonable assumption in computational models of biological evolution, e.g., in the line of [15]). Given a fixed data set S_{evolve} for evaluation, we could, alternatively, choose a maximum number of generations G and a partition of S_{evolve} into G subsets $S^1_{\text{evolve}}, \ldots, S^G_{\text{evolve}}$ and use only S^g_{evolve} for evaluation in generation g. This would implement the above recommendation. This procedure could be refined by keeping track of all data sets S^g_{evolve} that were used to evaluate some ancestor of an individual. Then, the evaluation data set for each individual could be chosen individually from those subsets that have not been used in the evaluation of any ancestor of the individual (and the algorithm stops if it runs out of data subsets for evaluating individuals).[1] Recommendation 3 requires that enough data are available for being split into a large number of subsets, each of which is representative for the learning problem. Unfortunately, this is often not the case in practice.

It is easy to see that overfitting to evaluation data can occur if we agree that overfitting to training data can occur. Let us consider adaptive systems that are trained using a data set S_{train} and may overfit to that data set. If such systems are optimized (e.g., in an evolutionary loop) based on an independent data set S_{evolve}, then the whole adaptation process can be viewed as a single adaptation algorithm operating on the training data set $S'_{\text{train}} = S_{\text{train}} \cup S_{\text{evolve}}$ suffering from overfitting to S'_{train}.

Problem 2 can be viewed as automatic parameter tweak overfitting, which can, in principle, be reduced using the same methods as for minimizing traditional overfitting [13]. This leads to the following good practice guideline.

Recommendation 4: Generalization performances must not be reported and compared based on data used in the design process. The generalization performance should be measured and compared on an external data set S_{extern} that is independent of the data S (and therefore of S_{evolve} and S_{final}) used in the design process.

This means that it is not a valid procedure to use some test data for determining the fitness of individuals and to then go on to use these same data to report the performance of the final hypothesis. This would lead to over optimistic results.

If we agree that we lose generalization performance in the evolutionary process, and if we take the fittest candidate

[1] This procedure was suggested by an anonymous reviewer.

found in this process as our final result, then the duration of the evolutionary optimization is crucial. The stopping time, for example, given by the maximum number of generations, is an important hyperparameter. The optimal stopping time depends on both the algorithm and the problem. This implies that statements such as "the evolutionary algorithm A produced classifiers with better generalization performance than the evolutionary algorithm B although we ran it for fewer generations" should perhaps be "the evolutionary algorithm A produced classifiers with better generalization performance than the evolutionary algorithm B because we ran it for fewer generations." Therefore, for an unbiased comparison of algorithms it is necessary to study the effects of this hyperparameter.

Recommendation 5: The number of evaluations in the evolutionary process should be analyzed as to its influence on overfitting, especially if the final result is taken to be the solution with the best fitness.

The proper number of generations is not known *a priori*. This leads to the idea of using early stopping to regularize the design process. That is, if enough data are available, we consider a data set not used in the evolutionary process at all (also not used for measuring the final generalization performance) for detecting overfitting. This data set can be used to stop the evolutionary optimization as soon as overfitting is observed and/or used for picking the final result from the evolved pattern recognizers. We refer to this data set as final selection data and denote it by S_{final}.

Instead of using an external data set, other early stopping rules can be considered for determining when to stop or for determining when one should have stopped, and then discarding all candidate solutions after that time point. These rules can be based on prior knowledge about the expected performance of a proper hypothesis or they can be based on heuristics analyzing the error trajectories, similar, for example, to early stopping heuristics for neural network learning [16].

The previous recommendations argued several times for hold-out data sets. Of course, if we have limited data resources, this implies taking away data that could otherwise be used for learning, and the more training data, the better the performance of the learning machine. Finding a good tradeoff between these data set sizes can be a challenging problem. Therefore, whenever possible, using data efficient techniques such as cross-validation is advisable.

C. Overfitting to Final Selection Data

If traditional overfitting is first-order overfitting to a data set S_{train} and parameter tweak overfitting is second-order overfitting to $S'_{\text{train}} = S_{\text{train}} \cup S_{\text{evolve}}$, then obviously there is also a third-order overfitting to the final selection data. Accordingly, we can define nth-order overfitting.

Problem 3 (Overfitting to Final Selection Data): Determining the final result based on a final selection data set S_{final} bears the risk of overfitting to S_{final}.

Why should then a third hold-out data set S_{final} be helpful at all? Why should this third overfitting problem be less severe than the second one?

To begin with, it makes a crucial difference that we use S_{final} only for one decision rather than using the same S_{evolve} for decision making in every iteration of the optimization algorithm. Furthermore, choosing from only very few hypotheses in the last selection step may reduce the risk of overfitting. To see this, let us look at (3), which indicates that the risk of wrongly estimating the generalization performance and thus picking the wrong candidate solution is reduced if fewer candidates are considered.

Thus, the considerations in this and the previous section lead to the following suggestion.

Recommendation 6: The final result should be selected using a data set not used in the design process. Preferably, only promising candidates should be considered in this selection process.

Promising candidates could be the set of those individuals that were the best in some generation (in contrast to considering all created individuals). This is the approach taken in, for instance, [17]. One may well argue that there is still a need for a better validation, both theoretical and experimental, that this last recommendation is indeed helpful.

IV. Experiments

This section will show that the previous conceptual considerations can actually make a difference in experimental studies. To this end, we consider a toy feature selection application from the literature and a simple evolutionary algorithm.

A. Example Problem

We consider the classic example by Trunk [18], [19] because it fulfills the following requirements.

1) The problem should be simple to describe and evaluate.
2) The classifier should be consistent (i.e., converge to the Bayes optimal classifier with increasing number of training samples) and the learning process should always find the parameter configuration that is assumed to be optimal given the training data.
3) The generalization performance of a model should be analytically computable.

The last point is of the utmost importance in the context of this paper, because the goal is to analyze the effects of generating a learning system based on finite sample data. Therefore, the analysis should be based on a measure that is independent of a sampling process (i.e., standard finite benchmark data sets are not appropriate in this paper).

In this binary classification problem, d-dimensional feature vectors $\boldsymbol{x} \in \mathbb{R}^d$ have to be assigned to one of two classes, which we label by 1 and -1. Both classes are *a priori* equally likely. The likelihood of observing $\boldsymbol{x} \in \mathbb{R}^d$ given the class $y \in \{\pm 1\}$ is a Gaussian

$$p(\boldsymbol{x} \mid y) = \frac{1}{\sqrt{2\pi}} e^{-\|\boldsymbol{x} - y\boldsymbol{m}\|^2/2} \tag{4}$$

with mean $y\boldsymbol{m}$ and unit covariance matrix. The vector $\boldsymbol{m}$ has components $m_i = \sqrt{1/i}$ for $i = 1, \ldots, d$. The Bayes optimal decision rule assigns an input $\boldsymbol{x}$ to class 1 if $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{m} > 0$ and to class 0 otherwise. The Bayes error is given by

$\rho\left(\sqrt{\sum_{i=1}^{d} 1/i}\right)$, where the auxiliary function ρ is defined by

$$\rho(r) = \int_r^\infty \frac{1}{\sqrt{2\pi}} e^{-z^2/2} \mathrm{d}z. \tag{5}$$

Since $\sum_{i=1}^{d} 1/i$ diverges for increasing d, it is easy to see that increasing the number of features reduces the error probability, which converges to zero as d increases [18]. The pattern recognition algorithm we consider is a simple Bayes plug-in classifier. We assume that we have the prior knowledge that the class-conditional densities are Gaussians with unit covariance matrices and that the problem is symmetric. Thus, our learning algorithm just has to estimate d parameters, namely the components of $\boldsymbol{m}$. Given a training sample $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_\ell, y_\ell)\}$ the algorithm estimates $\hat{\boldsymbol{m}} = \sum_{i=1}^{\ell} y_i \boldsymbol{x}_i$ and assigns an input $\boldsymbol{x}$ to class 1 if $\boldsymbol{x}^{\mathrm{T}}\hat{\boldsymbol{m}} > 0$ and to class 0 otherwise. The generalization error of this plug-in classifier can be computed analytically by

$$\rho\left(\boldsymbol{m}^{\mathrm{T}} \frac{\hat{\boldsymbol{m}}}{\|\hat{\boldsymbol{m}}\|}\right). \tag{6}$$

This can be proved as follows. Each hypothesis corresponds to a separating hyperplane $\{\boldsymbol{x} \mid \boldsymbol{x}^{\mathrm{T}}\hat{\boldsymbol{m}} = 0\}$ through the origin. Because of the symmetry of the problem, we can replace all negative patterns $(\boldsymbol{x}_i, -1)$ by positive patterns $(-\boldsymbol{x}_i, 1)$. Thus, the generalization error is given by the probability that $\boldsymbol{x}^{\mathrm{T}}\hat{\boldsymbol{m}} < 0$ for $\boldsymbol{x}$ drawn according to a Gaussian with mean $\boldsymbol{m}$ and unit covariance matrix. Because the distribution is spherical, we can decompose it into a 1-D Gaussian distribution with unit variance generating random variations from the mean in the directions $\pm\hat{\boldsymbol{m}}$ normal to the hyperplane and a $(d-1)$-dimensional Gaussian distribution responsible for the variations in the subspace perpendicular to $\hat{\boldsymbol{m}}$. Only the component normal to the hyperplane is relevant for the classification. The probability of sampling a pattern that is wrongly classified depends on the distance of $\boldsymbol{m}$ from the separating hyperplane, which is given by $\boldsymbol{m}^{\mathrm{T}}\hat{\boldsymbol{m}}/\|\hat{\boldsymbol{m}}\|$. The probability that a standard normally distributed random variable generates a step in the right direction of at least this length is given by (6).

As $\ell \to \infty$, the plug-in classifier converges to the Bayes optimal decision rule. However, Trunk showed that for fixed ℓ the error probability converges to chance level for $d \to \infty$ [18]. Therefore, this problem commonly serves as an example for showing the necessity of feature selection [19].

B. Evolutionary Feature Selection Algorithm

In the experiments, an evolutionary feature selection algorithm was applied to the example problem described above. The algorithm should select a proper subset of features from a d-dimensional feature vector. Because we want to evolve a simple solution that can be evaluated efficiently, we prefer a small number of features.

Each individual in the evolutionary algorithm was represented by a d-dimensional bit string (i.e., the search space was $\mathcal{G} = \{0,1\}^d$), which encoded a plug-in classifier as described above. Only if the ith bit was one did the plug-in classifier use feature i.

Two fitness functions were considered. Let $\|a\|_1$ denote the number of features used by individual $a \in \mathcal{G}$ (i.e., the number of ones in the bit string). In the first fitness function, the fitness was computed using a fixed hold-out data set. The data available for the design process were partitioned into a fixed training data set S_{train} and a test data set S_{evolve}. Each individual generated a hypothesis based on S_{train} and was evaluated using S_{evolve}

$$\Phi_{\text{hold-out}}(a) = \mathrm{er}_{S_{\text{evolve}}}(A_a(S_{\text{train}})) + \frac{\|a\|_1}{d \cdot |S_{\text{evolve}}|}. \tag{7}$$

The second term introduces a parsimony pressure. However, the scaling by $(d \cdot |S_{\text{evolve}}|)^{-1}$ ensures that the test error dominates the influence of the number of features. Only in the case of equal classification performance is the candidate solution with fewer features preferred.

In the second fitness function Φ_{CV}, k-fold cross-validation was used. The available data were partitioned into k disjoint, equally sized subsets. For each of these subsets $S_{\text{test},\,i}$, $i = 1, \ldots, k$, the classifier was trained using the union $S_{\text{train},\,i}$ of the $k-1$ other sets and a test error was computed on the left-out subset. The final cross-validation error is the average of the k test errors

$$\Phi_{\text{CV}}(a) = \frac{1}{k} \sum_{i=1}^{k} \mathrm{er}_{S_{\text{test},\,i}}(A_a(S_{\text{train},\,i})) + \frac{\|a\|_1}{d \cdot \sum_{i=1}^{k} |S_{\text{test},\,i}|}. \tag{8}$$

The parent and offspring population size were set to $\mu = \lambda = 100$. The initial population contained bit strings drawn uniformly at random. The offspring was generated by uniform crossover and bit-flip mutation [20]. Each parent reproduced once per generation. The probability of uniform crossover was $p_c = 0.8$. The probability a bit's flipping was $p_m = d^{-1}$. The μ new parents were selected from the current parents and offspring by elitist EP-style tournament selection with tournament size $Q = 2$ [21]. The number of splits in the cross-validation procedure was set to $k = 5$ [14] and 100 independent trials were conducted per problem scenario.

The maximum number of features was set to $d = 500$, the number of generations to $G = 500$, and $|S_{\text{evolve}} \cup S_{\text{train}}| = 2000$. In the first fitness function, $|S_{\text{train}}| = 1000$ was taken. Additionally, an independent data set S_{final} with $|S_{\text{final}}| = 100$ was created.

Furthermore, a series of experiments was conducted in which the data set S_{evolve} was resampled anew in each generation as discussed in Section III-B.[2] In many applications, resampling is not possible. However, this scenario, which follows Recommendation 3, matches computational models of natural evolution and also corresponds to the scenario in which the amount of available data is so large that we can partition it into subsets each of which is only used in one generation.

Let $A = \left\{a_i^{(g)} \mid i = 1, \ldots, \lambda \wedge g = 1, \ldots, G\right\}$ be the set of all candidate solutions created in the evolutionary process. We define $A^* = \left\{a^{*(g)} = \mathrm{argmin}_{a \in P^{(g)}} \Phi(a) \mid g = 1, \ldots, G\right\} \subset A$,

[2] In such a scenario, using elitist selection and no re-evaluation of the parent individuals is in general not recommended. However, this setting was used in the experiments in order to keep the difference from the experiments with a fixed hold-out data set as small as possible.

the set of all individuals that have been the best in at least one generation. Three ways to determine the final result of the evolutionary optimization were considered.

1) Picking the candidate solution having the best overall fitness

$$\text{argmin}_{a \in A}\, \Phi(a). \tag{9}$$

2) Selecting the final result by using the additional data set S_{final} and choosing the individual minimizing $\text{er}_{S_{\text{final}}}$ among all created individuals

$$\text{argmin}_{a \in A}\, \text{er}_{S_{\text{final}}}(a). \tag{10}$$

3) Selecting the final result by choosing the individual minimizing $\text{er}_{S_{\text{final}}}$ among all individuals that were the best individual in some generation

$$\text{argmin}_{a \in A^*}\, \text{er}_{S_{\text{final}}}(a). \tag{11}$$

The latter is closely related to employing early stopping (if the errors on S_{final} of the generations' best individuals were first nonincreasing and then, after reaching a minimum, nondecreasing, a naive early stopping would give the same result).

All experiments were conducted using the Shark machine learning library [22].

C. Results and Discussion

Fig. 1 summarizes the results without resampling. Due to the elitist selection, the fitness is nonincreasing. For both fitness functions, overfitting to the evaluation data was observed, as expected. This demonstrates the importance of Recommendation 4. In the early generations, the generalization error of the best individual in the population went down before it increased again, the expected effect that gave rise to Recommendation 5. The cross-validation-based fitness function performed better. Regardless of which of the three ways of determining the result a^* was applied, Φ_{CV} yielded machines with better generalization performance than $\Phi_{\text{hold-out}}$. Furthermore, overfitting was delayed. Still, even when cross-validation was used, the generalization loss started surprisingly early. The experiments show that stopping the evolutionary process early can indeed be crucial.

Before overfitting starts, selecting the final result based on fitness would have given the best generalizing solution. However, already after 50–100 generations, picking the final result from A^* based on S_{final} should have been preferred in accordance with Recommendation 6. Choosing the solution from the generations' best using the hold-out final selection data suffered less from overfitting. The differences between the two strategies relying on S_{final} are highly statistically significant (Wilcoxon rank-sum test in generation 500, $p < 0.001$).

Considering only the best individuals from each generation appeared to be clearly superior to choosing from all created individuals. The latter strategy was much worse in the beginning and gave similar results later in the course of evolution compared to considering simply the fitness. This supports our considerations about concentrating on promising candidates

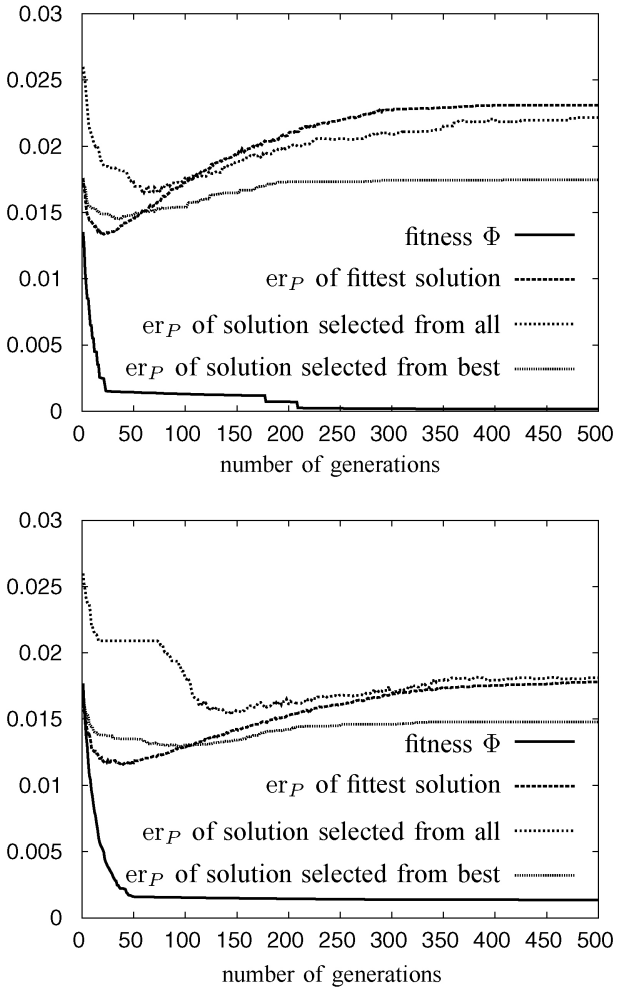

Fig. 1. Results using a single hold-out data set (top) and 5-fold cross-validation (bottom). The curves refer to the median of 100 trials. The solid line shows the fitness based on a hold-out data set. The other three curves show the true generalization performance er_P of solutions. The first nonsolid line gives the generalization error of the fittest individual in the population. The second nonsolid line gives the performance of the individual that was selected from all solutions generated so far using a hold-out data set S_{final}. The third nonsolid line shows the performance if the final result was not selected from all individuals using S_{final}, but only from the best in a generation. Note that choosing based on a finite data set S_{final} led to an increase in the true generalization performance er_P shown in the figure.

in the final selection step, as stated in Recommendation 6 and realized in [17].

Thus, using a hold-out data set S_{final} and monitoring performance of the generations' best individuals using S_{final} can indeed help. The experiments also indicate the potential advantages of selecting the final result by taking the best individual from A^* in terms of $\text{er}_{S_{\text{final}}}$, as recommended in Section III-C.

Increasing the size of S_{final} would of course reduce the effect of overfitting (see Recommendation 1). In the experiments, a small S_{final} was considered to pronounce the effects of overfitting, but in practice it often has to be small because all other available data are needed for the model building process. If one can follow Recommendation 1 and arrange that S_{final} be large, then computing the error on S_{final} is so close to er_P that no overfitting will occur.

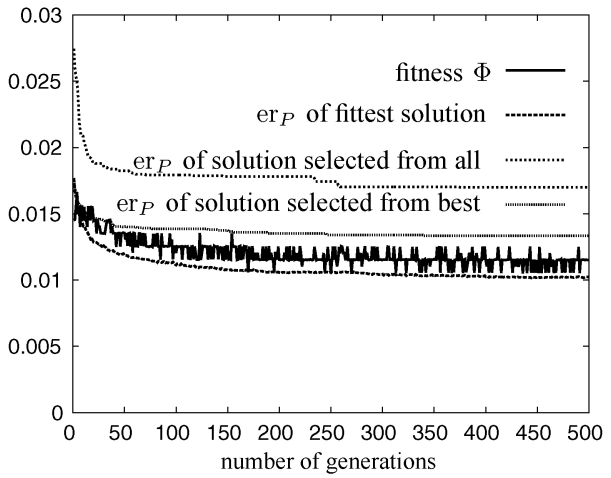

Fig. 2. This plot corresponds to the upper plot in Fig. 1 except that the hold-out data set was resampled in every generation.

Fig. 2 depicts the results using resampling. In accordance with Recommendation 3, overfitting was not observed. Due to the randomness in the fitness evaluation, the fitness trajectory was noisier than in Fig. 1. In this scenario, choosing the fittest candidate to be the result appeared to be better than following Recommendation 6 and choosing it using the (small) finite S_{final} (due to Problem 3). Nonetheless, in this setting, selecting only from the best in a generation still turned out to be preferable to selecting from all.

V. Conclusion

Evolutionary algorithms are frequently used for optimizing adaptive pattern recognition systems. As is the case for all learning and model selection algorithms, they are susceptible to overfitting. Often, the different types of overfitting that can occur are not properly considered in the algorithm and experimental design. This can lead to suboptimal performance, as well as to overoptimistic evaluations.

The purpose of this paper was to raise awareness of the different types of overfitting and to explain potential pitfalls when evolving machine learning systems. These considerations focus on the predominant scenario in which the fitness computations are based on point estimates. This paper listed ways of fighting overfitting, but did not, of course, provide an ultimate answer to the problem. A simple experiment showed that the negative effects of different types of overfitting can easily be observed in practice.

The main recommendation may be summarized as follows. The generalization performance of an (evolved) hypothesis must, of course, not be measured by its performance on data used for its design. Furthermore, when evolving adaptive pattern recognition systems, it can be helpful to use a pristine data set to select the final result from a small set of evolved candidate solutions, for example, from the best individuals of each generation. Then, fresh external data should be used to evaluate the selected solution. The problem with these recommendations is that they require data, and it may be expensive or even impossible to get enough data. Furthermore, the question arises of how to split the finite amount of data available for the design process. The larger the training set, the better the hypotheses produced by a learning algorithm. The larger the data sets involved in computing the fitness, the more precise the performance estimates for guiding the evolution. The larger the final selection data, the more reliable is the choice of the final result. How to find a good tradeoff is an open research question.

Simply limiting the number of generations in the optimization process and not letting it run until convergence seems to be a data and time efficient solution. This introduces a crucial hyperparameter, the maximum number of generations, that is problem-dependent and hard to choose *a priori*. A better solution appears to be to decide on the stopping time post hoc, either based on knowledge about the expected performance of a proper hypothesis or based on heuristics analyzing (training or hold-out data set) error trajectories.

Acknowledgment

The author would like to thank V. Heidrich-Meisner, A. Fischer, and T. Glasmachers for their comments on this manuscript.

References

[1] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.
[2] S. Nolfi, "Evolution and learning in neural networks," in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed., 2nd ed. Cambridge, MA: MIT Press, 2002, pp. 415–418.
[3] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: From architectures to learning," *Evol. Intell.*, vol. 1, no. 1, pp. 47–62, 2008.
[4] H. Fröhlich, O. Chapelle, and B. Schölkopf, "Feature selection for support vector machines using genetic algorithms," *Int. J. Artif. Intell. Tools*, vol. 13, no. 4, pp. 791–800, 2004.
[5] F. Friedrichs and C. Igel, "Evolutionary tuning of multiple SVM parameters," *Neurocomputing*, vol. 64, no. C, pp. 107–117, 2005.
[6] T. Glasmachers and C. Igel, "Uncertainty handling in model selection for support vector machines," in *Parallel Problem Solving From Nature* (LNCS, vol. 5199), G. Rudolph, Ed. Berlin, Germany: Springer-Verlag, 2008, pp. 185–194.
[7] C. Igel, "Evolutionary kernel learning," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Berlin, Germany: Springer-Verlag, 2010.
[8] D. J. Hand, "Academic obsessions and classification realities: Ignoring practicalities in supervised classification," in *Classification, Clustering, and Data Mining Applications*, D. Banks, L. House, F. R. McMorris, P. Arabie, and W. Gaul, Eds. Berlin, Germany: Springer-Verlag, 2004, pp. 209–232.
[9] M. Anthony and P. L. Bartlett, *Neural Network Learning: Theoretical Foundations*. Cambridge, U.K.: Cambridge Univ. Press, 1999.
[10] V. Vapnik and A. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory Probab. Its Applicat.*, vol. 2, no. 16, pp. 264–280, 1971.
[11] P. Bartlett and S. Mendelson, "Gaussian and Rademacher complexities: Risk bounds and structural results," *J. Mach. Learning Res.*, vol. 3, pp. 463–482, Nov. 2002.
[12] L. Bottou and C.-J. Lin, "Support vector machine solvers," in *Large Scale Kernel Machines*, L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, Eds. Cambridge, MA: MIT Press, 2007, pp. 301–320.
[13] J. Langford. (2005). *Clever Methods of Overfitting* [Online]. Available: http://hunch.net/?p=22
[14] T. Hastie, R. Tibshirani, and J. Friedmann, *The Elements of Statistical Learning*. Berlin, Germany: Springer-Verlag, 2001.
[15] L. G. Valiant, "Evolvability," *J. ACM*, vol. 56, no. 1, p. 3, 2009.
[16] L. Prechelt, "Early stopping—but when," in *Neural Networks: Tricks of the Trade* (LNCS, vol. 1524), G. B. Orr and K. R. Müller, Eds. Berlin, Germany: Springer-Verlag, 1998, pp. 55–69.

[17] S. Wiegand, C. Igel, and U. Handmann, "Evolutionary multiobjective optimization of neural networks for face detection," *Int. J. Computat. Intell. Applicat.*, vol. 4, no. 3, pp. 237–253, 2004.
[18] G. V. Trunk, "A problem of dimensionality: A simple example," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 1, no. 3, pp. 306–307, Jul. 1979.
[19] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical pattern recognition: A review," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 1, pp. 4–37, Jan. 2000.
[20] T. Bäck, *Evolutionary Algorithms in Theory and Practice*. Oxford, U.K.: Oxford Univ. Press, 1996.
[21] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. New York: IEEE, 1995.
[22] C. Igel, T. Glasmachers, and V. Heidrich-Meisner, "Shark," *J. Mach. Learning Res.*, vol. 9, pp. 993–996, Jun. 2008.

Christian Igel (SM'04) studied computer science at the Technical University of Dortmund, Dortmund, Germany. He received the Doctoral degree from the Faculty of Technology, Bielefeld University, Bielefeld, Germany, in 2002, and the Habilitation degree from the Department of Electrical Engineering and Information Sciences, Ruhr-University Bochum, Bochum, Germany, in 2010.

From 2002 to 2010, he was a Junior Professor for optimization of adaptive systems with the Institut für Neuroinformatik, Ruhr-University Bochum. In October 2010, he was appointed a Professor with special duties in machine learning with the Department of Computer Science, University of Copenhagen, Copenhagen, Denmark.