SATISFy: Towards a Self-learning Analyzer for Time Series Forecasting in Self-improving Systems

Christian Krupitzer, Martin Pfannemüller, Jean Kaddour, and Christian Becker
Chair of Information Systems II, Universität Mannheim, Germany
Email: {christian.krupitzer, martin.pfannemueller}@uni-mannheim.de, jk@jeankaddour.com, christian.becker@uni-mannheim.de

***Abstract*—Self-adaptive systems can adapt their managed resources to reflect changes in their environment or the resources themselves. However, sometimes these systems cannot handle situations due to uncertainty. Self-improvement enables the adaptation of the decision logic of such systems for coping with new situations. Proactive analysis predicts the need for self-improvement as well as reduces the delay for self-adaptation. However, implementing proactive analysis is a complex task which requires developers to analyze different algorithms and parameter combinations for finding the best fitting setting for the given data. This paper addresses this issue by presenting a model for a self-learning analyzer for proactive reasoning based on time series forecasting which can support self-improvement at runtime. We present a prototype implementation of such an analyzer and evaluate its performance for traffic prediction in an adaptive traffic management system.**

I. Introduction

Self-adaptive systems are able to adapt autonomously to changing environments or system configurations, e.g., hardware defects. To enable these adaptations, the system implements a control structure, the so-called MAPE cycle [1], i.e., monitoring the system resources and the environment, analyzing the results for changes that trigger an adaptation, planning the adaptation, and executing the planned actions. For analyzing and planning – often subsumed as reasoning – the approaches are classified according to their decision criteria into: rule-based, model-based, goal-based, and utility-based approaches [2]. However, uncertainty at runtime introduced due to a gap between design and runtime information leads to incompleteness or obsolescence of adaptation decision criteria. To address these issues, it is necessary that the adaptation logic is able to constantly improve its behavior, i.e., to self-improve [3]. Therefore, machine learning can be integrated to control this self-improvement.

One approach to detect a concept drift [4] – i.e., a change in data patterns for analyzing – uses predictions of data. Time series forecasting is a common predictive modeling problem which is faced in many fields, such as stock market predictions in finance or traffic forecasting. Although software developers, data scientists, and mathematicians can rely on a growing number of tools capable of running sophisticated calculations, there are only limited studies for performance optimizations of computing systems for time series available [5]. Further, developers often have to choose from many different techniques for prediction and their configurations, which leads to non-optimized configurations. The quality of the data, as well as their structure, influence the choice of the best algorithm and according parameters for prediction. This hinders their applicability. Consequently, such prediction techniques are often not sufficiently addressed in the adaptation logic [2].

This paper aims to support developers of self-adaptive systems in time series forecasting with a flexible analyzer to systematically select – online or off-line – the best performing learning algorithm with optimized parameters for a particular time series sub-sequence. The system obtains the space of different models during runtime and outputs its configuration afterward. The artifact is self-contained and can be used during development to identify the best approach for proactive analysis as well as at runtime to improve the analyzing algorithm. For the purpose of gathering new research findings which should attain ready-to-use and reproducible guidelines, the approach proposed in this paper is executed and evaluated in three experiments in the context of a real-world use case for traffic prediction. This paper's contributions are:

- The design of the *Self-learning analyzer for time series forecasting* (SATISFy) that supports self-learning prediction.
- A reusable self-contained implementation of the SATISFy module which can be used at design time or for self-improvement of the analyzer at runtime.
- An evaluation of the module in an intelligent transportation system for traffic prediction.

The remainder of this paper is structured as follows: Next, Section II discusses approaches for self-learning analyzers. Section III illustrates our system model. Following this system model, Section IV covers the implementation details. Section V elaborates on conducted experiments and discusses the self-learning analyzer's performance. Lastly, Section VI closes the paper with a conclusion.

II. Related Work

First, this section presents related work in the field of uncertainty and handling the same. This includes work with self-improvement layers for prediction and optimization. The second part of this section briefly summarizes works concerning learning in self-adaptive systems in general[1].

[1]We omit related work related to time series forecasting techniques, algorithm selection, and similar topics as the focus of the paper is on applying them and not contributing the this research area.

[6] gives an overview on categories for uncertainty in self-adaptive systems as well as approaches how to tackle them. The authors specify two large categories of uncertainty with sub-categories: variability and lack of knowledge. Four approaches are identified according to the variability dimension: *Rainbow*, *FUSION*, *ADC*, and *RESIST*. In the following, each approach is briefly introduced including a short description according to the uncertainty of the variability. The Rainbow approach for handling variability is described in [7]. For problem detection, the authors use a probabilistic method of a running average to mitigate noise effects as part of the monitoring data. This technique only focuses on reducing the noise of the input data. There is no optimization of the analyzing in another way. FUSION (FeatUre-oriented Self-adaptatION) is a framework for learning the impact of adaptation decisions on the system's non-functional goals [8]. The approach incorporates the variability of the context mainly by providing the possibility to learn the impact of certain features on the metrics at runtime. This learned knowledge is only used in the planning phase of the adaptation cycle. As shown in [3], FUSION is one approach for self-improvement. ADC (Anticipatory Dynamic Configuration) predictively specifies the required resources for a system's tasks [9]. It mitigates the uncertainty category parameters over time using probabilistic models by predicting the possible resource supply and demand of a system. Its goal is to maximize utility as well as to reduce the number of adaptations. ADC does not provide a generic framework. Additionally, only a single prediction approach is supported. The aim of RESIST (REsilient SItuated SofTware system) is mainly to predict the reliability of a system at runtime and adapt its configuration proactively [10]. Using probabilities it can predict the reliability of the system in the future for allowing proactive adaptations. RESIST uses Hidden Markov Models as well as Discrete Time Markov Chains here. A major drawback is that the approach is focused only on the prediction of reliability.

Additionally, [11] introduces a predictive analysis phase for the MAPE cycle. Using historical data of temperature sensors the authors want to reconfigure the sensing rate of the sensor nodes using an event-condition-action rule based on temperature predictions. The authors evaluate and select one classifier according to the available dataset at design time.

The following shortly presents related work in the field of learning and self-optimization for self-adaptive systems. Learning in self-adaptive systems is tightly coupled to self-optimization. A self-adaptive system continuously optimizes structure, parameters, or algorithms of the managed resources in order to become more efficient with regard to performance or cost [1]. Learning focuses on structural optimization (e.g., [8], [12]–[17]). Several approaches use reinforcement learning for structural optimization (e.g., [14], [18]–[22]). Other works address parameter optimization (e.g., [23]–[25]). Algorithmic optimization aims at optimizing the algorithms of the managed resources (e.g., [26], [27]). The field of Search-based Software Engineering (SBSE) applies search-based meta-heuristic techniques to software engineering – e.g., genetic programming – for examining large search spaces of candidate solutions to find a (near) optimal solution to problems in designing software. Contrary to the traditional approach, dynamic SBSE applies the principles of SBSE at runtime to determine the most suitable system configuration during planning of self-adaptation (e.g., [28]–[32]). Learning mainly targets the planning. Procedures for monitoring, analyzing, and executing have to be added.

All presented approaches are mainly tailored to one specific application or one single source of uncertainty. Additionally, approaches for learning new configurations of the analyzer or integrating proactive analyzing are rare in literature. Our goal is to support a large variety of prediction applications rather than tailoring the system towards a single case.

III. System Model

We used the MAPE-K model as underlying system model [1]. It separates the managed resources that perform the system functionality from the adaptation logic that controls the resources. For controlling the resources, the adaptation logic observes the resources as well as their environment and reasons on this data to determine necessary adaptation. As a de facto standard in the self-adaptive systems domain [2], [33], the adaptation logic integrates the MAPE-K functionality [1] for (i) monitoring the resources and their environment, (ii) analyzing the captured information, (iii) planning adaptation if required, and (iv) controlling the execution of the adaptation on the resources. Additionally, a knowledge repository can support the four main functions with knowledge, e.g., store (i) data captured by monitoring, (ii) the rules for analyzing, or (iii) information on the available system components for planning. Often, analyzing is reactive, i.e., the data is collected and the current state is compared with a desired one. Hence, issues in the performance of the system are detected after a triggering event. The resulting adaptations happen with a delay, which might decrease system performance. Proactive analysis uses the collected data to predict future states of the system or the environment. This enables proactive adaptation [2], i.e., the identification of issues and adaptation before they might decrease the system performance. However, the implementation of prediction with time series forecasting is complex, as the choice of an algorithm and training of a model – i.e., learning the algorithm's parameters – depends on the data. Using a self-learning analyzer, developers may test different algorithms and configurations to identify the most suitable configuration for time series forecasting in a specific application. Furthermore, the self-learning analyzer can do this at runtime and change the analyzer's configuration. SATISFy is a component that can be used as analyzer of time series data or complement an analyzer with time series prediction.

First of all, the analyzer needs data in the form of a time series as input. In order to cast time series data into the format of a regression problem and to apply machine learning methodologies, it is denoted as a stochastic process [34] consisting of random variables $X = X_1, X_2, \ldots, X_t - 1, X_t$ indexed by the time index $t \in \mathbb{N}$. The objective is to identify

correlations between the different data items to predict future values. As the data is comprised of several variables, the analyzer requires a definition of the target variables. Further, the data might be filtered and/or clustered. Clustering might improve the performance of the time series forecasting. Filtering might help to identify wrong data – e.g., unreliable sensor data – that could corrupt the prediction. Consequently, the developers should be able to define rules for both mechanisms. Last, time series forecasting depends on the time horizon for the prediction. This horizon should be flexible, so that different time horizons can be tolerated simultaneously. In the following, this section presents our design for such a self-learning analyzer component. Further, Section III-B explains the *Combined Algorithm Selection and Hyperparameter Optimization* (CASHO) for optimization of a time series forecasting algorithm's parameters.

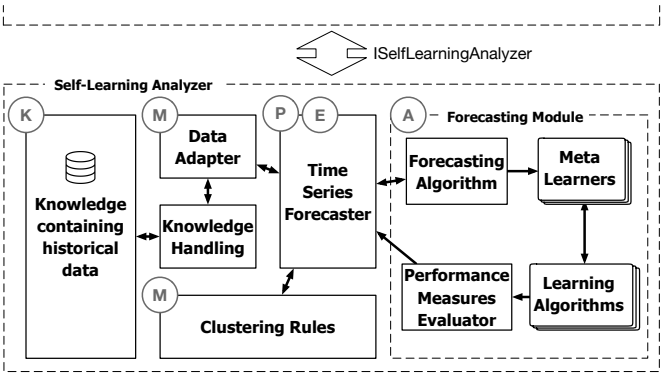

Fig. 1. An abstract white-box architecture design of the self-learning analyzer components. The `ISelfLearningAnalyzer` offers an interface to a module for self-improvement or other MAPE components. (M), (A), (P), and (E) reference the corresponding MAPE functions.

A. Design of the Self-learning Analyzer

SATISFy is contained of various elements that interact with each other. All are implemented as sub-modules of a meta-analyzer component (cf. Figure 1). The main module, the *Time Series Forecaster*, implements an interface called `ISelfLearningAnalyzer` to offer a structured access to its functionality. It provides methods for retrieving and loading the current data sets, clustering rules, or learning algorithms. These methods can be accessed by other modules – e.g., other MAPE components for accessing data or self-improvement of the analyzer – or a stand-alone interface, e.g., for testing at design time.

The *Time Series Forecaster* acts as the main controller of the system managing all the other components and the process sequence. It extracts data from the knowledge component – e.g., from a database or other data sources – and performs clustering of the data set, if desired (M)[2]. The *Forecasting Module* constitutes the prediction logic (A). The forecasting algorithm operates on the abstraction level of the time series analysis methodology, aware of the temporal order of the data. Since regression learners are not created for the need of detecting temporal orderings, the forecasting algorithm encodes the time dependency between each instance and its prede-/ and successor by creating lagged variables. While the *Meta-Learners* coordinate the subordinate *Learning Algorithms*, the latter directly report their results to the *Performance Measures Evaluator*. The *Time Series Forecaster* chooses the best algorithm (P) and configures its parameter according to the results of the *Performance Measures Evaluator* (E). For this purpose, the developer can configure various settings, such as the features, the targets, whether all data points should be used or just a moving window of a specified number of observations. Further, the developer is responsible to transform the data to a numerical representation that the time series algorithm can handle.

[2](M), (A), (P), and (E) reference the corresponding MAPE functions.

B. An Approach for Combined Algorithm Selection and Hyperparameter Optimization

The *Forecasting Module* is responsible to test and evaluate various algorithms and different parameter combinations for them. The results are used by the *Time Series Forecaster* to choose the best algorithm and its optimized parameters for the underlying data patterns. Therefore, two approaches are available: (i) ensemble learning or (ii) choosing the best algorithm and optimizing its parameters. In this paper, we focus on supervised learning algorithms for regression and neglect techniques such as Exponential Smoothing or ARIMA.

In *ensemble learning*, the idea is to combine various predictive models which learn from the same training set and obtain a higher predictive accuracy by doing so [35]. They are categorized as *meta-algorithm* or *meta-learner*. The predictive output can be the single output of the aggregation's best-performing algorithm or a combined calculation of all outputs. Moreover, the predictors' diversity determines how comprehensively the meta-learner can grasp different kinds of relationships. Provided that, different sorts of algorithms should be chosen unless it has proven that these algorithms come up with the best results for the particular data set. One major drawback of this method is the loss of interpretability when evaluating the ensemble system as the algorithm only returns an aggregated predicted value. Hence, it might not be clear how the algorithms contributed to the prediction.

An alternative to ensemble learning is to test various algorithms and their parameter combinations similar to a brute force approach. For this, this subsection explains the *Combined Algorithm Selection and Hyperparameter Optimization* (CASHO) algorithm for handling the analysis of several prediction algorithms. CASHO is used within the *Time Series Forecaster*. It clusters the data according to the defined clustering rules and searches for each cluster the best fitting algorithm using a meta-learner algorithm, such as AutoWEKA or grid search. Further, it enables hyper-parameter optimization. Hyper-parameters are parameters which need to be fixed prior to the beginning of the actual training

phase of a learning algorithm [36]. They express "higher-level" properties of the model independent from the underlying data set. Thus, they can not be derived from the learning process itself. However, tuning the parameters to optimize the performance of its model for a given data set is possible and offers a potential for improving the fit of a model to the training set. Most learning algorithms feature a set of *hyper-parameters* $\lambda \in \Lambda$ that modify its behavior. Given a learning algorithm $A^i \in \mathcal{A}$, the goal of *hyper-parameter optimization* is to determine the hyper-parameter setting $\lambda^{i*} \in \Lambda^i$ with optimal regression performance where Λ^i denotes the parameter space for the algorithm A^i. The *loss* generated by A when trained on $\mathcal{D}_{train}$ and evaluated on $\mathcal{D}_{valid}$ is noted as $\mathcal{L}(A, \mathcal{D}_{train}, \mathcal{D}_{valid})$. While applying k-fold *cross-validation*, a fixed number $k \in \mathbb{N}$ of folds is chosen and the training data is split into k equal-sized partitions $\mathcal{D}^1_{valid}, \ldots, \mathcal{D}^k_{valid}$, and sets $\mathcal{D}^i_{train} = \mathcal{D} \setminus \mathcal{D}^i_{valid}$ for $i = 1, \ldots, k$. The process is repeated k times, each time using another partition for training and the remainder for testing. Combining both, algorithm selection and hyper-parameter optimization, the goal of the CASHO algorithm can be expressed as

$$A^*_{\lambda^*} \in \underset{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}}{\operatorname{argmin}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A^{(j)}_{\lambda}, \mathcal{D}^{(i)}_{train}, \mathcal{D}^{(i)}_{valid}).$$

Algorithm 1 shows the CASHO algorithm. It consists of two parts. First, it checks if cluster rules (and data) are specified. If so, the clustering of the data set is performed. Afterwards, for each cluster (possibly the data set as a whole) and respectively for each preselected algorithm, first the best fitting hyper-parameter configuration is selected – by the established meta-learner – and its error measure is recorded. Finally, the algorithm assigns the best performing algorithm configuration for each cluster. In contrast to ensemble learning, the interpretability of the result is given as CASHO returns the best-fitting algorithm and its parameters.

Algorithm 1: CASHO

Data: $\mathcal{D} = \{(x_1, y_1), \ldots, (x_t, y_t)\}$, target variables $\mathcal{Y} = Y^1, \ldots, Y^i$, clustering rules $\mathcal{R} = R^1, \ldots, R^i$, algorithms $\mathcal{A} = A^1, \ldots, A^j$
Result: optimized algorithms for each cluster's target variables $\mathcal{A}^* = A^*_{D_1,Y_1}, \ldots, A^*_{D_1,Y_i}, A^*_{D_2,Y_1}, \ldots, A^*_{D_g,Y_i}$, where g is the amount of clusters

```
if D ≠ ∅ then
    if R ≠ ∅ then
        D_1, ..., D_g ⟵ clusters of D generated by R
    else
        D ⟵ D
for D^(i) ∈ D do
    for A^(j) ∈ A do
        A^(j)_λ* ⟵ best model of A^(j) for D^(i)
        for Y^(k) ∈ Y do
            ε^(j),(k) ⟵ error of A^(j) for the prediction of Y^(k)
              from D^(i)
    for Y^(k) ∈ Y do
        A*_{D_i,Y_k} ⟵ A_{D_i,Y_k} with smallest error ε^(j),(k)*
```

IV. Implementation

This section describes the implementation of a prototype of SATISFy using Java. The self-contained module is reusable and can be integrated into the adaptation logic of a self-adaptive system for interacting with other MAPE components. In the following, we target the implementation of the module itself omitting the implementation of the stand-alone variant.

A. Prediction Optimization Approaches

In accordance with the two strategies of algorithm and parameter optimization from Section III-B, ensemble learning and CASHO, we implemented three meta-learning approaches which aim to optimize the predictive performance. *Stacking* is a representative for ensemble learning. *Multi Search* and *AutoWEKA* fit the CASHO algorithm presented in Algorithm 1. All of them are instances of an abstract wrapper class that offers the possibility to integrate any meta-learner that corresponds to the aforementioned system model. These meta-learning approaches integrate various base learning algorithms. For all learning algorithms except one, we used the implementations provided by the data mining framework *Waikato Environment for Knowledge Analysis* (WEKA) [37]. It incorporates various machine learning techniques and contributes as a framework for developing new schemes by providing suitable interfaces. Additionally, it allows extending its basic functionality with external packages. Figure 2 illustrates this three-layer structure. Next, this section describes the three meta-level learner, Afterwards, it describes the implementation of the base learners.

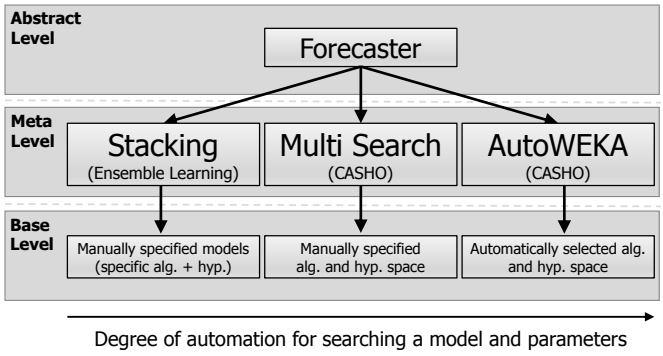

Fig. 2. Overview about the different levels of the prediction performance optimization approaches. Alg. = Algorithms, Hyp = Hyper-parameters.

1) Stacking: *Stacking* or *stacked generalization* aggregates multiple learning algorithms for ensemble learning. The idea behind Stacking is that it positions a meta-learner one level above the *base learners* – the original learning algorithms – which tries to learn which algorithm is the best for a specific problem instance [38]. This is carried out by transferring the predictions of all base learners as attributes to an additional training set of the *meta-model* where all outputs are accumulated. Different to other meta-learners, Stacking does not focus on a winner-takes-all strategy but can balance the outcomes over all algorithms if reasonable. The selection of the meta-learner's algorithm is not ascertainable, hence, [35] claims it is

difficult to theoretically analyze its behavior. On the contrary, in [39], Stacking is stated as the "best-known" meta-learning approach.

2) Grid Search and Multi Search: Grid Search exhaustively or randomly searches through a manually specified subset of the hyper-parameter space of a learning algorithm. This is comparable in complexity to a brute-force search. Grid Search is considered the most widely used strategy for hyper-parameter optimization [36]. We use the relative *multisearch* package provided by WEKA[3] which allows an arbitrary number of parameters to search through. A disadvantage of *Multi Search* is that it does not support automatic search space exploration. When the best parameter found is on the border of the manually specified parameter value range, extending the grid and continuing the search would be adjuvant for finding a better one.

3) AutoWEKA: AutoWEKA [40] is a meta-learner that automatically performs combined algorithm selection and hyper-parameter optimization for a given data set exploring all in *WEKA* available classification and regression algorithms as well as attribute selectors and evaluator filters. The latter are specified in the data pre-processing phase before the actual prediction is executed. AutoWEKA provides a completely automated approach which means the user does not have to specify which algorithms and corresponding hyper-parameter spaces should be taken into account in contrast to Multi Search. The main difference between AutoWEKA and other hyper-parameter optimization approaches is its calculation manner of *Bayesian Optimization*.

B. Implementation of Learning Algorithms

The afore-mentioned meta-learning approaches integrate various base learning algorithms. The following in WEKA implemented learners were used by the Stacking and Multi Search approach: `DecisionStump`, `DecisionTable`, `GaussianProcesses`, `IBk`, `KStar`, `LinearRegression`, `LWL`, `M5P`, `M5Rules`, `MLPRegressor`[4], `MultilayerPerceptron`, `RandomForest`, `RandomTree`, `REPTree`, `SMOreg`, `ZeroR`. The remaining classifiers available in WEKA are not used since they are not able to handle numeric attributes which is an underlying requirement for time series.

The configuration of a meta-learner can be a tedious task when the algorithm or hyper-parameter space has to be manually defined. Classes inherited from the abstract class `MetaLearningAlgorithm` remedy this effort so that meta-learners can be set up in less time. The class `ForecastingAlgorithm` is responsible for generating and evaluating predictions. For that reason, it accesses the other classes and can handle their distinctive characteristics.

The considered data sets require an attribute representing the time, the so-called `timestamp` and remaining numeric attributes. Date attributes, commonly used for representing the timestamp, are strings with the combined date and time format `yyyy-MM-dd'T'HH:mm:ss`. The encoding of the temporal ordering is done by the *timeseriesforecasting*-package[5].

We implemented clustering based on clustering rules that relate to time periods of weekdays, hours and minutes. A generic implementation supports developers in integrating further clustering criteria or defining additional clustering rules.

V. Evaluation

This section presents the evaluation of the optimization schemes in a traffic prediction use case. It targets SATISFy's core functions: inserting and application of time-based clustering rules, the performance of different accuracy optimization approaches and the exposure of information about the model-data fit of various predictive models to the corresponding data sets. To address these issues in the evaluation, it is guided by the following questions:

- **EQ1:** Which impact on the performance does clustering during the data pre-processing phase have? Different clustering rules are applied so that the resulting clusters vary significantly in their size.
- **EQ2:** Which impact on the performance does CASHO and ensemble learning (here: Stacking) have? WEKA's default configuration is compared against two optimization approaches.
- **EQ3:** Which algorithm does perform best on average? Multi Search as white-box approach guarantees that the selected algorithms share the same computing resources and can be compared against each other.

In the following, this section introduces the traffic prediction use case and the evaluation setting. Afterwards, it presents the results of the experiments and discusses them.

A. Use Case: Traffic Prediction

To evaluate the SATISFy under realistic circumstances, we use our implementation for prediction of traffic flows. This is in line with other works in the adaptive systems area, e.g., Sommer *et al.* uses ensemble time series for traffic prediction [41]. The chosen data set for the evaluation offers the traffic flow and average speed of all vehicles passed the counting station in the last 15 minutes. For the following experiments, a counting station on highway A8 near the city of Stuttgart, Germany was taken which contains 31392 unique time points, ranging from 2nd January, 2008 to 30th December, 2008. The data origins from [42].

B. Evaluation Setting

For the experiments, two IBM server blade instances were separately used. In each experiment, the performance measure mean absolute error (MAE) was selected to be optimized. Since the achieved accuracy highly depends on the used data set, the comparison of error measurements based on different

[3] *multisearch* package: https://github.com/fracpete/multisearch-weka-package

[4] External package outside of WEKA, see also http://wiki.pentaho.com/display/DATAMINING/MLPRegressor

[5] *timeseriesforecasting* package website: http://wiki.pentaho.com/display/DATAMINING/Time+Series+Analysis+and+Forecasting+with+Weka

training sets or the presentation of a certain achieved performance measure rate is of no avail to draw any conclusions. Rather than that, every time the same procedure was performed and as least as possible parameters were changed in each distinct test version. Table I presents all used metrics.

TABLE I
USED PERFORMANCE MEASURES [35]. $P = p_1, \ldots, p_t$ ARE PREDICTED VALUES; $A = a_1, \ldots, a_t$ ARE ACTUAL ONES.

Root mean-squared error	$RMSE$	=	$\sqrt{\frac{1}{n}\sum_{t=1}^{n}(p_t - a_t)^2}$
Root relative square error	$RRSE$	=	$\sqrt{\frac{\sum_{t=1}^{n}(p_t - a_t)^2}{\sum_{t=1}^{n}(\bar{a} - a_t)^2}}$
Mean absolute error	MAE	=	$\frac{1}{n}\sum_{t=1}^{n}\lvert(p_t - a_t)\rvert$
Relative absolute error	RAE	=	$\frac{\sum_{t=1}^{n}\lvert(p_t - a_t)\rvert}{\sum_{t=1}^{n}\lvert(a_t - \bar{a})\rvert}$
Standard deviation of P and A	S_{PA}	=	$\frac{\sum_{t=1}^{n}(p_t - \bar{p})(a_t - \bar{a})}{n-1}$
Standard deviation of P	S_P	=	$\frac{\sum_{t=1}^{n}(p_t - \bar{p})^2}{n-1}$
Standard deviation of A	S_A	=	$\frac{\sum_{t=1}^{n}(a_t - \bar{a})^2}{n-1}$
Correlation coefficient	CC	=	$\frac{S_{PA}}{\sqrt{S_P S_A}}$

C. Evaluation Results

In the following, we present the results of three experiments, each covering one of the evaluation questions *EQ1-EQ3*.

EQ1: Impact of Clustering on Accuracy

The clustering of time series data can be highly beneficial for improving predictive performance, especially when the data exhibits periodic-structures like seasons or trends. For example, the courses of the attribute traffic flow *Q* at a rush-hour on a Monday morning might be immensely different than on a Sunday night. In this experiment, the goal is to examine how the clustering of time series sub-sequences can influence the accuracy of the predictive models. Three different clustering rules were defined. R_1 denotes a clustering rule without any splitting, i.e., the whole data set is selected. R_2 shows a clustering rule splitting the data set into 20 clusters as visualized in Figure 3: Mo-Th, Fr, Sa, Su combined with morning rush hour, afternoon rush hour, and the times in between. R_3 splits each hour of a week into an own cluster, resulting in $7days * 24hours = 168$ clusters.

In this approach, we used AutoWEKA. Every error rate (cf. Table II) was calculated by using cross-validation with the amount of folds $k = 10$, hence, . The error rates were accumulated for each cluster and divided by the number of clusters in order to receive an averaged rate. The row *Instances* specifies the number of instances within a cluster on average.

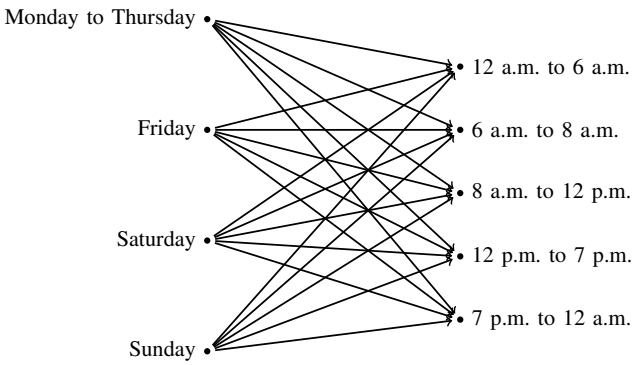

Fig. 3. 4 different day settings * 5 time slots = 20 clusters generated by rule R_2.

TABLE II
AVERAGED PERFORMANCE ERROR RATES OF THE TRAFFIC FLOW Q WITH THE THREE CLUSTERING RULES. BEST VALUES ARE WRITTEN IN BOLD.

R	**1**	**2**	**3**
Instances	31392	1569.6	186.86
MAE	220.47	197.35	**123.71**
RMSE	318.67	298.14	**171.28**
RAE in %	**14.48**	34.10	41.57
RRSE in %	**18.15**	40.36	43.18
CC	**0.9834**	0.89754	0.8296

In this experiment, the absolute error is pivotal since it intends to give an answer to the question of how large the training set should be in the case that different sizes might be considered. For that reason, the relative error rates have to be contemplated in relation to the amount of instances per cluster on average. To give an example, in many practical cases $0.4157 * 186.86 = 77.68$ respective to R_3 might be better than $0.1448 * 31392 = 4545.56$ for R_1. However, this assumption might differ for other experiment setups, e.g., when the amount of data is limited.

EQ2: Comparison of Optimization Approaches

In experiment 2, the performance of `SMOreg` – a *Support Vector Machine for Regression* algorithm – which is used in WEKA's default implementation for time series forecasting is compared with the (i) the ensemble learning based Stacking and (ii) AutoWEKA as representative of our CASHO algorithm. We used clustering rule R_2 from experiment 1 as illustrated in Figure 3. The test set comprises every last 4 time points available in the training set, i.e., the last hour. The future values for each hour were predicted based on the data from the hour frames of all weeks before. Stacking consisted of all the algorithms mentioned in Section IV-B. The meta-learner for ensemble learning was a linear regression model.

As shown in Table III, the AutoWEKA outperformed both other approaches. Stacking performs slightly worse than the default setup with `SMOreg`.

TABLE III
AVERAGED PERFORMANCE ERROR RATES OF THE TRAFFIC FLOW Q ACHIEVED BY THE DEFAULT CONFIGURATION AND THE OPTIMIZATION APPROACHES. S >5% DENOTES THE PERCENTAGE OF THE AVERAGED STANDARD DEVIATION HIGHER THAN 5%. BEST VALUES ARE WRITTEN IN BOLD.

Approach	SMOreg	AutoWEKA	Stacking
$S > 1\%$ in %	**96.25**	98.75	98.75
$S > 5\%$ in %	86.25	**85.00**	86.25
$S > 10\%$ in %	76.25	**71.25**	77.50
$S > 25\%$ in %	47.50	**45.00**	53.75
MAE	456.05	**445.33**	539.33
RMSE	694.38	**581.03**	683.24
RAE in %	8.24	**4.64**	13.01
RRSE in %	49.69	**41.58**	48.89
CC	0.89	**0.92**	0.88

EQ3: Ranking of Algorithms

In experiment 3, the frequency of the algorithms performing best while predicting the traffic flow was counted. The configuration consisted of the CASHO-based Multi Search approach with 168 clusters generated with clustering rule R_3. Figure 4 presents the comparison of the algorithms used in the Multi Search approach. The algorithms *decision table* (DT), *random forest* (RF), *random tree* (RT), and *support vector machines* (SVM) clearly outperformed the rest. Only *REP tree* (REPT) and *M5 tree* (M5P) contribute recognizable.

D. Discussion

The experiments prove the applicability and the performance using SATISFy. In the following, we discuss the results.

We applied different clustering rules in experiment 1 so that the resulting clusters vary significantly in their size for analyzing the impact of clustering on the performance. Whereas [43] do not see a benefit of automatic clustering of time series subsequences, our results indicate that clustering can support time series forecasting at least when domain knowledge is applied.

In experiment 2, we compare WEKA's default time series forecasting against two optimization approaches. The results indicate that the CASHO technique AutoWEKA outperforms the other approaches. The default algorithm SMOreg and Stacking perform similarly. Additionally, as CASHO compatible approaches – e.g., AutoWEKA – chooses the best algorithm out of a set of algorithms and optimizes its parameters, their results are better interpretable and reusable. This is an additional benefit of CASHO over ensemble learning.

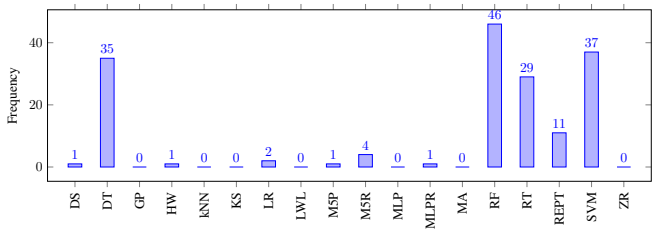

Fig. 4. The frequency of the best performing algorithms forecasting the traffic flow: Decision stump (DS), Decision table (DT), Gaussian processes (GP), Holt winters* (HW), K-nearest-neighbor (kNN), K-star (KS), Linear regression (LR), Locally weighted learning (LWL), M5 tree (M5P), M5 rules (M5R), Multilayer perceptron (MLP), Multilayer perceptron regression (MLPR), Moving average (MA), Random forest (RF), Random tree (RT), REP tree (REPT), Support vector machines (SVM), and ZeroR (ZR).

In the last experiment, we use Multi Search as it is a white-box optimization approach to compare several prediction algorithms. The results showed that a few algorithms dominated the results. Especially, Random Forest outperforms all other algorithms significantly. In total, 40% of the algorithms were not chosen once. For more than 27% of all clusters, one algorithm achieved clearly the best error rate. This might also explain why Stacking performs similarly to the default algorithm as it combines the small set of good algorithms with the larger set of worse performing algorithms.

The implementation's computation time does not constitute a major focus of this work but needs to be considered as an important limitation regarding the analyzer's practicability to a specific system. A probability distribution modeling a utility function of the hyper-parameter space is a mathematical way to assess the importance of each models' parameter option. This leads to a more systematic way to explore reasonable space configurations than a brute-force search as in Multi Search. AutoWEKA bases on a similar approach but does not offer explicit and accessible information about the utility distribution of each parameter space. Adjusting CASHO to support this is part of future work.

Another feature worthwhile to consider would be the implementation of a clustering algorithm in order to avoid the manual specification of clustering rules. However, in the conducted experiments, domain knowledge was applied by the definition of clustering rules. as the possible combinations of the two variables are manageable manually. For other use cases and for integrating more variables for clustering, WEKA also supports several implementations of clustering algorithms.

Lastly, the analyzer has to demonstrate its capability in continuous use, outside the system development environment in situations with widely varying data set properties. In this work, we performed several experiments, however, related to one use case only. Comprehensive testing in additional use cases taking noisy data into account and with a high variety of attributes is part of future work. This includes that the system needs to cope with forecasted situations where no adaptation knowledge is available. This must be supported by a learning approach for such situations. Additionally, self-adaptive system might have higher-dimensional system states, i.e. the targets. So far, we focused on one dimension only. The extension to higher-dimensional variables is future work.

VI. CONCLUSION

This paper presents SATISFy, a self-learning analyzer for time series forecasting. The analyzer can support developers at design time for finding the best algorithm for proactive analyzing. At runtime, it can be used within a self-improving module for changing the algorithm of the analyzer. The implementation is self-contained. A clearly defined interface

enables the integration into self-adaptive systems. Further, the system model enables the integration of several forecasting approaches simultaneously and support automatic clustering. We evaluate SATISFy in a use case for traffic prediction. The results indicate that our approach to self-learning can significantly improve the prediction performance within time series forecasting.

Finally, the approach also allows developers to find the best suitable algorithm at design time. For future work, we plan to add a reinforcement learning-based decision logic to adjust the analyzing algorithm at runtime. Therefore, the developer has to specify the optimization criteria as the evaluation has shown that different failure metrics can be optimized by different approaches for time series forecasting.

Acknowledgment

This work has been co-funded by the German Research Foundation (DFG) as part of project A4 within the Collaborative Research Center (CRC) 1053 – MAKI.

References

[1] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, no. Part B, pp. 184–206, 2015.

[3] C. Krupitzer, F. M. Roth, M. Pfannemüller, and C. Becker, "Comparison of Approaches for Self-Improvement in Self-Adaptive Systems," in *Proc. ICAC*, 2016, pp. 308–314.

[4] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Proc. SBIA*, 2004, pp. 286–295.

[5] N. K. Ahmed, A. F. Atiya, N. E. Gayar, and H. El-Shishiny, "An Empirical Comparison of Machine Learning Models for Time Series Forecasting," *Econometric Reviews*, vol. 29, no. 5-6, pp. 594–621, 2010.

[6] N. Esfahani and S. Malek, "Uncertainty in self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems II*, 2010, pp. 214–238.

[7] S.-W. Cheng and D. Garlan, "Handling uncertainty in autonomic systems," in *Proc. ASE*, 2007.

[8] A. Elkhodary, N. Esfahani, and S. Malek, "FUSION: A Framework for Engineering Self-tuning Self-adaptive Software Systems," in *Proc. FSE*, 2010, pp. 7–16.

[9] D. Narayanan and M. Satyanarayanan, "Predictive resource management for wearable computing," in *Proc. MobiSys*, 2003.

[10] D. Cooray, S. Malek, R. Roshandel, and D. Kilgore, "Resisting reliability degradation through proactive reconfiguration," in *Proc. ASE*, 2010, pp. 83–92.

[11] I. D. P. Anaya, V. Simko, J. Bourcier, N. Plouzeau, and J. Jézéquel, "A prediction-driven adaptation approach for self-adaptive sensor networks," in *Proc. SEAMS*, 2014, pp. 145–154.

[12] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White, "A Multi-Agent Systems Approach to Autonomic Computing," in *Proc. AAMAS – Vol. 1*, 2004, pp. 464–471.

[13] D. Fisch, E. Kalkowski, and B. Sick, "Collaborative Learning by Knowledge Exchange," in *Organic Computing – A Paradigm Shift for Complex Systems*. Springer, 2011, pp. 267–280.

[14] J. Dowling and V. Cahill, "Self-managed Decentralised Systems Using K-components and Collaborative Reinforcement Learning," in *Proc. WOSS*. ACM, 2004, pp. 39–43.

[15] C. Parra, D. Romero, S. Mosser, R. Rouvoy, L. Duchien, and L. Seinturier, "Using Constraint-based Optimization and Variability to Support Continuous Self-adaptation," in *Proc. SAC*, 2012, pp. 486–491.

[16] A. Pandey, G. A. Moreno, J. Camara, and D. Garlan, "Hybrid Planning for Decision Making in Self-Adaptive Systems," in *Proc. SASO*, 2016, pp. 130–139.

[17] H. Prothmann, F. Rochner, S. Tomforde, J. Branke, C. Müller-Schloer, and H. Schmeck, "Organic Control of Traffic Lights," in *Autonomic and Trusted Computing*. Springer, 2008, pp. 219–233.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[19] D. Kim and S. Park, "Reinforcement Learning-Based Dynamic Adaptation Planning Method for Architecture-based Self-Managed Software," in *Proc. SEAMS*, 2009, pp. 76–85.

[20] M. Caporuscio, M. D'Angelo, V. Grassi, and R. Mirandola, "Reinforcement Learning Techniques for Decentralized Self-adaptive Service Assembly," in *Proc. ESSOC*. Springer, 2016, pp. 53–68.

[21] T. Zhao, W. Zhang, H. Zhao, and Z. Jin, "A Reinforcement Learning-based Framework for the Generation and Evolution of Adaptation Rules," in *Proc. ICAC*. IEEE, 2017, pp. 103–112.

[22] J. Panerati, M. Triverio, M. Maggio, and M. D. Santambrogio, "On How to Coordinate the Behavior of Independent Adaptive Systems," in *Proc. Int. Work. Feed. Comput.*, 2012, pp. 1–6.

[23] T. Bäck and H.-P. Schwefel, "An Overview of Evolutionary Algorithms for Parameter Optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 1–23, 1993.

[24] M. Tanabe, K. Tei, Y. Fukazawa, and S. Honiden, "Learning environment model at runtime for self-adaptive systems," in *Proc. SAC*, 2017, pp. 1198–1204.

[25] W. Brockmann, N. Rosemann, and E. Maehle, "A Framework for Controlled Self-optimisation in Modular System Architectures," in *Organic Computing – A Paradigm Shift for Complex Systems*. Springer, 2011, pp. 281–294.

[26] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An Architecture-Based Approach to Self-Adaptive Software," *IEEE Intelligent Systems*, vol. 14, no. 3, pp. 54–62, 1999.

[27] D. A. Kafaf and D. K. Kim, "A web service-based approach for developing self-adaptive systems," *Comput. Electr. Eng.*, vol. 63, no. 1, pp. 260–276, 2017.

[28] M. Harman, E. Burke, J. A. Clark, and X. Yao, "Dynamic adaptive search based software engineering," in *Proc. ESEM*, 2012, pp. 1–8.

[29] D. Menasce, H. Gomaa, S. Malek, and J. P. Sousa, "SASSY: A Framework for Self-Architecting Service-Oriented Systems," *IEEE Software*, vol. 28, no. 6, pp. 78–85, 2011.

[30] A. J. Ramirez, D. B. Knoester, B. H. Cheng, and P. K. McKinley, "Applying Genetic Algorithms to Decision Making in Autonomic Computing Systems," in *Proc. ICAC*, 2009, pp. 97–106.

[31] L. Wang, "Using Search-Based Software Engineering to Handle the Changes with Uncertainties for Self-Adaptive Systems," in *Proc. ESEC/FSE*. ACM, 2017, pp. 1014–1017.

[32] P. Zoghi, M. Shtern, M. Litoiu, and H. Ghanbari, "Designing Adaptive Applications Deployed on Cloud Environments," *ACM TAAS*, vol. 10, no. 4, pp. 1–26, 2016.

[33] M. Salehie and L. Tahvildari, "Self-Adaptive Software: Landscape & Research Challenges," *ACM TAAS*, vol. 4, no. 2, p. Art. 14, 2009.

[34] S. Roberts, M. Osborne, M. Ebden, S. Reece, N. Gibson, and S. Aigrain, "Gaussian processes for time-series modelling," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 371, no. 1984, 2012.

[35] I. H. Witten, E. Frank, and M. Hall, *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2011.

[36] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[37] G. Holmes, A. Donkin, and I. H. Witten, "WEKA: a machine learning workbench," in *Proc. ANZIIS*, 1994, pp. 357–361.

[38] D. H. Wolpert, "Stacked Generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.

[39] L. Piplani, K. Gupta, and E. Goyal, "A Quantitative Comparison of Neural Networks and Logistic Regression," *International Journal of Advanced Engineering and Global Technology*, vol. 3, no. 11, 2015.

[40] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms," *CoRR*, vol. abs/1208.3719, 2012.

[41] M. Sommer, A. Stein, and J. Hhner, "Ensemble time series forecasting with xcsf," in *Proc. SASO*, 2016, pp. 150–151.

[42] J. Schlaich and M. Friedrich, "Staumeldungen und routenwahl in autobahnnetzen – teil 1: Analyse von staumeldungen," *Straßenverkehrstechnik*, vol. 14, no. 10, pp. 621–627, 2008.

[43] E. Keogh and J. Lin, "Clustering of time-series subsequences is meaningless: Implications for previous and future research," *Knowledge and Information Systems*, vol. 8, no. 2, pp. 154–177, 2005.