Mach Learn (2008) 70: 45–88
DOI 10.1007/s10994-007-5024-8

A formal framework and extensions for function approximation in learning classifier systems

Jan Drugowitsch · Alwyn M. Barry

Received: 6 March 2006 / Revised: 22 August 2007 / Accepted: 9 September 2007 /
Published online: 30 October 2007

Abstract Learning Classifier Systems (LCS) consist of the three components: function approximation, reinforcement learning, and classifier replacement. In this paper we formalize the function approximation part, by providing a clear problem definition, a formalization of the LCS function approximation architecture, and a definition of the function approximation aim. Additionally, we provide definitions of optimality and what conditions need to be fulfilled for a classifier to be optimal. As a demonstration of the usefulness of the framework, we derive commonly used algorithmic approaches that aim at reaching optimality from first principles, and introduce a new Kalman filter-based method that outperforms all currently implemented methods, in addition to providing further insight into the probabilistic basis of the localized model that a classifier provides. A global function approximation in LCS is achieved by combining the classifier's localized model, for which we provide a simplified approach when compared to current LCS, based on the Maximum Likelihood of a combination of all classifiers. The formalizations in this paper act as the foundation of a currently actively developed formal framework that includes all three LCS components, promising a better formal understanding of current LCS and the development of better LCS algorithms.

Keywords Learning classifier systems · Function approximation · Kalman filter

1 Introduction

Accuracy-based Learning Classifier Systems (LCS), which integrate function approximation, genetic algorithm driven search, and in many cases temporal difference learning, aim at the autonomous production of human-readable results that are the most compact

Editor: Risto Miikkulainen.

J. Drugowitsch (✉)
Department of Brain and Cognitive Sciences, University of Rochester, Rochester, NY, USA
e-mail: jdrugowitsch@bcs.rochester.edu

A.M. Barry
Department of Computer Science, University of Bath, Bath, UK
e-mail: A.M.Barry@bath.ac.uk

generalized representation whilst also maintaining high predictive accuracy. Alternatively, they can be seen as ensemble methods that evolve minimal, interpretable populations of predictors that have the advantage over conventional ensemble methods to dynamically modify number and location of the predictors to find a minimal but sufficient representation of the desired solution. Our previous work has demonstrated the competitiveness of this technique when applied to the task of data-mining (Saxon and Barry 2000; Greenyer 2000), work that has subsequently been extensively built upon (e.g. Bernardo et al. 2001, Dixon et al. 2002; Barry et al. 2004). Other researchers have provided further demonstration of its power in many single-step tasks, and have recently extended these results to the discovery of piecewise partially overlapping linear function approximations (Wilson 2002; Lanzi et al. 2005d). However, our work has demonstrated that current LCS approaches have only limited success in other than relatively trivial multi-step tasks (Barry 2002, 2003). These limitations have stimulated research to formulate partial models of the LCS (e.g. Bull 2002; Butz 2004; Wada et al. 2005). However, these developments have produced models of selective subcomponents that do not adequately capture the global dynamics of the components of the LCS (function approximation, temporal difference learning and classifier replacement) and neglect the effect of interactions between the components that can invalidate the local models. Nonetheless, they have contributed to our understanding of the rather complex dynamics of the genetic algorithm driven search in LCS.

In order to make progress in understanding the global behavior of LCS we aim to produce a framework that permits the study of LCS as a whole by the combination of interacting models of subcomponents. The primary objectives of this work are to (i) cover all LCS components within that framework to allow for systematic analysis of the components and their interaction, (ii) design the framework flexibly to utilize it in developing extensions to current LCS methods, and (iii) borrow notation and concepts from related fields to allow direct comparison and lower the barrier of translating new developments in other fields to LCS. Developing the framework was approached by splitting LCS into three components that relate to other Machine Learning techniques and are studied separately and in combination. Even though this paper only focuses on one of the components, this component is investigated with integration in mind. From constructing the framework we expect to be able to develop LCS that do not only feature enhanced performance, but are also based on theoretically justified approaches instead of the now commonly applied heuristics that require careful tuning of parameters.

1.1 Background and history

Learning Classifier Systems are rule-based machine learning systems that learn a set of classifiers with the aim of optimizing the reward in a reinforcement learning framework. This framework defines a single agent that interacts with an environment by sensing the environment's state and choosing an action depending on the current state. Based on the performed action and the current state, the agent receives external scalar reward which it aims to maximize in the long run. Throughout the years, systems have emerged that do not comply to this framework. However, for historical reasons we will describe them from the reinforcement learning perspective.

An LCS maintains a set of classifiers, each of which consists of a condition that promotes an action, and a set of performance measures. The condition *matches* one or several states of the environment. Upon observing a state, the LCS groups all matching classifiers in sets according to the action that they promote. Each of those sets results in a prediction of the expected return (as an accumulation of discounted future expected rewards) when performing the promoted action, calculated as a function of the performance measures of each of

the set's classifiers. Assuming a greedy policy, the action with the highest expected return is performed. Hence, in contrast to conventional reinforcement learning methods that provide a prediction of the expected return by using a single function approximation technique for all states, LCS assemble this prediction by combining the prediction of a set of localized models (that is, the classifiers), with the advantage and initial motivation of LCS of dynamically assigning resources to areas of the state space where the expected reward is harder to model.

A similar approach is followed when LCS are used for data mining. In this case, the classifier condition represents the set of attributes of the classification task that a certain classifier matches, and the action indicates the class that is predicted from those attributes. The external reward is now an indicator of the correctness of the promoted class, and the aim is to increase this correctness. Thereby, LCS again aims at dynamically relocating its classifiers to classes and sets of attributes that are harder to model.

Once external reward is received, it has to be distributed adequately over the classifiers that caused this reward (known as the *credit allocation scheme*). An early economically inspired approach was to let each successful classifier pay a bid taken from their performance measure into a bid pool that is then distributed to the performance measure of successful classifiers of the previous step. This popular temporal-difference based schema, known as the *Implicit Bucket Brigade* (Holland 1985; Riolo 1987a, 1987b), promotes in such a way classifiers that were indirectly responsible for receiving the reward. Gradually, the Bucket Brigade algorithm became discredited as a method of credit assignment due to slow transmission of rewards and consequent instability of the scene-setting classifiers (Forrest and Miller 1990). In the last decade, the relation of credit assignment in LCS to temporal-difference learning became more and more apparent, as demonstrated by the independent development of SARSA (Sutton 1996) in Wilson's ZCS (Wilson 1994) from the LCS side, and in On-Line Q-Learning (Rummery and Niranja 1994) from the reinforcement learning side. Finally, Watkin's Q-Learning (Watkins 1989) was explicitly adapted for credit assignment in Wilson's eXtended Classifier System (XCS) (Wilson 1995).

If the set of classifiers was constant with constant mixing weights, then LCS are subsumed by some methods in reinforcement learning with function approximation.[1] However, the aim of LCS is not only to maximize the long-term reward, but also to find a minimal set of rules to do so. For this task it employs a steady-state Genetic Algorithm (GA) that creates new classifiers and removes old ones periodically. The creation of new classifiers relies on well-performing classifiers in the current set, leading to the question of how to determine the performance of a classifier. Before XCS, most LCS (known as *strength-based* LCS) used the classifier's prediction of expected return as a measure of its performance, but that led to problems of the maintenance of low-reward classifiers, the lack of knowledge of the expected return of sub-optimal actions, and the emergence of classifiers that match overly large areas of the state space (Kovacs 2002). Additional problems due to particular LCS architectures were the emergence of parasitic classifiers (Smith 1994) and instabilities due to stochastic action selection (Compiani et al. 1990; Forrest and Miller 1990).

With XCS, Wilson used the classifier's accuracy of predicting the expected return as its measure of performance, where the accuracy is some inverse function of the expected prediction error (Wilson 1995). This type of LCS is now commonly known as an *accuracy-based*

[1] Having a constant set of classifiers with constant mixing weights makes LCS equivalent to a linear function approximation architecture. As such, it can be treated as reinforcement learning with linear function approximation, which has already been investigated at length (e.g. Tsitsiklis and Van Roy 1997; Nedić and Bertsekas 2003).

LCS. As one would expect, it resolved the problems of strength-based LCS by building a complete and accurate prediction map over the action/state space. Due to its niche GA architecture, it promotes general classifiers over specific classifiers, but only up to a certain implicitly specified level to maintain a certain generality/accuracy tradeoff (Butz 2001). In combination, this has been demonstrated in a range of problems to identify an optimally general set of (mostly) disjoint classifiers that predict the expected return for all state/action combinations.[2]

Accuracy-based LCS also lend themselves for use as adaptive function approximators (Wilson 2001): The action of the classifier can be removed, and the condition matches part of the domain for the target function. Instead of state and reward the classifiers are given the function argument as state and the function value as reward. Initially the classifiers were only able to approximate one value over their subset of the domain, but soon thereafter their approximation abilities were extended to approximate first-order (Wilson 2002; Lanzi 2005a), and second- and third-order (Lanzi 2005a) polynomials. In addition, the algorithms for function approximation were extended from the initially used Least Mean Square algorithm to the Normalized Least Mean Square algorithm (Wilson 2002) and the Recursive Least Squares algorithm (Lanzi et al. 2005d). Their approach, however, was ad-hoc rather than introducing new algorithms in a bottom-up approach from first principles.

Currently there are no strictly formal performance guarantees for XCS, although strong arguments have been made for its ability to avoid the emergence of overly general classifiers (Kovacs 2002), and that it is a PAC-learner for k-DNF functions (Butz 2004). Even these only hold over single-step tasks, and applications in multi-step tasks still only produce limited success. As previously mentioned, it is the lack of formal models of the interacting components of LCS to analyse these problems that is the motivation for our work.

1.2 Generalized function approximation in learning classifier systems

With the introduction of function approximation as an application domain, it became clear that adaptively approximating functions of some kind forms the core of all modern LCS. Let us consider a function that maps some domain that we will call the *state space* into the set of real numbers. Table 1 summarizes the form of this state space for the different LCS tasks, and indicates whether the function is stationary or non-stationary. When performing straightforward function approximation it is clear that we will use the function domain as our state space. For data mining the state space is formed by a combination of the set of attribute values and the set of classes, and we aim at learning the *correctness* of certain attribute/class combinations. Similarly, single-step tasks are dealt with by learning the reward that is given

Table 1 The different LCS application domains and the according state space over which to perform the function approximation

Task	State space	Stationary
Function approximation	function domain	Yes
Data mining	attributes × classes	Yes
Single-step tasks	features × actions	Yes
Multi-step tasks	features × actions	No

[2]For an introductory overview of LCS see (Bull 2004), and for further detail on XCS see (Butz 2004, 2006).

for certain state/action combinations, where the states are identified by their features. This leads to the state space being formed by the set of feature values and possible actions. The same state space applies to multi-step tasks, but in their case we are interested in learning the expected return rather than the reward. As the expected return is given through some form of reinforcement learning or dynamic programming algorithm, the function values are not explicitly available, which makes multi-step tasks the only ones where the function to be approximated varies over time.

Therefore, LCS acts as an adaptive function approximation architecture, where the definition of function domain and range depends on the problem at hand. Each classifier matches only a subset of the function domain, where that subset is given by the classifier condition for straight-forward function approximation tasks, and by the combination of condition and action for all other tasks. We can thus deal only with function approximation, because all the other cases can be reduced to this case.

While the power of LCS comes from dynamically adapting its number of classifiers and their localization in the state space, we first need to understand in detail how a static set of classifiers is able to act as a function approximator, before considering how classifiers can be added, removed, or relocated to improve the quality of the function approximation. Thus, the investigations in this paper are restricted to such a static set of classifiers, each of which provides a local function approximation over a subspace of the state space. The global function approximation is provided by a weighted combination of the local models. Our aim is to describe a formalism that allows us to define what it means for the task of local function approximation to be solved optimally, and to analyse and extend current algorithmic approaches.

The next logical step, which we consider as a separate but closely linked task to function approximation by a static set of classifiers, is how classifiers can be replaced to improve the quality of this function approximation, leading to *adaptive* function approximation. Considering multi-step tasks as the prime motivator for the development of LCS, the analysis of how the adaptive function approximation of LCS is combined with reinforcement learning is the last challenge in providing a formal framework for all three components of LCS, that have now clearly identified as being: function approximation (for a fixed set of classifiers), classifier replacement, and reinforcement learning.

To summarize, we concentrate here on the function approximation component of LCS. To be able to relate our investigations to existing literature in function approximation and adaptive filter theory, we will adopt the following basic assumptions:

- The set of classifiers is fixed. This is equivalent to keeping the set of states of the state space that a classifier matches time-invariant.
- The states are observed according to a time-invariant probability distribution. This is usually the case in the long run.
- The value function is time-invariant (i.e. stationary). As shown in Table 1 this assumption is fulfilled in all tasks but multi-step tasks. As the function in multi-step tasks is a product of the reinforcement learning algorithm, their treatment requires analysis of the interaction between reinforcement learning and function approximation. Such analysis is beyond the scope of this paper, but the reader is referred to our further work in (Drugowitsch and Barry 2006b).

Some of the assumptions will have to be removed once the interaction between different components of LCS is studied. However, even when only considered in combination with the replacement of classifiers, it can stand on its own as a method of adaptive function approximation (e.g. Wilson 2001) and for use in data-mining (e.g. Bernardo et al. 2001).

The long-term outcomes of this work are potentially wide, but immediate achievements can be stated as

- providing the first part of the framework for the theoretical analysis of LCS,
- providing a sound theoretical foundation in which to evaluate other work within function approximation in LCS,
- enabling the development and integration of new and more powerful algorithms for function approximation in a controlled and systematic manner,
- bridging the gap between the current work in function approximation within reinforcement learning to equivalent work within LCS,
- enabling the development of a more formal linkage between function approximation, reinforcement learning, and classifier replacement,
- moving the whole LCS community to a much more formal approach to development and evaluation of the LCS method.

1.3 Structure

The function approximation-part of the framework is introduced in Sect. 2, together with the aim of function approximation in LCS. Its notation is based on Bertsekas and Tsitsiklis' work that investigates the interaction between reinforcement learning and function approximation (Bertsekas and Tsitsiklis 1996), and on Sutton and Barto's introductory text (Sutton and Barto 1998). As such, it should be easily accessible for anyone working in that field. First, the use of that framework is demonstrated in Sect. 2.3, where we discuss partial matching and the likely consequences of an alternative approximation objective. Appendix A summarizes the notation and identifies in which section the symbols used appear for the first time.

Section 3 uses the framework to consider single classifiers and what it means for them to have an optimal approximation. This section is particularly important as it not only states the definite approximation goal for single classifiers, but also gives the conditions under which this optimality is reached. This is done for the case when the full function is known, and for the sample-based case which aims at updating the approximation with every additional observation of a function value.

Section 4 follows as a logical consequence of Sect. 3, discussing different gradient-based algorithms that aim at implementing a sample-based approach towards optimality as outlined before. As a further demonstration of the usefulness of the framework, in Sect. 5 we utilize it to develop a new and potentially superior algorithmic approach that is based on the Kalman filter (Kalman 1960), and relates it to the Recursive Least Squares approach. A short experimental section follows that demonstrates this superiority.

So far single classifiers have been considered. In Sect. 7 we discuss how to best integrate the approximations of the classifiers into one approximation over the whole state space. This method is based on modeling the prediction of a classifier by a normal distribution and using the Maximum Likelihood Estimate to calculate the most likely overall value. As part of this investigation we examine the mixing parameter and give recommendations on how to set it.

Throughout this paper we give several examples that relate our developments to currently used LCS architectures. These examples show on one hand how the framework matches these architectures, and on the other hand gives new insights into currently used methods.

2 The framework

In this section we define the framework which will be used to study how Learning Classifier Systems use their classifiers to approximate functions. We will first give a general problem formulation, followed by the overall aim, and how this aim is approached by Learning Classifier Systems. In addition, we will present a short discussion on (i) partial rather than exclusive (binary) matching and, (ii) why LCS function approximation architectures that do *not* emphasize the independence of classifiers w.r.t. approximation might cause problems when used in combination with classifier replacement.

2.1 Problem formulation and aim

Let $V : S \to \mathbb{R}$ be the function we want to approximate.[3] The function's domain S, called the *state space*, is either a countable set or an uncountable set, which we will map onto $\mathbb{N}$ or $\mathbb{R}$ respectively.[4]

The function V is not directly observable but will be sampled or observed with a given time-invariant sampling distribution. Let $\pi : S \to [0, 1]$ define the sampling distribution,[5] giving the probability of sampling state i by $\pi(i)$. The function is sampled in discrete time-steps $t = 0, 1, \ldots$, giving the sequence of states $\{i_0, i_1, \ldots\}$, the sequence of function values $\{V(i_0), V(i_1), \ldots\}$, and the sequence of approximations $\{\tilde{V}_0, \tilde{V}_1, \ldots\}$, where $\tilde{V}_t : S \to \mathbb{R}$ is the approximation after observing $V(i_t)$.[6]

In addition to observing the sequence of function values, we simultaneously observe a set of features of the current state. These features are formed through a set of L basis functions $\{\phi_l : S \to \mathbb{R}\}_{l=1,\ldots,L}$, where function $\phi_l(i)$ returns the lth feature of state i. In combination, they form the *feature vector* $\phi : S \to \mathbb{R}^L$, returning the features of state i as a vector $\phi(i) = (\phi_1(i), \ldots, \phi_L(i))'$. Note that this, as well as all other vectors in this paper, is a column vector, unless otherwise stated, as indicated by the transpose $\cdot'$.

The selection of features is either left to the designer of the system or depends on the set of sensors that are available. Even though the quality of the approximation is highly dependent on the available features, we will not discuss good heuristics for selecting those features, and the interested reader is referred to standard literature about linear function approximation architectures, and the large body of existing work on feature selection.

The aim of the function approximator is to minimize the mean-squared error (MSE) given by

$$\int_S \pi(i)(V(i) - \tilde{V}(i))^2 \, \mathrm{d}i,$$

[3] The symbol V is used with foresight to value function approximation in multi-step tasks.

[4] Please note that all of the following is appropriate for the case of a countable state space, but might require some addition technical conditions for an uncountable state space, which are omitted to ease readability.

[5] If the dynamics of the sampling is determined by a Markov Chain, as is the case for application in multi-step tasks, this sampling distribution can be thought of as the steady-state distribution of the Markov Chain. Hence, with a change of policy the steady-state distribution will also change, which can lead to an oscillation in the selection of policies (Koller and Parr 2000). This is a known but unsolved problem in reinforcement learning with function approximation.

[6] In general, any symbol with subscript $\cdot_t$ indicates the time dependency of its meaning and refers to its state after observing the properties of state i_t.

where $\tilde{V} : S \to \mathbb{R}$ is the approximation of V. For a countable set S the integral can be replaced by a sum. The squared error is weighted by the sampling distribution, as it emphasizes the error of states that are visited most frequently, and as it naturally represents the approximation target of most standard approximation algorithms.

2.2 LCS function approximation architecture

2.2.1 A single classifier

An LCS utilizes a finite set of K classifiers to approximate the function V. We will enumerate the classifiers with $1, \ldots, K$, and denote a classifier parameter of classifier k by the subscript $\cdot_k$.

Each classifier k *matches* a particular subset $S_k \subseteq S$ of the state space S, which we will call the *matched states set*.[7] As discussed in Sect. 1.2, the matched states set of a classifier is given by its condition and its action. The aim of a classifier is to approximate the function V over the classifier's matched states set S_k. To ease notation, we will utilize the indicator function $I_{S_k} : S \to \{0, 1\}$, which is defined as

$$I_{S_k}(i) = \begin{cases} 1 & \text{if } i \in S_k, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Hence, $I_{S_k}(i)$ returns 1 only if classifier k matches state i.

To avoid introducing additional complexity, we will restrict ourselves to linear approximation architectures. It is possible to use LCS with non-linear approximation, but this might introduce local optima (see e.g. Haykin 1999) and will significantly complicate analysis. Linear architectures are characterized by state-independent approximation parameters and their linear relation to the features of the states. The approximation parameters are the adaptable values that modify the shape of the approximation, and are given by the parameter vector $w_k \in \mathbb{R}^L$, also called the *weight vector*, of classifier k. For any matched state i, the dot product of the classifier's weight vector and the feature vector $\phi(i)$ of that state gives the approximation $\tilde{V}_k(i)$ by classifier k, that is

$$\tilde{V}_k(i) = \sum_{l=0}^{L} w_k(l)\phi_l(i) = w_k'\phi(i). \tag{2}$$

2.2.2 Approximation goal of a single classifier

Our approximation goal is for each classifier k to minimize the mean-squared error $f_k : \mathbb{R}^L \to \mathbb{R}$ over its matched states set S_k; that is, we want to minimize

$$\begin{aligned} f_k(w_k) &= \frac{1}{\int_{S_k} \pi(i)\,\mathrm{d}i} \int_{S_k} \pi(i)(V(i) - \tilde{V}_k(i))^2\,\mathrm{d}i \\ &= \mathbb{E}_k((V - \tilde{V}_k)^2), \end{aligned} \tag{3}$$

where $\mathbb{E}$ is the expectation operator. The whole term is scaled by the inverse of the probability of observing any of the matching states to make the errors comparable between

[7] In LCS nomenclature the *match set* refers to the set of classifiers that match a particular state. For this reason we call S_k the matched states set rather than the match set.

different classifiers. To ease notation, we define the classifier-centric expectation $\mathbb{E}_k$ by $\mathbb{E}_k(X) = \mathbb{E}(I_{S_k})^{-1}\mathbb{E}(I_{S_k}X)$, giving the normalized expectation over the states that classifier k matches. As before, for a countable set S the integral of (3) can be replaced by a sum.

As the function is usually not fully observable, the mean-squared error cannot be evaluated when performing sample-based approximation. Therefore, we will define an approximation $\varepsilon_{k,t}$ to the mean-squared error f_k at time t, which is given by

$$\varepsilon_{k,t} = \frac{1}{c_{k,t} - L} \sum_{m=0}^{t} I_{S_k}(i_m)(V(i_m) - w'_{k,t}\phi(i_m))^2, \tag{4}$$

where $c_{k,t}$ is the *match count*[8] of classifier k at time t, giving the number of observed states until time t that are in the matched states set S_k; that is $c_{k,t} = \sum_{m=0}^{t} I_{S_k}(i_m)$. The L is subtracted in the denominator to keep the error estimate unbiased.

Note that the sequence of states $\{i_m\}$ is determined by the sampling distribution π, which will automatically introduce a weighting by this distribution in (4), as is required to approximate (3). In general, it is known that the mean-squared error approximation $\varepsilon_{k,t}$ is an unbiased estimate of the mean-squared error f_k (see e.g. Graybill 1961, Chap. 6.2). This has the consequence that if we are using an algorithm that minimizes $\varepsilon_{k,t}$, then we can guarantee convergence of the classifier error to the mean-squared error f_k with probability 1 (Eweda and Macchi 1987). We will come back to this concept after having introduced Kalman filter classifiers in Sect. 5.

A feature of LCS is to track the utility of a classifier at the same time as performing the approximation. For accuracy-based classifiers, the utility is defined as some inverse of the approximation of the mean-squared error $\varepsilon_{k,t}$. Thus, when discussing possible algorithms for classifier parameter approximation in LCS, we need to consider tracking the approximation error in addition to providing a useful approximation.

Example 1 (Common linear approximation architectures in LCS) As also employed in Wilson's original XCS (Wilson 1995), the most commonly used feature choice in LCS is $\phi(i) = 1$ for all $i \in S$. As the classifier weight vector w_k has the same number of elements as the feature vector, we only have one element $w_k(1)$, which directly represents the approximated function value $V_k(i) = w_k(1)$, for all $i \in S_k$. This scalar is called the *prediction* in LCS jargon. In minimizing the mean-squared error f_k (see (3)), the classifier weight averages the function values over the states that the classifier matches, weighted by the sampling distribution. For that reason we will call such classifiers *averaging classifiers*.

Recently, the power of LCS w.r.t. predicting continuous functions was increased by introducing more complex feature vectors, such as in (Wilson 2002, 2004; Lanzi 2005a). Initially, a feature vector $\phi(i) = (1, i)'$ for all $i \in S$ was used. This allows a classifier to approximate straight lines by $V_k(i) = w_k(1) + i\,w_k(2)$, where $w_k(1)$ determines the bias and $w_k(2)$ gives the slope. Lanzi et al. (2005a) extended this concept by using feature vectors $\phi(i) = (1, i, i^2)'$ and $\phi(i) = (1, i, i^2, i^3)'$ to represent quadratic and cubic functions. Naturally, the choice of feature vectors is not restricted to nth-order polynomials, but can also include radial-basis functions (RBFs) or any other function, as long as they can be formulated as defined by (2).

[8] In traditional LCS jargon, the match count is called the *experience* of a classifier.

2.2.3 Mixing classifiers

As one classifier possibly only matches a subset of the state space, i.e. $S_k \subset S$, we need to integrate the approximation of all classifiers to recover an approximation over the whole domain of V. Let $\psi_k : S \to [0, 1]$ be the mixing weight of classifier k for state i, where $\sum_{k=1}^{K} \psi_k(i) = 1$ for any $i \in S$, and $\psi_k(i) = 0$ for all $i \notin S_k$. We can then form the overall approximation $\tilde{V}$ of V by

$$\tilde{V}(i) = \sum_{k=1}^{K} \psi_k(i) \tilde{V}_k(i) = \sum_{k=1}^{K} \psi_k(i) w_k' \phi(i). \tag{5}$$

As all mixing weights for one state sum up to 1, the overall approximation is always bounded by the highest and lowest approximation for that state. Additionally, classifiers that do not match state i do not contribute to the approximation of that state, due to the condition $\psi_k(i) = 0$ for all $i \notin S_k$. When we come to examine specific algorithms in Sects. 4 and 5, we will only deal with a single classifier, but we will discuss mixing strategies again in Sect. 7.

2.2.4 Matrix notation and approximation as projection

For countable state spaces it is possible to define the values of V by a vector of the size of the state space. Subsequently we can create a feature matrix and define the approximation $\tilde{V}_k$ of a classifier as a linear operation on that feature matrix. That allows us to define the approximation of a classifier as a projection of the value vector into the classifier's approximation space. This is particularly useful when analysing LCS function approximation for use in reinforcement learning, as described in (Drugowitsch and Barry 2006b). As we will neither use it for the discussion of the different algorithms, nor for classifier mixing, we will not introduce it here. However, the interested reader is referred to (Drugowitsch and Barry 2006a) for more information.

2.3 Further issues

Even though not immediately part of our framework, the two issues presented here show both the generality and the limits of the framework. On one hand we discuss an extension where classifiers can match certain states to a degree rather than in a simple binary fashion, and its feasibility and consequences. On the other hand, we compare the architecture of our framework to another architecture that is in use, and give arguments why we have decided for one rather than the other.

2.3.1 Non-binary matching

Non-binary matching means using a matching-representing function (in our case I_{S_k}) that returns values of range [0, 1], allowing for matching to a degree. Such matching was already experimented with in strength-based LCS, and is also proposed for accuracy-based LCS (e.g. Butz 2005). Nevertheless, to our knowledge there exists no proper comparison between binary and non-binary matching. Butz only mentions that "Preliminary experiments in that respect [...] did not yield any further improvement in performance" (Butz 2005).

By looking at (3) we can see that the indicator function I_{S_k} acts as a multiplier to the sampling distribution and can hence be interpreted as a modification of it. From the point-of-view of a classifier, there is no difference between not matching and not visiting a state,

as $I_{S_k}(i)\pi(i) = 0$ for all $i \notin S_k$. Equally, states that are matched are visited according to their sampling distribution. Note that for $I_{S_k}(i)\pi(i)$ to be the probabilities of a classifier-centric state distribution, we need to normalize it by the constant $\mathbb{E}(I_{S_k})^{-1}$, as it otherwise might violate $\int_{S_k} \pi(i)\,\mathrm{d}i = 1$.

When applying the above interpretation to non-binary matching, I_{S_k} can take values between 0 and 1 in addition to $\{0, 1\}$. Hence, rather than just choosing which states are observed and which are not, a reduced matching value has the effect of reducing the sampling probability of that state. Thus, when minimizing (3) the approximation error of states with a lower matching value has reduced influence on our minimization goal. As classifiers still approximate independently, we have no reason to believe that non-binary matching might not work, as long as a lower degree of matching is also considered in the algorithmic approximation of the mean-squared error after (4). However, due to the lack of experimental evidence, and to keep further analysis simple, we will only consider the case of binary matching, as given by (1).

2.3.2 Arguments against aggregating classifier approximations

Recently, Wada et al. (2005) have introduced a modified LCS, similar to Kovacs' SB-XCS (Kovacs 2002) and related to Wilson's ZCS (Wilson 1994), that removes the independence of function approximation between different classifiers. In our current framework, each classifier aims at finding the best approximation of its matching part of the function V, independent of the approximation of other classifiers. Another approach is to not minimize the error of each classifier separately, as given by (3), but to minimize the error of the overall approximation, that is to minimize

$$\int_S \pi(i)\left(V(i) - \sum_{k=1}^{K} I_{S_k}(i) w_k' \phi(i)\right)^2 \mathrm{d}i.$$

In that case, the function is approximated by

$$\tilde{V}(i) = \sum_{k=1}^{K} I_{S_k}(i) w_k' \phi(i),$$

without considering different mixing weights for different classifiers, in contrast to (5). Specifically, the approximation of the different classifiers is aggregated rather than averaged. Such a measure linearizes classifier mixing and eases analysis by relating it to reinforcement learning with linear function approximation.

Let us now consider the consequences of replacing classifiers in the population.[9] In the case of aggregating the classifier approximations, removing one classifier from the population will also remove its contribution to the approximation of all the states that it matched. Therefore, the approximation for all those states will change, resulting in a deviation from the minimal error. Hence, caused by the aggregated approximation, the classifiers matching those states need to correct their approximations to again return to the optimal approximation for the new population. The necessary change might be significant and can cause severe

[9]Even though we currently only consider constant populations, the analysis up to this point has been done with classifier replacement in mind.

temporary performance drops.[10] Additionally, estimating the quality of a single classifier becomes harder, as the error of an approximation cannot be assigned to a single classifier but only to all classifiers that match that state. As a result, it becomes hard for accuracy-based LCS with aggregating classifiers to judge the quality of a classifier.

Using averaging rather than aggregation removes the interdependence between the different classifiers. Replacing classifiers in the population does not influence the optimal approximation of other classifiers and allows for more stability in the overall approximation. The error of a single classifier can be easily judged by the mean-squared error over its matching states, as given by (3).

For all the above reasons we believe that aggregating classifier approximations will introduce more problems than it solves. An analysis that uses such an architecture as its basis deviates too much from the spirit of current LCS to be of much support for future development. Therefore, we will only concentrate on the case of averaged classifier approximation, as outlined in the description of our framework.

3 Optimality

This section deals with what it means for an approximation performed by a single classifier to be *optimal*. Having knowledge of such an optimality measure, we can then compare different algorithms that aim to reach an optimal approximation.

We will discuss optimality on one hand for the case where we have full knowledge of the function V, and on the other hand for sample-based approximation. Both cases are defined by minimizing their corresponding mean-squared errors, which is a convex function and therefore has a unique minimum. Additionally, we will give expressions for the mean-squared error when it is minimized, as the classifier's error when performing optimal approximation gives the minimal error that a classifier can achieve and is therefore a good indicator of its quality.

3.1 Optimality given full knowledge of the function

Using the definition of the mean-squared error given by (3), and the linear approximation architecture of classifiers, as given by (2), we can state the following:

Theorem 1 *Given that the function $V : S \to \mathbb{R}$ is fully known, the approximation of classifier k is optimal (i.e. the approximation error is minimal) if the features are orthogonal to the approximation error, that is if*

$$\int_{S_k} \pi(i)\phi(i)(V(i) - w_k'\phi(i))\,\mathrm{d}i = 0$$

holds, which can also be written as $\mathbb{E}_k(\phi\phi')w_k = \mathbb{E}_k(V\phi)$.

In that case, the mean-squared error of that classifier is given by

$$\frac{1}{\int_{S_k} \pi(i)\,\mathrm{d}i}\left(\int_{S_k} \pi(i)V(i)^2\,\mathrm{d}i - \int_{S_k} \pi(i)\tilde{V}_k(i)^2\,\mathrm{d}i\right),$$

[10] The degraded performance of the described function approximation architecture was demonstrated empirically in (Wada et al. 2004), although Booker (2006) has suggested that high-error classifiers in such an architecture only contribute marginally to the overall approximation, which makes their removal less disruptive.

which is the minimal error that can be achieved with that classifier, and can also be written as $\mathbb{E}_k(V^2) - \mathbb{E}_k(\tilde{V}_k^2)$.

Proof Combining (3) and (2), and omitting the constant term $\mathbb{E}(I_{S_k})^{-1}$, gives

$$\int_{S_k} \pi(i)(V(i) - w_k'\phi(i))^2 \,\mathrm{d}i$$

as the minimization goal for classifier k. The optimality condition can be found by taking the first derivative of the above w.r.t. w_k and setting it to zero. Given that a minimum exists, the resulting minimum is unique, as the above is a convex function of w_k. Its form $\mathbb{E}_k(\phi\phi')w_k = \mathbb{E}_k(V\phi)$ is found by simple algebraic manipulation of the optimality condition multiplied by $\mathbb{E}(I_{S_k})^{-1}$ on both sides.

The simplified error term can be derived by using $V(i) = w_k'\phi(i) + (V(i) - w_k'\phi(i))$ and the optimality condition to get

$$\begin{aligned}\int_{S_k} \pi(i)V(i)^2 &= \int_{S_k} \pi(i)(w_k'\phi(i))^2 \,\mathrm{d}i \\ &\quad + 2w_k' \int_{S_k} \pi(i)\phi(i)(V(i) - w_k'\phi(i)) \,\mathrm{d}i \\ &\quad + \int_{S_k} \pi(i)(V(i) - w_k'\phi(i))^2 \,\mathrm{d}i \\ &= \int_{S_k} \pi(i)\tilde{V}_k(i)^2 \,\mathrm{d}i + \int_{S_k} \pi(i)(V(i) - \tilde{V}_k(i))^2 \,\mathrm{d}i.\end{aligned}$$

The second integral of the right-hand side is the mean-squared error scaled by $\mathbb{E}(I_{S_k})$, which leads straight to the minimal error expression. □

To attach a more intuitive understanding, we will call the matrix $\mathbb{E}_k(\phi\phi')$ the feature vector correlation matrix, and $\mathbb{E}_k(V\phi)$ the function-feature correlation matrix.[11]

3.2 Optimality by sampling

Usually, we cannot observe all of the function V at once. Instead, a sequence of observations $\{V(i_0), V(i_1), \ldots\}$ is used to perform its approximation. Hence, rather than minimizing the mean-squared error f_k as given by (3), we want to minimize its approximation $\varepsilon_{k,t}$ (see (4)). The optimality condition of the result is well known as the *Principle of Orthogonality* (see e.g. Haykin 2002):

Theorem 2 (Principle of orthogonality) *The approximation of function V by classifier k is optimal* (*i.e. the approximation error is minimal*) *if the sequence of feature vectors $\{\phi(i_0), \phi(i_1), \ldots\}$ is orthogonal to the sequence of approximation errors for the matching*

[11] Technically, calling these terms *correlation matrices* is a slight misuse of the term, as they might be shifted by a certain bias. Still, as they express some form of correlation within the pairs $\phi\phi'$ and $V\phi$, we feel that calling them correlation matrices is appropriate.

states $i_m \in S_k$. That is, at time t,

$$\sum_{m=0}^{t} I_{S_k}(i_m)\phi(i_m)(V(i_m) - \tilde{V}_{k,t}(i_m)) = 0 \tag{6}$$

has to hold, where $\tilde{V}_{k,t}(i_m) = w'_{k,t}\phi(i_m)$, and $w_{k,t}$ is the classifier's weight vector at time t.

In that case, the approximated mean-squared error of classifier k is

$$\varepsilon_{k,t} = \frac{1}{c_{k,t} - L} \sum_{m=0}^{t} I_{S_k}(i_m)(V(i_m)^2 - \tilde{V}_{k,t}(i_m)^2),$$

which is the minimum approximation error at time t.

Proof Both the optimality condition and the minimal approximation error are derived using the same approach as in Theorem 1, but this time minimizing (4) rather than (3). For more details see (Drugowitsch and Barry 2006a). □

Using this principle, we can pre-multiply the optimality condition from the Principle of Orthogonality by $w'_{k,t}$ and replace $\tilde{V}_{k,t}(i_m)$ by $w'_{k,t}\phi(i_m)$, to get

$$\sum_{m=0}^{t} I_{S_k}(i_m) w'_{k,t}\phi(i_m)(V(i_m) - w'_{k,t}\phi(i_m)) = 0.$$

Together with (2) that gives:

Corollary 1 (Corollary to the principle of orthogonality) *The approximation of function V by classifier k is optimal (i.e. the approximation error is minimal) if the sequence of the approximations $\{\tilde{V}_{k,t}(i_0), \tilde{V}_{k,t}(i_1), \ldots\}$ is orthogonal to the sequence of approximation errors for the matching states $i_m \in S_k$. That is, at time t*

$$\sum_{m=0}^{t} I_{S_k}(i_m)\tilde{V}_{k,t}(i_m)(V(i_m) - w'_{k,t}\phi(i_m)) = 0$$

has to hold, where $w_{k,t}$ is the classifier's weight vector at time t, and $\tilde{V}_{k,t}$ is its approximation, given by $\tilde{V}_{k,t}(i) = w'_{k,t}\phi(i)$ for state i.

Hence, when we have an optimal approximation, both the sequence of features $\{\phi(i_m)\}_{m=0}^{t}$ and the sequence of approximations $\{\tilde{V}_{k,t}(i_m)\}_{m=0}^{t}$ are orthogonal to the sequence of approximation errors $\{V(i_m) - \tilde{V}_{k,t}(i_m)\}_{m=0}^{t}$ for the states that classifier k matches. Orthogonality is given because their dot product is zero. As the real function values V are the vector sum of their approximations $\tilde{V}_k$ and the errors of that approximation, the optimal approximation of V is an orthogonal projection of the function into the plane that holds the optimal approximation. This plane coincides with the plane that is spanned by the feature vectors, as both are orthogonal to the approximation error. Hence, the approximation that minimizes the mean-squared error is an orthogonal projection of the function into the feature space for a particular classifier, a well-known result from linear algebra (e.g. Williams 2005).

Example 2 (Optimal approximation for averaging classifiers) Let us consider an averaging classifier k, with a feature vector of $\phi(i) = (1)$ for all $i \in S$, in which case by (2), $\tilde{V}_k(i) = w_k(1)$ for all $i \in S_k$. Then, by solving Theorem 1 for w_k, the optimal approximation given a known function V is

$$w_k(1) = \mathbb{E}_k(\phi\phi')^{-1}\mathbb{E}_k(V\phi) = \mathbb{E}_k(V),$$

which is the normalized expectation over function V for all matched states. In that case, the mean-squared error is

$$\begin{aligned}\mathbb{E}_k(V^2) - \mathbb{E}_k(\tilde{V}_k^2) &= \mathbb{E}_k(V^2) - \mathbb{E}_k(\mathbb{E}_k(V)^2)\\ &= \mathbb{E}_k(V^2) - \mathbb{E}_k(V)^2\\ &= \mathrm{var}_k(V),\end{aligned}$$

where var_k stands for the normalized variance over all matched states, that is $\mathrm{var}_k(X) = \mathbb{E}(I_{S_k})^{-1}\mathrm{var}(I_{S_k}X)$. This demonstrates that the mean-squared error is the normalized variance of the function values over the matched states.

For the sample-based approach, let us consider the state sequence $\{i_0, \ldots, i_t\}$ at time t. Then, if we substitute (2) in Theorem 2 and solve for $w_{k,t}$, the optimal approximation at time t is

$$\begin{aligned}w_{k,t}(1) &= \left(\sum_{m=0}^{t} I_{S_k}(i_m)\phi(i_m)\phi(i_m)'\right)^{-1}\sum_{m=0}^{t} I_{S_k}(i_m)V(i_m)\phi(i_m)\\ &= \frac{1}{c_{k,t}}\sum_{m=0}^{t} I_{S_k}(i_m)V(i_m),\end{aligned}$$

which is the average of the function values of all matched states in the state sequence. The approximated mean-squared error becomes

$$\varepsilon_{k,t} = \frac{1}{c_{k,t} - L}\sum_{m=0}^{t} I_{S_k}(i_m)\left(V(i_m)^2 - \left(\frac{1}{c_{k,t}}\sum_{p=0}^{t} I_{S_k}(i_p)V(i_p)\right)^2\right),$$

which is the sample variance of the function value of all matched states up until time t.

4 Gradient-based algorithmic approaches

In this section we will discuss several gradient-based algorithmic implementations of function approximation and their advantages and disadvantages. They all have in common that they follow the gradient of the error function or a local estimate of it. As their implementation is computationally and spatially cheap they are currently the standard choice for use in LCS and other machine learning methods.

The Genetic Algorithm in LCS relies on a good weight vector and error estimate for use as the fitness of a classifier. Therefore it is important to quickly get a good idea of the approximation error of a classifier. For that reason, we also discuss convergence rate and stability of the methods described.

4.1 Steepest gradient descent

Steepest Gradient Descent is a well-known method for function minimization, that performs small steps along the steepest gradient towards the next local minimum. Let us consider the mean-squared error f_k according to (3) as the function we want to minimize. Then, the steepest gradient descent algorithm is defined by

$$w_{k,t+1} = w_{k,t} - \alpha_t \frac{1}{2} \nabla_{w_k} f_k(w_{k,t}), \tag{7}$$

where $\alpha_t > 0$ is the scalar step size at time t, and $\nabla_{w_k} f_k(w_{k,t})$ is the gradient of f_k w.r.t. w_k, which is, according to (3),

$$\nabla_{w_k} f_k(w_{k,t}) = -2(\mathbb{E}_k(V\phi) + \mathbb{E}_k(\phi\phi')w_{k,t}).$$

The algorithm starts at an arbitrary weight vector $w_{k,0}$ and reduces the mean-squared error f_k with each step along the gradient. As the error function is a convex function, moving along the steepest gradient will under some assumptions lead us to the only minimum and with it to the optimal approximation.

4.1.1 Stability criteria

By definition, the step size α_t can change with time. If it is kept constant, that is $\alpha_t = \alpha$ for all $t \geq 0$, and the gradient $\nabla_{w_k} f_k(w_{k,t})$ is Lipschitz continuous,[12] then the steepest gradient descent method is guaranteed to converge to the minimum of the function, if that minimum exists (Bertsekas and Tsitsiklis 1996, Proposition 3.4). It is possible to show that the Lipschitz continuity holds if the Hessian $\nabla^2_{w_k} f_k(w_{k,t})$ is bounded over $\mathbb{R}^L$, which depends on the definition of the basis functions $\phi_1, \ldots, \phi_L$.

Another condition for the stability of steepest gradient descent, which is easier to evaluate, is for the step size α to hold

$$0 < \alpha < \frac{2}{\lambda_{k,\max}}, \tag{8}$$

where $\lambda_{k,\max}$ is the largest eigenvalue of the feature vector correlation matrix $\mathbb{E}_k(\phi\phi')$ (Haykin 2002, Chap. 4). As the feature correlation matrix is formed by the features of the states of the environment, the step size to keep the algorithm stable is highly dependent on the choice of the basis functions that form the feature vector.

4.1.2 Time constant bounds

The rate of convergence is also dependent on the eigenvalues of the feature vector correlation matrix. Let τ be the *time constant*[13] of the weight vector update. Then this time constant is bounded by

$$\frac{1}{-\ln(1-\alpha\lambda_{k,\max})} \leq \tau \leq \frac{1}{-\ln(1-\alpha\lambda_{k,\min})}, \tag{9}$$

[12] A function $f : M \to \mathbb{R}$ is Lipschitz continuous, if there exists a finite constant scalar K such that $\|f(a) - f(b)\| \leq K\|a - b\|$ for any $a, b \in M$. The magnitude K is a measure of the continuity of the function f.

[13] The time constant is a measure of the *responsitivity* of a dynamic system. A low time constant means that the system responds quickly to a changing input. Hence, it is inversely proportional to the rate of convergence.

where $\lambda_{k,\max}$ and $\lambda_{k,\min}$ are the largest and the smallest eigenvalue of $\mathbb{E}_k(\phi\phi')$ respectively (Haykin 2002, Chap. 4). As a low time constant τ implies a higher rate of convergence, we would prefer $\lambda_{k,\min}$ and $\lambda_{k,\max}$ to be close together for a tight bound and large such that $1-\alpha\lambda_k$ is close to 0, which maximizes the negative logarithm and keeps τ small. However, if the eigenvalues are widely spread, which indicates ill-conditioned features, then the settling time of the gradient descent algorithm is limited by the smallest eigenvalue $\lambda_{k,\min}$ (Bertsekas and Tsitsiklis 1996, Chap. 3). Therefore, the convergence rate is—as the stability criterion—dependent on the choice of the basis functions.

4.1.3 Applicability

As given by the algorithm definition, the steepest gradient descent method requires knowledge of the gradient $\nabla_{w_k} f_k(w_{k,t})$ for each step that it takes. Hence, we would have to have full knowledge of the mean-squared error, which requires full knowledge of our function V at each step. We could approach this by approximating the function over a finite number of steps, but we could never be sure of the quality of that approximation. Hence, we will treat the algorithm as a theoretical tool rather than one that we will use for implementation in LCS. It was discussed here, as it shares some theoretical properties with the Least Mean Squares algorithm that we will describe in the next section.

Example 3 (Stability criteria and time constant for some classifier types) Let us start with investigating averaging classifiers, defined by a feature vector of $\phi(i) = (1)$ for all $i \in S$, giving one eigenvalue $\lambda = 1$ for $\mathbb{E}_k(\phi\phi') = \mathbb{E}(I_{S_k})^{-1}\mathbb{E}(I_{S_k})$. Hence, according to (8) the steepest gradient descent method is stable for $0 \leq \alpha \leq 2$. Equation (9) gives its time constant as $\tau = -\ln(1-\alpha)^{-1}$, indicating a lower time constant (i.e. faster convergence) for a larger step size, which is what we would intuitively expect.

We can apply the same analysis to classifiers that approximate first-order polynomials, as given by the feature vector $\phi(i) = (1, i)'$. That gives the feature vector correlation matrix

$$\mathbb{E}_k(\phi\phi') = \mathbb{E}_k \begin{pmatrix} 1 & i \\ i & i^2 \end{pmatrix}$$

with the eigenvalues $\lambda_1 = 0, \lambda_2 = 1 + \mathbb{E}_k(i^2)$. Hence, for steepest gradient descent to be stable, the step size has to obey

$$0 \leq \alpha \leq \frac{2}{1 + \mathbb{E}_k(i^2)},$$

which demonstrates that the larger the values for the state i, the smaller the step size α has to be to still guarantee stability of the algorithm. The time constant is bounded by

$$\frac{-1}{\ln(1-\alpha(1+\mathbb{E}_k(i^2)))} \leq \tau \leq \infty,$$

which shows that a large eigenvalue spread $|\lambda_2 - \lambda_1|$ caused by on average high values for the state i pushes the time constant towards infinity, resulting in very slow convergence. Therefore, the convergence rate of steepest gradient descent depends frequently on the range of the features.[14] We will demonstrate this dependency empirically in Sect. 6.

[14] A similar analysis related to LCS was done in (Lanzi et al. 2005d), but there the stability criteria for steepest gradient descent were applied to the LMS algorithm.

4.2 Least mean square algorithm

The Least Mean Square (LMS) algorithm is very much related to steepest gradient descent, but rather than performing gradient descent on the full gradient of the function, it performs gradient descent on a current local approximation. For this reason it is also called the *Stochastic Incremental Steepest Gradient Descent* algorithm, or *ADALINE*, or, after their developers Widrow and Hoff (1960), the *Widrow-Hoff Update*.

Let us consider the mean-squared error (see (3)), or its step-wise approximation (see (4)), both of which take $I_{S_k}(i_t)(V(i_t) - w'_{k,t}\phi(i_t))^2$ as the error for state i_t at time t. The LMS applies the same update equation as steepest gradient descent (see (7)), but rather than taking the overall gradient, it approximates it by the gradient of the error at time t, as given above. That leads to the LMS update

$$w_{k,t+1} = w_{k,t} + \alpha_t I_{S_k}(i_t)\phi(i_t)(V(i_t) - w'_{k,t}\phi(i_t)), \tag{10}$$

which is the weight update equation used by XCS (Wilson 1995) for a feature vector of $\phi(i_t) = 1$. The term $I_{S_k}(i_t)$ has the effect that only classifiers that match the current state i_t are updated. As the gradient estimate is only based on the current state, this method suffers from *gradient noise*. Due to this noise, a constant step size α will cause the method to perform random motion close to the optimal approximation (Haykin 2002, Chap. 5).

4.2.1 Misadjustment due to local gradient estimate

The difference between the mean-squared error $f_k(w_k)$ and the minimum estimation error of the LMS algorithm is called the *excess mean squared estimation error*. The ratio between the excess mean squared estimation error and the minimum mean-squared error is called the *misadjustment*, which is a measure of how far away the convergence area of LMS is from the optimal approximation. The estimation error value for some small constant step size α can, according to (Haykin 2002, Chap. 5), be estimated by

$$f_k(w_k) + \frac{\alpha f_k(w_k)}{2} \sum_{j=1}^{J} \lambda_j,$$

where $f_k(w_k)$ stands for the minimum mean-squared error, and λ_j is the jth out of J eigenvalues of the feature vector correlation matrix $\mathbb{E}_k(\phi\phi')$. This shows that the excess mean squared estimation error is (i) always positive, and (ii) is proportional to the step size α. Thus, reducing the step size will also reduce the misadjustment due to local gradient estimation. Indeed, under the standard stochastic approximation assumptions that $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$, the Lipschitz continuity of the gradient, and some Pseudogradient property of the gradient, we can guarantee convergence to the optimal approximation (Bertsekas and Tsitsiklis 1996, Proposition 4.1).

4.2.2 Stability criteria and average time constant

As the LMS filter is a traversal filter of length one, using only the current observation for its approximation, no concrete bounds for the step size can be currently given (Haykin 2002, Chap. 6). However, if the step size is small when compared to the inverse of the largest eigenvalue of the feature vector correlation matrix, then the stability criteria are the same as for steepest gradient descent (see (8)).

As the gradient changes with each step, we can only give expressions for the local time constant that varies with time (for more details see Drugowitsch and Barry 2006a). On average, however, the time constant can be bounded in the same way as for steepest gradient descent (see (9)), with the same consequences.

As discussed before, the misadjustment is proportional to the step size. On the other hand, the time constant is inversely proportional to the step size. Hence, we have conflicting requirements and can either aim for a low estimation error or a fast rate of convergence, but we will not be able to satisfy both requirements with anything other than a compromise.

4.3 Normalized LMS algorithm

As we can see from the LMS update (10), the magnitude of the weight update is directly proportional to the feature vector $\phi(i_t)$, causing *gradient noise amplification* (Haykin 2002, Chap. 6). Hence, if we have large values in some elements of the feature vector, the correction based on a local error will be amplified and causes additional noise. This problem can be overcome by weighting the correction by the squared Euclidean norm of the feature vector. Hence, the update equation changes to

$$w_{k,t+1} = w_{k,t} + \alpha_t I_{S_k}(i_t) \frac{\phi(i_t)}{\|\phi(i_t)\|^2} (V(i_t) - w'_{k,t}\phi(i_t)),$$

which is the update equation that is used in XCSF and was introduced in (Wilson 2002). This update equation can also be derived by calculating the weight update vector that minimizes the norm of the weight change $\|w_{k,t+1} - w_{k,t}\|^2$, subject to the constraint $I_{S_k}(i_t) w'_{k,t+1}\phi(i_t) = V(i_t)$. As such, the normalized LMS filter can be seen as a solution to a constrained optimization problem.

Regarding stability, the step size parameter α is now weighted by the inverted squared norm of the feature vector. Hence, stability in the mean-squared error sense is dependent on the current state. The lower bound is still 0, and the upper bound will be generally larger than 2 if the state values are overestimated, and smaller than 2 otherwise. The optimal step size, located at the largest value of the mean-square deviation, is in the centre of the two bounds (Haykin 2002, Chap. 6).

As expected, the normalized LMS algorithm features a rate of convergence that is higher than that of the standard LMS filter, as demonstrated in simulation (Douglas 1994). One drawback of this modification is that $\|\phi(i_t)\|^2$ has to be checked for being zero, to avoid divisions by zero. In that case, no weight update needs to be performed, as $\|\phi(i_t)\|^2 = 0$ implies that $\phi(i_t)$ is zero in all its elements.

4.4 Summary

In this section we have introduced gradient-based approximation algorithms for single classifiers that are commonly used in LCS. Even though the steepest gradient descent algorithm cannot be used for sample-based approximation, it was discussed as it shares properties w.r.t. stability and rate of convergence with its local approximation, the LMS filter. Due to the gradient noise amplification we have also introduced the normalized LMS filter that avoids that problem and features a faster convergence rate.

Overall, both variants of the LMS filter have low computational and spatial costs. However, as they rely on the local approximation of the gradient, they introduce misadjustment that is proportional to the step size. As a result we have to compromise between a high convergence rate and a low misadjustment as we cannot achieve both at the same time. In

addition, the rate of convergence is influenced by the eigenvalues of the feature vector correlation matrix and might be severely reduced, as illustrated in Example 3 and empirically and analytically demonstrated in (Lanzi et al. 2005d).

The Genetic Algorithm in LCS relies on a fast and correct approximation error estimate which subsequently requires a good estimate of the optimal weight vector. Due to the dependence of the convergence rate of gradient-based methods on the choice of features, they might not give good estimates quickly enough for the GA to differentiate between good and bad classifiers, and might therefore degrade overall LCS performance. The next section will introduce a set of algorithms that are computationally more expensive than the LMS algorithm but feature a vastly improved tracking of both the optimal weight vector and its associated approximation error.

5 The Kalman filter and recursive least squares algorithm

The Kalman filter is a recursive solution to the discrete-data linear filtering problem (Kalman 1960). In this section we apply the Kalman filter to the approximation task and simultaneously to tracking the approximation error. Being able to use the Kalman filter for that task is of advantage as "[...] the Kalman filter is optimal with respect to virtually any criterion that makes sense" (Maybeck 1979, Chap. 1).

Additionally, we will show how using the Kalman filter for updating the weight vector of a classifier relates to the optimality conditions that we have introduced in Sect. 3, by showing its equivalence to the Recursive Least Squares algorithm. We will then use this knowledge to derive accurate approximation error tracking in addition to the weight update.

5.1 The system model and update equations

5.1.1 The classifier system model

The Kalman–Bucy system model (Kalman and Bucy 1961) describes how a noisy process modifies the state of a system, and how this affects the noisy observation of the system. Both the process and the observation are assumed to be linear, and all noise is zero-mean white Gaussian noise.

We will apply that model to single classifiers by assuming that the process is stationary which conforms to our assumption that the function to approximate is stationary with zero noise. In addition, the measurements are in a linear relation to the system state and all deviations from that linearity are covered by zero-mean white Gaussian noise. This gives us the model

$$V(i_t) = W'_{k,t}\phi(i_t) + \xi_{k,t}, \tag{11}$$

where $V(i_t)$ is the measurement, $W_{k,t}$ is our current model of the system state, $\phi(i_t)$ is the known measurement vector that specifies the linear relation between the system state and the measurement, and $\xi_{k,t}$ is the measurement noise, all at time t. The noise is independent and identically distributed with a variance of $\Xi_{k,t}$.

The system state W_k is modeled by a multivariate normal distribution that at a certain time t is given by its mean vector $w_{k,t} \in \mathbb{R}^L$ and its $L \times L$ covariance matrix $\Sigma^w_{k,t}$, that is $W_{k,t} = N(w_{K,t}, \Sigma^w_{k,t})$. Together with the zero-mean Gaussian noise, the measurement also can be modeled by a Gaussian given by $N(V(i_t), \Xi_{k,t})$, and the system state and the measurements are jointly Gaussian. More details on the distribution and interrelation of the different system variables can be found in (Maybeck 1979, Chap. 5) or (Anderson and Moore 1979, Chap. 1).

5.1.2 Bayesian estimation update

The aim of a classifier is to get a good model of the system state $W_{k,t}$ and of the measurement noise variance $\Xi_{k,t}$ that can then be used for predicting a measurement given a certain measurement vector $\phi(i)$. By our model according to (11), that prediction is given by the univariate normal distribution $N(w'_{k,t}\phi(i), \Xi_{k,t})$ which is centred on $w'_{k,t}\phi(i)$ and has variance $\Xi_{k,t}$. This clearly shows the relation to our previously introduced linear approximation architecture (see (2)), which is the mean—and equally the maximum likelihood—of the prediction for the measurement given by the Kalman filter classifiers.

Starting with a prior for the system state $W_{k,-1} = N(w_{k,-1}, \Sigma^w_{k,-1})$, we can include the information given by the measurements by conditioning the system state on those measurements. Considering the previously described distribution relation between the different system variables, we derive (see Appendix B.1) the following Bayesian update equations for conditioning the previous system state $W_{k,t-1}$ on the measurement $N(V(i_t), \Xi_{k,t})$:

$$\zeta_{k,t} = I_{S_k}(i_t)\Sigma^w_{k,t-1}\phi(i_t)(I_{S_k}(i_t)\phi(i_t)'\Sigma^w_{k,t-1}\phi(i_t) + \Xi_{k,t})^{-1}, \tag{12}$$

$$w_{k,t} = w_{k,t-1} + \zeta_{k,t}(V(i_t) - w'_{k,t-1}\phi(i_t)), \tag{13}$$

$$\Sigma^w_{k,t} = \Sigma^w_{k,t-1} - \zeta_{k,t}\phi(i_t)'\Sigma^w_{k,t-1}. \tag{14}$$

While $\zeta_{k,t}$ is a temporary measure that depends on the previous $\Sigma^w_{k,t-1}$ and information about the current observation $\{\phi(i_t), I_{S_k}(i_t), \Xi_{k,t}\}$, both $w_{k,t}$ and $\Sigma^w_{k,t}$ are the model-defining parameters that are updated from their previous state $w_{k,t-1}$ and $\Sigma^w_{k,t-1}$. This form of the Kalman filter update is commonly called the *Covariance Form*. Let us for now assume that we know the variance $\Xi_{k,t}$ of the current measurement. This is obviously not the case and we will later show how to estimate the measurement noise variance at the same time as the system state.

The measurement residual $V(i_t) - w'_{k,t-1}\phi(i_t)$ in (13) is the difference between the measurement $V(i_t)$ and its estimate $w'_{k,t-1}\phi(i_t)$ before $V(i_t)$ is known. The *Kalman gain* ζ_t determines how much the current estimate is corrected.

From the update equation we can see that as the measurement noise variance $\Xi_{k,t}$ approaches zero, the gain $\zeta_{k,t}$ weights the residual more heavily. On the other hand, as the weight covariance $\Sigma^w_{k,t}$ approaches zero, the gain $\zeta_{k,t}$ assigns less weight to the residual (Welch and Bishop 2004). This is the behavior that we would intuitively expect, as low-noise measurements should be valued higher than high-noise measurements.

5.1.3 Inverse covariance form

Using the Kalman filter for estimating the system state requires the setting of the prior estimate $W_{k,-1}$. In many cases, we do not have any knowledge about what the correct values might be and setting random initial values will cause an unnecessary bias. Complete lack of information about the initial system state can be modeled as the limiting case of certain eigenvalues of $\Sigma^w_{k,-1}$ going to infinity (Maybeck 1979, Chap. 5.7), hence using a non-informative prior. This will work in theory, but would cause problems in implementation due to large numerical errors when evaluating the Kalman gain according to (12).

Another approach to the Kalman update is to operate on the inverse of the covariance rather than the covariance itself (for derivation see Appendix B.2), resulting in the update

$$(\Sigma^w_{k,t})^{-1} = (\Sigma^w_{k,t-1})^{-1} + I_{S_k}(i_t)\phi(i_t)\Xi^{-1}_{k,t}\phi(i_t)', \tag{15}$$

$$(\Sigma^w_{k,t})^{-1}w_{k,t} = (\Sigma^w_{k,t-1})^{-1}w_{k,t-1} + I_{S_k}(i_t)\phi(i_t)\Xi^{-1}_{k,t}V(i_t). \tag{16}$$

Note that rather than updating the weight vector $w_{k,t}$ directly, we are now performing updates on the vector $(\Sigma^w_{k,t})^{-1}w_{k,t}$. The weight estimate can be recovered by

$$w_{k,t} = [(\Sigma^w_{k,t})^{-1}]^{-1}[(\Sigma^w_{k,t})^{-1}w_{k,t}].$$

This shows that we are only required to perform a matrix inversion for the computation of the value of the weight vector, but not for the update itself. This is of advantage if our initial inverse covariance $(\Sigma^w_{k,-1})^{-1}$ is singular, which is the case if we set it to zero to avoid initial estimation bias. Hence, we can perform updates until the inverse covariance attains full rank, which subsequently allows for a unique weight estimate. This is not the case for the Kalman filter in its Covariance Form, as given by (12), (13) and (14).

Example 4 (Inverse covariance form for singular inverse covariance) Let us consider a classifier with feature vectors $\phi(i) = (1, i)'$. Setting $(\Sigma^w_{k,-1})^{-1} = 0$ allows us to avoid any initial bias. Let us now assume that at time $t = 0$ we observe value $V(i_0)$ in state i_0, which is matched by classifier k. According to (15) and (16), the classifier parameters of classifier k are updated to

$$(\Sigma^w_{k,0})^{-1} = \begin{pmatrix} \Xi^{-1}_{k,0} & \Xi^{-1}_{k,0} i_0 \\ \Xi^{-1}_{k,0} i_0 & \Xi^{-1}_{k,0} i_0^2 \end{pmatrix},$$
$$(\Sigma^w_{k,0})^{-1} w_{k,0} = \begin{pmatrix} \Xi^{-1}_{k,0} V(i_0) \\ \Xi^{-1}_{k,0} i_0 V(i_0) \end{pmatrix}.$$

Clearly, $w_{k,0}$ cannot be computed due to the singularity of $(\Sigma^w_{k,0})^{-1}$. However, we can still perform further updates of the inverse covariance until it is of full column rank and can be inverted. Until then, we can use the pseudo-inverse to obtain first estimates of the weight vector, resulting in

$$w_{k,0} = \frac{1}{1 + i_0^2} \begin{pmatrix} V(i_0) \\ i_0 V(i_0) \end{pmatrix}.$$

5.2 Equivalence to recursive least squares

The Recursive Least Squares (RLS) algorithm gives a method of accurately tracking the weight vector that minimizes the sample-based mean-squared error, as given by (4). We will here present a variant that minimizes a *weighted* mean-squared error so that we can relate it to the Kalman filter.

Let us assume that we want to minimize a mean-squared error that at each time m is weighted by some value $\Xi^{-1}_{k,m}$, with the error given by

$$\sum_{m=0}^{t} I_{S_k}(i_m) \Xi^{-1}_{k,m} (V(i_m) - w'_{k,t} \phi(i_m))^2. \tag{17}$$

Then, starting with $(\Sigma^w_{k,-1})^{-1} = 0$, the iterative update

$$w_{k,t} = w_{k,t-1} + I_{S_k}(i_t) \Xi^{-1}_{k,t} ((\Sigma^w_{k,t1})^{-1})^{-1} \phi(i_t)(V(i_t) - w'_{k,t-1}\phi(i_t)), \tag{18}$$
$$(\Sigma^w_{k,t})^{-1} = (\Sigma^w_{k,t-1})^{-1} + I_{S_k}(i_t)\phi(i_t)\Xi^{-1}_{k,t}\phi(i_t)', \tag{19}$$

tracks the weight vector $w_{k,t}$ that minimizes above error. The derivation of the update equations can be found in Appendix B.3.

It is easy to see that (19) is equivalent to the covariance update of the Inverse Covariance Form of the Kalman filter, given by (15). By pre-multiplying (18) by $(\Sigma^w_{k,t})^{-1}$ we get

$$\begin{aligned}(\Sigma^w_{k,t})^{-1}w_{k,t} &= (\Sigma^w_{k,t})^{-1}w_{k,t-1} + I_{S_k}(i_t)\Xi^{-1}_{k,t}\phi(i_t)V(i_t) - I_{S_k}(i_t)\Xi^{-1}_{k,t}\phi(i_t)w'_{k,t-1}\phi(i_t)\\ &= (\Sigma^w_{k,t-1})^{-1}w_{k,t-1} + I_{S_k}(i_t)\Xi^{-1}_{k,t}V(i_t)\phi(i_t),\end{aligned}$$

which is equivalent to the Kalman filter weight update, as given by (16).

This equivalence shows that the Kalman filter tracks the weight vector that minimizes the weighted mean-squared error and is therefore optimal in the mean-squared error sense. An additional result is that the noise does not necessarily have to be normally distributed, as that assumption is not taken by the RLS update. Hence, we can get a minimum-variance weight estimate using the Kalman filter even when discarding the assumption of Gaussian noise.

5.3 Estimating error and weight simultaneously

So far we have assumed that the measurement noise variance $\Xi_{k,t}$ is known for each measurement. With respect to our system model that is equivalent to knowing the final approximation error *before* estimating the weight vector. This is naturally not the case in LCS, where for every classifier we want to find the optimal weight vector at the same time as estimating the approximation error. Finding a method to do this is the topic of this section.

5.3.1 Minimum model error approximation

In the Kalman system model (11) all deviations of the measurements from the linear model $W'_{k,t}\phi(i_t)$ are attributed to the measurement noise $\xi_{k,t}$. We could attempt to get an expression for the noise variance at each measurement, using the information gained by the measurement values. However, at time t having $t+1$ measurements and the same number of measurement noise variances, and in addition requiring to find the values of the weight vector, we have more degrees of freedom than we have measurements.

To reduce the degrees of freedom we will make the assumption that the measurement noise variance is constant over all measurements, that is $\Xi_{k,m} = \Xi_k$ for all $m = 0, 1, \ldots$. In addition we will adopt the *Minimum Model Error (MME)* philosophy that aims at finding the approximation that minimizes the model error, given by the measurement noise variance (Mook and Junkins 1988). It is based on the *Covariance Constraint* condition, which states that the measurement-minus-estimate error covariance must match the measurement-minus-truth error covariance, that is

$$I_{S_k}(i_t)(V(i_t) - w'_{k,t}\phi(i_t))^2 \approx \Xi_k.$$

Given that constraint and the assumption of not having any process noise we can define the model error at time t by weighting the left-hand side of above equation by the inverted right-hand side, that is

$$\Xi_k^{-1}\sum_{m=0}^{t} I_{S_k}(i_m)(V(i_m) - w'_{k,t}\phi(i_m))^2.$$

Note that this error is equivalent to the approximated mean-squared error $\varepsilon_{k,t}$ (4) scaled by a constant, and therefore has the same minimum. Assuming a constant measurement noise variance has led us to minimize the same sample-based error that we originally intended to minimize.

5.3.2 Incremental error update

Tracking the weight vector with the Kalman filter under the constant measurement noise assumption minimizes the approximated mean-squared error with each additional measurement and therefore conforms to the Principle of Orthogonality (Theorem 2). Thus, we can use the minimum error term from that theorem to get the following incremental error update:

Theorem 3 *Given the sequence of states* $\{i_0, i_1, \ldots\}$, *the sequence of features* $\{\phi(i_0), \phi(i_1), \ldots\}$, *and the sequence of function values* $\{V(i_0), V(i_1), \ldots\}$, *and an iterative weight vector update that conforms to the Principle of Orthogonality* (*Theorem* 2), *then the update*

$$\begin{aligned}(c_{k,t} - L)\varepsilon_{k,t} = (c_{k,t-1} - L)\varepsilon_{k,t-1} \\ + I_{S_k}(i_t)(V(i_t) - w'_{k,t}\phi(i_t))(V(i_t) - w'_{k,t-1}\phi(i_t)),\end{aligned}$$

starting with $\varepsilon_{k,-1} = 0$ *tracks the approximated mean-squared error* (4).

Proof See Appendix B.4. □

Therefore, by setting $\Xi_k = 1$ in any form of the Kalman filter update, the tracked weight vector conforms to the Principle of Orthogonality. This allows us to use above theorem to track the approximation error with similar accuracy.

5.3.3 Non-Gaussian error

We have previously demonstrated that the Kalman filter tracks the weight vector that minimizes a weighted mean-squared error, where the weight is given for each measurement by the inverse of the measurement noise variance. Having now assumed that this variance is constant, it can be removed from the error expression that is to be minimized, and our weight vector tracking conforms to the Principle of Orthogonality and is therefore optimal in the mean-squared error sense.

The Kalman filter assumes Gaussian noise, but by relating it to the RLS algorithm we have shown that this assumption is not necessary. Similarly, the above error update is derived without making any assumptions about how the error might be distributed. Therefore, we can use both the weight update and the error update in cases where the error is non-Gaussian, and will still achieve minimal variance.

5.3.4 Convergence

With each weight and error update we find the approximation that minimizes the approximated mean-squared error (4). As already stated in Sect. 2.2.2, it is known that this sample-based error measure converges to the mean-squared error (3) with $t \to \infty$. Hence, the optimality condition of Theorem 2 converges to that of Theorem 1 for full knowledge of the function to approximate. As a consequence, both the estimated weight vector and the approximated mean-squared error will converge to the optimum given by Theorem 1, with probability 1.

Example 5 (Kalman filter update for averaging classifiers) Let us consider an averaging classifier k, using feature vector $\phi(i) = (1)$ for all $i \in S$. As we want to estimate the measurement noise variance simultaneously to finding the optimal approximation, we will assume it to be a constant Ξ_k. To avoid any initial bias, let $(\Sigma^w_{k,-1})^{-1} = 0$. Then, according to (15), the inverse covariance is updated by

$$(\Sigma^w_{k,t})^{-1} = (\Sigma^w_{k,t-1})^{-1} + I_{S_k}(i_t)\Xi_k^{-1} = \Xi_k^{-1} c_{k,t},$$

which is the match count, scaled by the inverse measurement noise variance. Hence, the covariance (which is in this case a variance) of our weight estimate decreases exponentially with the number of matched states that were visited. That shows that our estimate will become increasingly more accurate.

The weight update is performed according to (16), that is

$$\begin{aligned}(\Sigma^w_{k,t})^{-1} w_{k,t} &= (\Sigma^w_{k,t-1})^{-1} w_{k,t-1} + I_{S_k}(i_t)\Xi_k^{-1} V(i_t) \\ &= \Xi_k^{-1} \sum_{m=0}^{t} I_{S_k}(i_m) V(i_m),\end{aligned}$$

giving for $w_{k,t}$

$$w_{k,t} = \frac{(\Sigma^w_{k,t})^{-1} w_{k,t}}{(\Sigma_{k,t})^{-1}} = \frac{1}{c_{k,t}} \sum_{m=0}^{t} I_{S_k}(i_m) V(i_m),$$

which is the average over the function values of the state sequence $\{i_0, \ldots, i_1\}$, when only considering matched states. Note that due to the constant measurement noise variance, this weight estimate is independent of the measurement noise variance.

As the weight vector is optimal for each t, applying the error approximation update from Theorem 3 tracks the function sample variance (see Example 2). Note that the variance of the weight estimate indicates how sure we are about the current estimate and is a different measure than the sample variance, which indicates how well the linear architecture is able to approximate the function of its matching states.

Interestingly, XCS applies the MAM update that is equivalent to averaging the input for the first α^{-1} inputs, where α is the step size, and then tracks the input using the LMS algorithm (Wilson 1995). In other words, it bootstraps its weight estimate using the Kalman filter algorithm with a measurement noise variance of 1, and then continues tracking of the input using the LMS algorithm. Note that this is only the case for XCS applying averaging classifiers, and does not apply to XCS-derivates that apply more complex function approximation, such as XCSF (Wilson 2002). Even though it is not explicitly stated in (Wilson 2002), we assume that the MAM update was not used in those XCS-derivates.

5.4 A computationally cheap implementation

In addition to the standard form of the Kalman filter, as given by the Covariance Form, the Inverse Covariance Form, and the RLS algorithm, we present a computationally cheap implementation which we have used for experiments, but for which we cannot give any guarantees on its numerical stability when applied for long time frames.

As our developments aim at application in LCS, we want to approximate the error while tracking the optimal weight vector. This calls for the use of a constant measurement noise variance, which consequently allows us to use the approximated mean-squared error (4) as

our approximation goal. Hence, we can utilize the Principle of Orthogonality (Theorem 2), from which we can rewrite the condition for the optimal weight vector as follows

$$\left(\sum_{m=0}^{t} I_{S_k}(i_m)\phi(i_m)\phi(i_m)'\right)w_k = \sum_{m=0}^{t} I_{S_k}(i_m)V(i_m)\phi(i_m).$$

Let us denote the matrix in brackets of the left-hand side by $A_{k,t}$ and the vector of the right-hand side by $b_{k,t}$. We can then compute the weight vector $w_{k,t}$ by

$$w_{k,t} = A_{k,t}^{-1} b_{k,t},$$

given that $A_{k,t}$ is non-singular. Vector $b_{k,t}$, starting with $b_{k,-1} = 0$, can be updated by

$$b_{k,t} = b_{k,t-1} + I_{S_k}(i_t)V(i_t)\phi(i_t).$$

To avoid taking the inverse every time we update $A_{k,t}$, we can apply the Sherman–Morisson formula to operate on the inverse, that is

$$A_{k,t}^{-1} = A_{k,t-1}^{-1} - I_{S_k}(i_t)\frac{A_{k,t-1}^{-1}\phi(i_t)\phi(i_t)'A_{k,t-1}^{-1}}{1 + I_{S_k}(i_t)\phi(i_t)'A_{k,t-1}^{-1}\phi(i_t)}.$$

The approximation error is again updated according to Theorem 3.

To recover the covariance matrix $\Sigma_{k,t}^{w}$, we can give its inverse by

$$(\Sigma_{k,t}^{w})^{-1} = \Xi_k^{-1}\sum_{m=0}^{t} I_{S_k}(i_m)\phi(i_m)\phi(i_m)',$$

which is matrix $A_{k,t}$ scaled by the measurement noise variance Ξ_k. As the error $\varepsilon_{k,t}$ is the current best approximation of the measurement noise variance, we can calculate the covariance matrix by

$$\Sigma_{k,t}^{w} = \varepsilon_{k,t}A_{k,t}^{-1}.$$

The disadvantage of this update is that we require matrix $A_{k,t}$ to be set to an initial value $A_{k,-1}$, which introduces a bias to the approximation. This bias can be kept low by setting $A_{k,-1} = I\delta$, where δ is a large scalar and causes the initial inverse $A_{k,t}^{-1}$ to be small. This can still introduce numerical instabilities, but should be sufficient for most applications. Additionally, the update still converges to the same optimal approximation as $t \to \infty$. Setting $A_{k,-1}^{-1} = 0$ is invalid as it will cause $A_{k,t}^{-1} = 0$ for all $t = 0, 1, \ldots$.

5.5 Stability of the Kalman filter

The Covariance Form of the Kalman filter, as summarized by (12), (13) and (14) suffers from serious numerical difficulties, as described in (Haykin 2002, Chap. 6). Considering (12) and (14), for example, the covariance matrix is updated by the difference between two non-negative matrices. Unless the system is numerically completely accurate, the result of the subtraction may not be non-negative definite, which is not acceptable for a covariance matrix. Additionally, the property of symmetry of the covariance matrix might not be preserved due to numerical inaccuracy. The Inverse Covariance Form, given by (15) and (16), shows better properties but is still affected by the possible loss of symmetry.

A method for overcoming these stability issues is to use numerically stable unitary transforms at every iteration, such as, for example, the *Square-Root Form* of the Kalman filter. This form is mathematically equivalent to the other Kalman filter forms, but only operates on the lower triangle of the covariance matrix and hence preserves symmetry. In addition, in that form the matrix is much less likely to become indefinite. We will not discuss any further details here, but refer the interested reader to (Haykin 2002, Chap. 11), (Anderson and Moore 1979, Chap. 6.5) or (Maybeck 1979, Chap. 7).

5.6 Comparison to the LMS filter

As stated in (Haykin 2002, Chap. 9), the RLS algorithm typically features an order of magnitude faster rate of convergence than that of the LMS algorithm. Additionally, its rate of convergence is less sensitive to the spread of eigenvalues of the feature vectors. As our use of the Kalman filter is equivalent to a form of the RLS algorithm, these findings also apply to the Kalman filter.

On the downside, the Kalman filter is both computationally and spatially more complex than the LMS filter. It requires the storage of the covariance of the weight estimates, and employs matrix multiplication and eventual inversion, depending on the update type that is used. A detailed summary of the different forms of Kalman filters and their computational complexities can be found in (Maybeck 1979, Chap. 7.8).

5.7 Summary

In this section we have introduced algorithms that are able to track both the weight vector and the approximation error according to our previously given optimality conditions. The Kalman filter was derived as a Bayesian update and based on the assumption of Gaussian noise. Later this assumption was relaxed by relating it to the RLS algorithm that minimizes the weighted mean-squared error but does not assume a particular error distribution.

To track the approximation error we have shown how the assumption of a constant measurement noise makes the weight update conform to the Principle of Orthogonality, from which we can then derive an incremental update equation that tracks the minimal error on which this principle is based.

As already emphasized, a rapid and accurate estimate of the quality of a classifier is crucial for the performance of the Genetic Algorithm in LCS. The weight and error update equations derived in this section provide such an estimate and can therefore be expected to improve the general performance of LCS. The commonly used LMS filter is known to suffer from several drawbacks that deteriorate its rate of convergence, and should therefore only be used if the spatial and computational costs of the Kalman filter prohibit its employment. Considering only averaging classifiers with a weight vector of size 1, the Kalman filter has the same complexity as LMS (see Example 5) and should therefore always be the preferred choice.

6 Experimental comparison

In this section we demonstrate the superiority of Kalman filter based classifiers when compared to gradient-based methods. The comparison is performed based on the current use of the LMS and RLS algorithm in XCS and XCSF. It is by no means meant to be a complete evaluation of the use of Kalman filter classifiers in LCS, but only a demonstration of previously discussed concepts like the misadjustment of the LMS filter, and the reduced rate

of convergence for ill-conditioned features. We will perform a full analysis of the system once we completed at least a primary analysis and eventual improvement of all component of LCS. Until then, we will use the Kalman filter to act as a model that helps us to gain better understanding and is built upon in further analysis (e.g. Drugowitsch and Barry 2006b, 2006c). For the Kalman filter classifier to act as a drop-in replacement for currently used methods its evaluation has to extended beyond experiments with single classifiers to its use in combination with the Genetic Algorithm, such as in data-mining using the UCI data set (Hettich and Bay 1999) as a benchmark, and in multi-step tasks in both discrete and continuous environments (Littman 2005).

For a comparison of the performance of different classifier types in the current state-of-the-art classifier system XCSF the interested reader is referred to (Lanzi et al. 2005d), where Lanzi et al. introduce the RLS classifier (using LMS for error tracking) and compare its performance in function approximation tasks to XCSF using the LMS algorithm. In a follow-up study, they also evaluate its use in discrete (Lanzi et al. 2005b) and continuous (Lanzi et al. 2005c) multi-step tasks. In a recent study (Loiacono 2006) we have collaborated with Loiacono and Lanzi to demonstrate the effect of the better error estimate of the Kalman filter on the generalization capabilities of XCSF. The results confirmed that the higher accuracy in its error prediction enabled the Genetic Algorithm to push the classifier population closer to the desired generalization boundary.

Here we will look at single classifiers and compare the rate of convergence and sensitivity to noise between different classifier types. The two cases that we will deal with are (i) averaging classifiers that have to approximate a noisy constant, and (ii) classifiers that approximate parts of a sinusoid by a first-order polynomial. We have chosen the first case to investigate the accuracy of the error approximation. The second case tests both the approximation of the weight vector and the mean squared error.

Statistical comparison of the different methods is for the weight vector based on the mean squared error of its prediction when compared to the optimal prediction over the classifier's matched range, evaluated over 1000 evenly spaced samples. The error estimates are compared by their squared difference to the mean-squared errors of the optimal weight vector's error. As either comparison is done on a performance curve that results from plotting the error over time, we cannot apply the standard analysis of variance (ANOVA) to determine if the methods perform significantly different, as its assumption of homogeneity of covariances is violated. Thus, we utilize the randomized ANOVA procedure (Piater et al. 1998), that was specifically design for such tasks. It is based on estimating the sampling distribution of the null hypothesis ("all methods feature the same performance") by sampling the standard two-way ANOVA F-values from randomly reshuffled performance curves between the methods, where we use a sample size of 5000. The two experimental factors are the type of classifier that is used, and the number of samples that the classifier has been trained on, where performance is measured by the corresponding error when compared to the optimal values. We are only reporting significant difference between classifier types, and are using Tukey's HSD post hoc test to determine the direction of the effect. All statistical analysis is performed over 20 experimental runs.

6.1 Classifier types

We will compare three kinds of classifiers:

Original XCS Classifiers (Wilson 1995): These classifiers apply the LMS algorithm for the averaging case (i.e. using a feature vector of $\phi(i) = (1)$ for all $i \in S$), and the normalized LMS algorithm for any other case (as introduced in Wilson 2002). In the averaging

case, they will additionally perform MAM update, as described in Example 5. The approximation error is tracked by applying the LMS algorithm to the current error of the error estimate

$$I_{S_k}(i_t)(I_{S_k}(i_t)(V(i_t) - w'_{k,t}\phi(i_t))^2 - \varepsilon_{k,t-1})^2,$$

which gives

$$\varepsilon_{k,t} = \varepsilon_{k,t-1} + \alpha_t I_{S_k}(i_t)((V(i_t) - w'_{k,t}\phi(i_t))^2 - \varepsilon_{k,t-1}).$$

To be precise, the error that is tracked in XCS is for no apparent reason the mean-absolute error $I_{S_k}(i_t)|V(i_t) - w'_{k,t}\phi(i_t)|$, but to be consistent with our error definition and to keep the results comparable, we have chosen to modify the algorithm to track the mean-squared error.[15] For all experiments, the step size α is kept constant and set to the commonly applied value $\alpha = 0.2$. All other values are initialized to 0.

RLS Classifiers (Lanzi et al. 2005d): The RLS Classifiers use the Recursive Least Squares algorithm, as introduced in Sect. 5.2, for its weight update, with a constant measurement noise variance of $\Xi_k = 1$.[16] Its covariance matrix $\Sigma^w_{k,t}$ is initialized to $\Sigma^w_{k,-1} = \delta I$ with $\delta = 1000$. Error update is performed in the same way as for the original XCS classifiers by the LMS algorithm. The step size is again set to $\alpha = 0.2$.

Kalman filter Classifiers: These classifiers apply the Kalman filter update under the Minimum Model Error philosophy, as described in this paper. The form of update used is described in Sect. 5.4. It is initialized with $\delta = 1000$.

6.2 Tracking the error

In our first experiment we apply averaging classifiers, using a feature vector of $\phi(i) = (1)$, for all $i \in S$. Each classifier matches all states, which is why we can reduce the system to a single state without the loss of generality. The sampled function values of that state are sampled from $N(5, 1)$, denoting a normal distribution with mean 5 and variance 1. We have not considered noisy sampling in developing conditions for optimality and discussing the algorithms.[17] However, by assuming that each sample describes the function value of a different state, the developed theory is still valid. The optimal values are $w_k(1) = 5$ and $f_k(w_k) = 1$.

Figure 1 shows the results of a single run over the first 50 samples. As the weight vector for the XCS classifier is initialized to $w_{k,-1} = 0$, it requires about 15 samples to get close to the optimal weight value. Due to considering only the local gradient, it then performs stochastic motion around that optimal point. The RLS and Kalman classifiers both apply a mathematically equivalent algorithm and therefore also have equivalent weight approximations. These approximations are obviously significantly more stable and less noisy, which is as expected, as they track the optimal approximation given all past observations. Statistical analysis has confirmed that the difference is indeed significant (randomized ANOVA:

[15] This modification to the XCSF error measure was after initial submission of this paper also proposed in (Loiacono 2006).

[16] The standard RLS algorithm does not consider measurement noise, and therefore always operates under the condition of a constant measurement noise variance.

[17] As can be seen from the Kalman filter system model (11), both the model error and the measurement noise are captured by the noise component. Therefore, additional noise in measuring the function values will increase the approximation error that is contributed to the deviation from the linear model by the variance of that noise. Given that the *real* measurement noise is equal for all classifiers, it will increase in the approximation error for all classifiers by an equal amount.

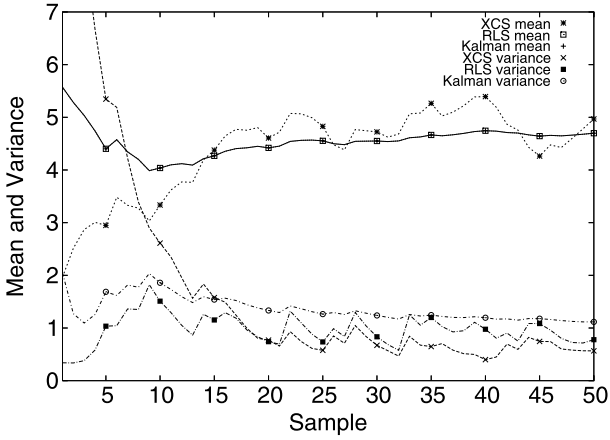

Fig. 1 Comparing the approximation of the mean and the variance of values sampled from $N(5, 1)$, using different classifier types. The optimal mean is 5, which is quickly reached by both the RLS and the Kalman filter classifier, as they are using formally equivalent methods to estimate their weight values. The XCS classifier's estimate of the weight value, on the other hand, is more noisy, which is due to the local gradient estimate of the LMS algorithm. This noise is also visible in the variance estimate of the XCS and RLS classifiers, both of which utilize the LMS algorithm to estimate the variance. The Kalman filter, on the other hand, performs direct tracking, and thus features a visibly more stable estimate close to the optimum of 1

$F_{\text{alg}}(2, 2850) = 38.0$, $F^*_{\text{alg},0.01} = 25.26$, $p < 0.01$; Tukey's HSD: no difference between Kalman and RLS, $p > 0.05$, but both significantly different to XCS, $p < 0.01$).

The approximated error, given by the variance, reflects the same picture. Both XCS and RLS classifiers use the LMS algorithm to update their error approximation and therefore require more initial time to get close to the optimal, and then perform stochastic motion around it. The XCS variance estimate is particularly bad in the beginning, as its approximation of the mean is worse than for the RLS classifier. The Kalman filter classifier again features better initial performance and higher stability in its approximation. Even though the RLS classifier seems to reside closer to the optimal error approximation, this is due to the initial bias of setting $\varepsilon_{k,-1} = 0$. The Kalman filter classifier initially overestimates the variance, but its estimate is theoretically optimal and its initial deviation from the long-run optimum stems from the randomness of the sampling process. Statistical comparison over 20 experimental runs does not reveal any difference between the error estimation performance of the XCS and RLS classifier, but shows that they are significantly worse than Kalman filter (randomized ANOVA: $F_{\text{alg}}(2, 2850) = 53.68$, $F^*_{\text{alg},0.001} = 29.26$, $p < 0.001$; Tukey's HSD: no difference between XCS and RLS, $p > 0.05$, but both significantly different to Kalman, $p < 0.01$).

6.3 Sensitivity to ill-conditioned features

The second experiment requires the classifiers to approximate a sinusoid $V(i) = \sin(i)$, using a first-order polynomial, given by the feature vector $\phi(i) = (1, i)'$, for all $i \in S$. We employ two classifiers, 1 and 2, of which the first matches the range $S_1 = [0, \frac{\pi}{2})$, and the second matches the range $S_2 = [\frac{\pi}{2}, \pi)$. Evaluating the optimality conditions of Theorem 1 (see Appendix B.5 for derivation), we get the optimal weight vectors $w_1 \approx (0.115, 0.665)'$ and $w_2 \approx (2.202, -0.665)$, and equal mean-squared errors $f_1(w_1) = f_2(w_2) \approx 0.00394$.

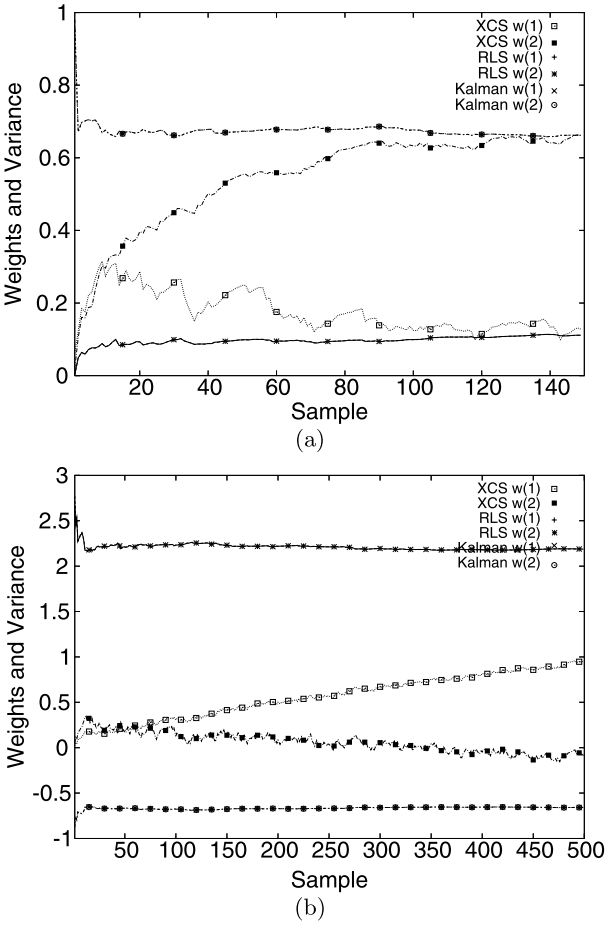

Fig. 2 Approximating a sinusoid using a first-order polynomial and different classifier types. **a** shows the weight values for the approximation over the range $[0, \frac{\pi}{2})$. **b** shows the same over the range $[\frac{\pi}{2}, \pi)$. While the RLS algorithm used by both the RLS and the Kalman filter estimates the optimal weight values (see text) quickly, the LMS algorithm used by the XCS algorithm suffers from slow convergence. The convergence rate of the LMS algorithm depends on the value and range of the feature vectors, and is worse for larger values, as is clearly visible when comparing graphs (**a**) and (**b**)

A single experiment consists of two runs, where in the kth run we uniformly sample states from S_k and only observe classifier k.

Figure 2 shows the first 150 observations for classifier 1 in (a), and the first 500 observations for classifier 2 in (b), for a single experimental run. As in the first experiment, the weight vectors are congruent for RLS and Kalman filter classifiers, as their approximation methods are mathematically equivalent. For both classifiers, the Kalman filter approximations feature better initial performance and higher stability if compared to XCS using the normalized LMS algorithm. What becomes particularly apparent in Fig. 2(b) is that ill-conditioned features reduce the rate of convergence of the LMS algorithm, which was already postulated in Example 3 and is now confirmed by the slow convergence of XCS. As the error approximation behaves similar to the first experiment, with the Kalman filter demonstrating better initial values and higher stability than both the RLS and XCS classifiers (and even worse initial performance of XCS than in the first experiment), we have omitted the graph showing that error approximation.

In terms of performance when compared to the optimum by evaluating the mean squared errors over 1000 evenly spaced samples for each of the classifiers, the XCS classifier performs significantly worse than the RLS or Kalman filter classifier in both cases,

as confirmed by statistical analysis over 20 experimental runs (for $[0, \frac{\pi}{2})$: randomized ANOVA: $F_{\text{alg}}(2, 2850) = 171.41$, $F^*_{\text{alg},0.001} = 32.81$, $p < 0.001$; Tukey's HSD: XCS vs. RLS and Kalman, $p < 0.01$, Kalman vs. RLS, $p > 0.05$; for $[\frac{\pi}{2}, \pi)$: randomized ANOVA: $F_{\text{alg}}(2, 17100) = 4268.7$, $F^*_{\text{alg},0.001} = 577.89$, $p < 0.001$; Tukey's HSD: XCS vs. RLS and Kalman, $p < 0.01$, RLS vs. Kalman, $p > 0.05$).

6.4 Summary

The experiments have confirmed what was previously predicted: gradient-based methods are generally slower to converge and more noisy than methods that track the optimal weight vector and approximation error accurately. The first experiment shows this difference clearly for the weight and error estimate of XCS classifiers and the error estimate of RLS classifiers. The second experiment demonstrates how ill-conditioned features reduce the rate of convergence of gradient-based methods significantly. These results reemphasize that gradient-based classifiers should only be used if computational or spatial constraints prohibit the use of Kalman filter classifiers.

7 Mixing classifier estimates

So far, we have concentrated on finding optimal approximations for single classifiers. This section concentrates on how the approximation of single classifiers can be assembled to give an approximation over the whole domain of the function V.

We have already described how mixing is performed in our framework (see (5)), and presented arguments why we should prefer averaging over approximations rather than aggregating them. The open question that we will consider in this section is how to set the mixing weights $\psi_k(i)$. For that purpose, we will assume a fixed set of classifiers that have all converged to the optimal approximation. Hence, we will drop the subscript $\cdot_t$, making the parameters time-independent, including $\varepsilon_{k,t}$, which we will simply denote by ε_k. Nevertheless, all findings from this section are also valid for the time-dependent case, as we can simply observe the LCS at a fixed time t and follow the same line of reasoning.

7.1 Accuracy-based mixing

By definition, the mean-squared error of a classifier gives the average approximation error over its matched states. Hence, a lower mean-squared error also implies an on average higher approximation accuracy. As we aim at mixing the classifiers in a way to increase the overall approximation accuracy, making the mixing weight of a classifier inversely proportional to its approximation error seems intuitively correct.

In XCS (Wilson 1995), the first accuracy-based LCS, the accuracy of classifier k is separately calculated by some inverse power-function of the approximation error, and then additionally scaled relative to the accuracy of classifiers that match some states in S_k, which introduces scaling of the mixing weights w.r.t. the approximation error of classifiers that share the same area in the state space. Such additional scaling might have a smoothing effect on the mixing weights but from the mixing point-of-view it is not strictly necessary. Note that in XCS, there is no explicit separation between the use of accuracy for mixing classifiers, and as a measure of a classifier's fitness for application in the evolutionary component of XCS. In our opinion, those two values do not necessarily have to be equal, and

currently we do not aim at redefining the fitness of a classifier but rather concentrate exclusively on the mixing weights. We define these weights closely related to how YCS (Bull 2005) (a simplified version of XCS) defines the fitness of a classifier, that is

$$\psi_k(i) = \frac{I_{S_k}(i)\varepsilon_k^{-\nu}}{\sum_{p=1}^{K} I_{S_p}(i)\varepsilon_p^{-\nu}}, \tag{20}$$

where $\nu \in \mathbb{R}^+$ is a positive constant that allows additional control on the mixing weights.[18] It is easy to check that a mixing weight defined like that satisfies the constraints given by our framework, that is for any $i \in S$, $\sum_{k=1}^{K} \psi_k(i) = 1$, and $\psi_k(i) = 0$ if $i \notin S_k$. Note that (20) is undefined for states that no classifier matches, but in such cases it is clear that we are unable to give an approximation.

7.2 Using the maximum likelihood estimate

In deriving the update equations for the Kalman filter, we have modeled the weight estimate by a Gaussian, given by $N(w_k, \Sigma_k)$. The same can be said for the observation of V for state i, which can be modeled by a Gaussian with mean $\tilde{V}_k(i) = w_k'\phi(i)$ and the estimated variance of $\tilde{V}_k$, as given by $\phi(i)'\Sigma_k\phi(i) + \Xi_k$ (see Appendix B.1 for derivation). Having a model of the observation for each classifier, we can derive a mixed model of all classifiers by using the Maximum Likelihood Estimate (MLE).

The MLE defines the value of highest likelihood by the maximum of the *likelihood function*, which is the product of the probability density function of the models of all classifiers. As deriving the logarithm of the likelihood function is usually easier and gives the same maximum, we will use this *log-likelihood*, denoted by Λ, to derive the maximum. Let us fix a state $i \in S$ and define $\mu_k = w_k'\phi(i)$ as the mean and $\sigma_k = \sqrt{\phi(i)'\Sigma_k\phi(i) + \Xi_k}$ as the standard deviation of the Gaussian model for classifier k.[19] Then, substituting for the probability density function of a Gaussian, the log-likelihood Λ for our approximation $\tilde{V}(i)$ is given by

$$\begin{aligned}\Lambda(\tilde{V}(i)) &= \sum_{k=1}^{K} I_{S_k}(i) \ln\left(\frac{1}{\sqrt{2\pi}\sigma_k} \mathrm{e}^{\frac{(\tilde{V}(i)-\mu_k)^2}{2\sigma_k^2}}\right) \\ &= \sum_{k=1}^{K} I_{S_k}(i) \ln\left(\frac{1}{\sqrt{2\pi}\sigma_k}\right) + \frac{1}{2}\sum_{k=1}^{K} I_{S_k}(i)\frac{(\tilde{V}(i)-\mu_k)^2}{\sigma_k^2}.\end{aligned}$$

Adding $I_{S_k}(i)$ is equivalent to taking the probability density function of classifier k to the power of $I_{S_k}(i)$, and ensures that we ignore classifiers that do not match state i. We can get the unique maximum of the convex log-likelihood function by setting its first derivative w.r.t. $\tilde{V}(i)$ to zero, which results in

$$\tilde{V}(i) = \sum_{k=1}^{K} \mu_k \frac{I_{S_k}(i)\sigma_k^{-2}}{\sum_{p=1}^{K} I_{S_p}(i)\sigma_p^{-2}},$$

[18] Such power factors have also been used in XCS and YCS, but in both cases there is no separation between the classifier fitness and the mixing weights. In our study, ν exclusively concerns classifier mixing and does not influence the fitness.

[19] To be able to calculate the MLE we have to assume the independence of the models given by the classifiers. Considering dependence between the classifier models would not allow us to derive a closed-form solution to the MLE.

giving the value for state i with the highest likelihood. Hence, the most likely estimate of the function value for state i is the sum of the approximation of each matching classifier, weighted by the normalized inverse variance of that classifier.

When taking a closer look at the value model variance $\phi(i)'\Sigma_k^w\phi(i) + \Xi_k$, we can see that it consists of the variance caused by the uncertainty of the value estimate by the classifier weight, $\phi(i)'\Sigma_w^w\phi(i)$, and the measurement noise variance Ξ_k. The uncertainty of the estimate decreases with the number of observations, which causes the value estimate to be judged as being more certain and have a higher influence, as more observations are made. The noise variance should be independent of the number of observations and gives a bias to the value model variance. Therefore, combining both adjusts classifier mixing by (i) the time and state-dependent certainty of the classifier's approximation and (ii) the final quality of its approximation.

An underlying assumption of our model is that the measurement noise can be modeled by a Gaussian. In our case, the measurement noise represents the deviation of the function values of V from our linear architecture, about the distribution of which we cannot make any predictions. In the value model variance, the Gaussian nature is partially expressed by the state-dependent term $\phi(i)'\Sigma_k^w\phi$. We currently cannot give any qualitative comments about the influence of this term on the overall estimate, but assume that it is of advantage to functions where the deviation is indeed expressible by a Gaussian, and of possible disadvantage to functions where that is not the case. However, this statement is speculative and definitely needs further investigation.

For now, let us drop the term $\phi(i)'\Sigma_k^w\phi$ from the value model variance, and let it be defined by $\sigma_k = \sqrt{\Xi_k}$. Then the above expression for the most likely value estimate becomes

$$\tilde{V}(i) = \sum_{k=1}^{K} w_k'\phi(i)\frac{I_{S_k}(i)\Xi_k^{-1}}{\sum_{p=1}^{K} I_{S_p}(i)\Xi_p^{-1}}.$$

By noting that ε_k is the approximation for Ξ_k, and by combining (5) and (20), we can see that the above is equivalent to using classifier mixing as in (20) with $\nu = 1$. Thus, using $\nu = 1$ gives the MLE of the value estimate, given that the value variance is approximated by the approximation error ε_k.

7.3 Investigating parameter ν

From the previous section we could assume that setting $\nu = 1$ gives the best overall approximation. In this section we will show that this might indeed be a good value, but that setting ν is actually not that clear-cut.

By its definition (see (20)), ν influences the spread in weighting between classifiers with different accuracy. The higher ν, the more importance is given to a low approximation error. In the limiting case, which is $\nu \to \infty$, the mixing weights are set so that only the approximation of the seemingly most accurate classifier is considered, that is

$$\lim_{\nu\to\infty} \psi_k(i) = \lim_{\nu\to\infty} \frac{I_{S_k}(i)\varepsilon_k^{-\nu}}{\sum_{p=1}^{K} I_{S_p}(i)\varepsilon_p^{-\nu}} = \begin{cases} 1 & \text{if } k = \operatorname{argmin}_{p\in\{1,\ldots,K\}}\varepsilon_p, \\ 0 & \text{otherwise.} \end{cases}$$

Setting it to 0 weights all matching classifiers equally, independent of their accuracy.

Intuitively, setting $\nu = \infty$ only considers the classifier with the lowest error and should therefore give the best approximation. This error, however, is the *average* error over the classifier's matched states and does not tell us anything about the approximation error for

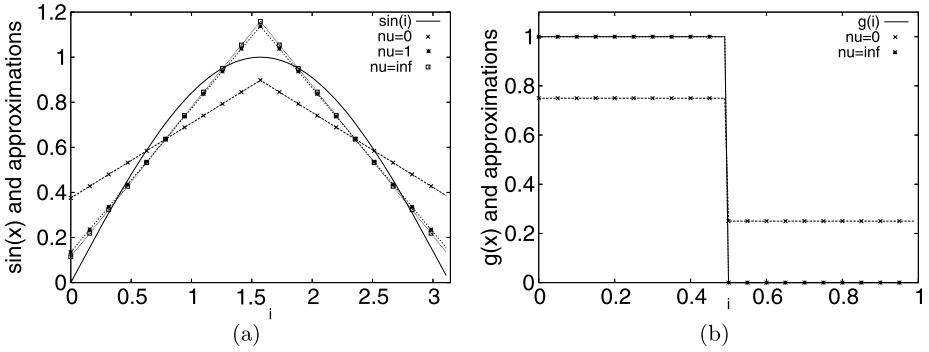

Fig. 3 Applying different values of mixing parameter ν when approximating **a** a sinusoid in range $[0, \pi)$, and **b** a step function $g(i)$

particular states. Hence, in the worst case, the classifier could happen to only have the lowest error within a set of states for which its approximation is actually worse than that of other classifiers. In such a case, more moderate settings for ν might be appropriate.

Let us consider two simple examples, as shown in Fig. 3: In the first example we want to approximate $V(i) = \sin(i)$ over the range $[0, \pi)$, using three classifiers with matched states sets $S_1 = [0, \frac{\pi}{2})$, $S_2 = [\frac{\pi}{2}, \pi)$, and $S_3 = [0, \pi)$. Its approximation for $\nu = 0$, $\nu = 1$ and $\nu = \infty$ is shown in Fig. 3(a). The second example concerns the step-function

$$g(i) = \begin{cases} 1 & \text{if } i < 0.5, \\ 0 & \text{otherwise,} \end{cases}$$

which we approximate by another 3 classifiers which match $S_1 = [0, 0.5)$, $S_2 = [0.5, 1)$, and $S_3 = [0, 1)$ respectively. Its mixed approximation is shown in Fig. 3(b) for the mixing parameters $\nu = 0$ and $\nu = \infty$. Both graphs show that $\nu = 0$ might never be a good idea, as it ignores the approximation error of the classifiers. The case is not as clear-cut for values $\nu > 0$, particularly around $i = \frac{\pi}{2}$ for the sinusoid, where some value $0 < \nu < 1$ might give the best approximation. In the case of the step-function, classifiers 1 and 2 are completely accurate ($\varepsilon_1 = \varepsilon_2 = 0$) and therefore dominate the mixing for any $\nu > 0$. This is an extreme case and might not happen frequently, but is always appropriate, as zero-error classifiers do not have any approximation error for any of its matching states.

To summarize, $\nu = 0$ ignores the accuracy of a classifier and might therefore lead to rather inaccurate overall approximations. On the other hand, $\nu = \infty$ might overemphasize low-error classifiers in certain cases and increase the discontinuity in the approximation. In general, the topic requires further investigation, but to our current knowledge, $\nu = 1$ actually seems to give the best compromise between the two extremes.

8 Discussion and future work

In this paper, we have introduced parts of a formal framework that aims at studying learning classifier systems, concerning how a set of classifiers approximate a function separately and in combination. The notation and structure of the framework is aligned to allow for relation of the separate components of LCS to common Machine Learning techniques, and is inspired by the one presented in (Bertsekas and Tsitsiklis 1996).

Firstly considering single classifiers, we have defined their optimality by minimizing the squared approximation error over all states that the classifier matches. As minimizing the cost function (3) or (4) is formally equivalent to a weighted least squares problem with weights $I_{S_k}(i_m)$, all approaches to solving this problem are technically incremental learning approaches to a weighted least squares problem. Alternatively, one can derive the same problem statement by assuming a probabilistic formulation with constant noise variance and seeking the maximum likelihood solution (Bishop 2006), as demonstrated here by the equivalence of the Kalman filter and RLS update equations. Either way, we end up with the same update equations, using either slowly converging gradient-based approaches, or preferably, RLS or similar methods that provide the optimal least-squares solution with each additional update.

While we have discussed the relation of introduced incremental weight updates to existing methods throughout the text, let us here take a summarized view: LCS that do not train their classifiers independently of each other, such as those described in (Booker 2006; Kovacs 2002; Wada et al. 2005; Wilson 1994), are not covered by our framework, for reasons given in Sect. 2.3. XCS and its derivates, on the other hand, train their classifiers independently and can therefore be easily related to the methods discussed in this paper: while the original XCS (Wilson 1995) uses (10) with averaging classifiers, that is $\phi(i) = (1)$ for all $i \in S$, for its weight update, we have for the first time explicitly identified that the error that this update equation minimizes is given by (4). Similarly, XCSF (Wilson 2002) implicitly minimizes the same cost function by using the NLMS update equation (4.3). The cost function was for the first time explicitly described in (Lanzi et al. 2005d) to introduce the RLS algorithm to update the weight vector in XCSF, but without including the matching term $I_{S_k}(i_m)$. Matching was introduced, as in XCS, by only updating the weight vector of classifiers that match the current state. While this is an algorithmically valid approach, our derivation from first principles explicitly shows that all update methods have the same underlying linear model and minimize the same cost function.

With the introduction of the Kalman filter we have shown that under the assumption of constant measurement noise, the Bayesian update of the multivariate Gaussian weight vector model results in the same equations as the RLS algorithm, and therefore equally minimizes the matching-weighted sum of squared errors. In addition, we have gained probabilistic interpretations of the different parameters involved in the classifier model, such as that the model's prediction given a new state i is given by the Gaussian $N(w_k'\phi(i), \phi(i)'\Sigma_k\phi(i) + \Xi_k)$. Note that the variance of this prediction is on one hand formed by the noise variance estimate Ξ_k, and on the other hand by the uncertainty of the weight vector model $\phi(i)'\Sigma_k^w\phi(i)$. Thus, we can now judge the confidence of a prediction made by a classifier not only based on the global approximation error, but also based on our certainty about the weight vector estimate. This property needs to be further explored to provide confidence measures for the global approximation.

In contrast to the large amount of research activity seeking to improve the weighted vector estimation methods in XCS, its method of estimating the classifier model quality based on the absolute error rather than the squared error was left untouched since its initial introduction. We have firstly questioned its consistency here, as the squared error measure (rather than the absolute error) is not only more consistent with the weight vector update methods, but also allows us to formulate a method that directly tracks this error. Used as a drop-in replacement for the mean absolute error measure in XCSF, we have shown in a follow-up study that it, indeed, improves the generalization capabilities due to its higher accuracy of evaluating the model quality of a classifier (Loiacono 2006).

With respect to mixing the classifiers, that is, to combine their prediction to form the prediction of the global function approximation, we have recently performed additional work

in (Drugowitsch and Barry 2006d, 2007a) that confirms that accuracy-based mixing indeed outperforms XCS mixing, and a set of other alternative heuristics. In more detail, we have additionally investigated linear mixing models that do not rely on the principle of weighted averaging, and some other quality measures for the classifier prediction. In a set of function approximation experiments, we have shown that while the linear mixing models might be able to outperform mixing by weighted average (and that includes accuracy-based mixing) when only a small number of classifiers is used, they do not scale well computationally with the number of classifiers, and are unable to maintain their performance lead when the number of classifier rises. Mixing as done in XCS was identified as being significantly worse than all other methods, and the accuracy-based mixing introduced here provided the best overall performance.

How does the work performed in this study help us to complete the framework by additionally considering classifier replacement and reinforcement learning? Concentrating first on classifier replacement, one has to ask the question how to identify a desirable set of classifiers? Taking the least squares error or maximum likelihood as a performance measure would result in an optimal set of classifiers being given by a single classifier per input sample, as in that way all samples can be approximated completely error-free. However, to avoid such over-fitting, we need to introduce methods that balance over- and under-fitting of the global approximation. Borrowing from the field of Model Selection, we have addressed this problem by embedding the function approximation framework introduced in this paper in a fully Bayesian formulation, and have then applied Bayesian model selection to identify how well a particular set of classifiers explains the given samples (Drugowitsch and Barry 2007b, 2007c). More explicitly, given a set of classifiers by their number and location in the input space, their parameters can be tuned by conditioning their parameter priors on the given samples, just as done for a single classifier by the Kalman filter. This allows us to derive an expression of the probability of the samples given the current classifiers, and—by applying Bayes' rule—also an expression for the probability of the given set of classifiers given the current samples. Thereby, the task of finding a good set of classifiers is equivalent to increasing their probability, given the available samples. This approach is appealing as it is conceptually clear, and implicitly prevents over-fitting of the model (MacKay 1991). Currently, we can only provide a batch implementation which requires that all available samples are available at once. How to turn this approach into an incremental learner such that we can replace classifiers dynamically and still increase the probability of the whole set is the topic of future research.

With respect to embedding the adaptive function approximation of LCS in reinforcement learning, we have provided a preliminary analysis in (Drugowitsch and Barry 2006b, 2006c). The main concern is that the interaction between the function approximation architecture and the different reinforcement learning algorithms can lead to instabilities. Q-Learning, for example, is known to be unstable even with the simplest linear function approximation architectures. We can provide stability guarantees by showing that the LCS function approximation architecture satisfies certain functional properties, that we will not discuss in detail here. While (Drugowitsch and Barry 2006b, 2006c) do provide a detailed overview of the required analysis, it still lacks conclusive results. Nonetheless, it points out the direction that future work has to concentrate on.

9 Conclusion

By providing a formal framework for function approximation in Learning Classifier Systems we have for the first time shown that the classifiers are independently trained localized

regression models of the target function, where their localization is determined by their matched states sets. Their training is a weighted least squares problem, for which we have derived both batch and incremental learning methods from first principles. We have shown how to combine the localized models to a global model by the principle of maximum likelihood, resulting in an inverse-variance weighted combination of the local models.

The introduced framework forms the basis of our work that aims at capturing all component of LCS to provide a sound theoretical foundation for these systems, enable the development of new, better understood, methods, and bridge the gap between LCS and other closely related machine learning methods. It has already been of significant value in our follow-up work that has moved us notably closer to the goal of a unified LCS framework and analysis.

Acknowledgements We would like to thank Pier Luca Lanzi and Daniele Loiacono for comments on the first draft of this paper, and for pointing us to the randomized ANOVA procedure. We also thank the anonymous reviewers for comments and suggestions.

Appendix A: Notation

The following table gives an overview of the meaning of the different symbols and in which section they are introduced:

Symbol	Stands for	Section
V	Function to approximate, $V : S \to \mathbb{R}$	2.1
$\tilde{V}$	Approximation of V	2.1
$\tilde{V}_t$	Approximation of V at time t	2.1
$\tilde{V}_k$	Approximation of V by classifier k	2.2.1
S	State space, domain of V	2.1
i_t	State observed at time t, $i_t \in S$	2.1
π	Function that gives the sampling distribution over S	2.1
L	Size of the feature and weight vector	2.1
ϕ_l	lth basis function, giving lth feature of some state	2.1
$\phi(i)$	feature vector $\in \mathbb{R}^L$, giving features of state $i \in S$	2.1
K	number of classifiers in the population	2.2.1
S_k	Set of states that classifier k matches, $S_k \subseteq S$	2.2.1
I_{S_k}	Indicator function for set S_k	2.2.1
w_k	Weight vector $\in \mathbb{R}^L$ of classifier k	2.2.1
$w_{k,t}$	Weight vector w_k at time t	2.2.2
f_k	Mean-squared error of classifier k	2.2.2
$\varepsilon_{k,t}$	Sample-based error of classifier k at time t	2.2.2
$c_{k,t}$	Number of states matched by classifier k up to time t	2.2.2
$\mathbb{E}_k$	Expectation taken w.r.t. classifier k	2.2.2
$\psi_k(i)$	Mixing weight for classifier k in state i	2.2.3
α	Step size for gradient-based algorithms	4.1
λ_k	Eigenvalues of the feature vector correlation matrix	4.1
$W_{k,t}$	Model of system state w.r.t. classifier k	5.1.1
$\Sigma^w_{k,t}$	$L \times L$ weight covariance matrix	5.1.1
$\xi_{k,t}$	Measurement noise for classifier k at time t	5.1.1
$\Xi_{k,t}$	Measurement noise variance of $\xi_{k,t}$	5.1.1

Appendix B: Derivations and proofs

The derivations and proofs in this section are given in their short form to present the idea of the derivation/proof rather go through each step in full detail. For full detail see (Drugowitsch and Barry 2006a).

B.1 Kalman filter in covariance form

The Kalman filter is based on repeatedly conditioning the system state model given by $W_{k,t-1} = N(w_{k,t}, \Sigma^w_{k,t})$ on the measurements $V_t = N(V(i_t), \Xi_{k,t})$. As the system state and the measurement are jointly Gaussian, the Bayesian update is given by (e.g. Anderson and Moore 1979, Chap. 3)

$$\begin{aligned} w_{k,t} &= \mathbb{E}(W_{k,t-1}|V_t = N(V(i_t), \Xi_{k,t})) \\ &= \mathbb{E}(W_{k,t-1}) + \mathrm{cov}(W_{k,t-1}, V_t)\mathrm{var}(V_t)^{-1}(V(i_t) - \mathbb{E}(V_t)), \\ \Sigma_{k,t} &= \mathrm{cov}(W_{k,t-1}|V_t = N(V(i_t), \Xi_{k,t})) \\ &= \mathrm{cov}(W_{k,t-1}, W_{k,t-1}) - \mathrm{cov}(W_{k,t-1}, V_t)\mathrm{var}(V_t)^{-1}\mathrm{cov}(V_t, W_{k,t-1}). \end{aligned}$$

Based on the definition of $W_{k,t}$ and the system model (11) we get the following expectations, variances and covariances:

$$\begin{aligned} \mathbb{E}(W_{k,t}) &= w_{k,t}, \\ \mathrm{cov}(W_{k,t}, W_{k,t}) &= \Sigma^w_{k,t}, \\ \mathbb{E}(V_t) &= w'_{k,t-1}\phi(i_t), \\ \mathrm{var}(V_t) &= \phi(i_t)'\Sigma^w_{k,t-1}\phi(i_t) + \Xi_{k,t}, \\ \mathrm{cov}(W_{k,t-1}, V_t) &= \Sigma^w_{k,t-1}\phi(i_t), \\ \mathrm{cov}(V_t, W_{k,t-1}) &= \phi(i_t)'\Sigma^w_{k,t-1}. \end{aligned}$$

To perform the conditioning only for measurement of states that the classifier matches, we replace $\phi(i_t)$ by $\sqrt{I_{S_k}(i_t)}\phi(i_t)$ and $V(i_t)$ by $\sqrt{I_{S_k}(i_t)}V(i_t)$. This change is justified by observing that including the function I_{S_k} in the state error $I_{S_k}(i_t)(V(i_t) - w'_{k,t-1}\phi(i_t))^2$ causes the same modifications. The effect is that non-matching states seem to have zero error, as $\sqrt{I_{S_k}(i_t)}\phi(i_t) = \sqrt{I_{S_k}(i_t)}V(i_t) = 0$.

The final form of the Kalman filter in its Covariance Form, given by (12), (13), and (14), is found after some algebraic manipulation, by substituting the expressions for the expectations, variances, and covariances in the Bayesian update equations.

B.2 Kalman filter in inverse covariance form

The Inverse Covariance Form can be derived by applying the Matrix Inversion Lemma (e.g. (Haykin 2002, Chap. 9.2)) to the weight covariance update (14) with (12) being substituted for $\zeta_{k,t}$, which leads us straight to the Inverse Covariance Form of the weight covariance update (15).

Getting the weight update (16) requires some more steps: Firstly, by combining (12) and (14) we can express the Kalman gain by

$$\zeta_{k,t} = I_{S_k}(i_t)\Sigma^w_{k,t}\phi(i_t)\Xi^{-1}_{k,t},$$

which together with (13) gives

$$w_{k,t} = w_{k,t-1} - I_{S_k}(i_t)\Sigma^w_{k,t}\phi(i_t)\Xi^{-1}_{k,t}\phi(i_t)'w_{k,t-1} + I_{S_k}(i_t)\Sigma^w_{k,t}\phi(i_t)\Xi^{-1}_{k,t}V(i_t).$$

From this we get the final weight update (16) by pre-multiplying above by $(\Sigma^w_{k,t})^{-1}$ and substituting (15) for the first $(\Sigma^w_{k,t})^{-1}$ on the right-hand side of the resulting equation.

B.3 The recursive least squares algorithm

The derivation in this section is based on (Haykin 2002, Chap. 9) and (Bertsekas and Tsitsiklis 1996, Chap. 3). We aim at finding and iterative update that minimizes (17) at every time t. To find the minimum of the error we find the gradient of (17) and set it to 0, which results in

$$\left(\sum_{m=0}^{t} I_{S_k}(i_m)\Xi^{-1}_{k,m}\phi(i_m)\phi(i_m)'\right)w_{k,t} = \sum_{m=0}^{t} I_{S_k}(i_m)\Xi^{-1}_{k,m}V(i_m)\phi(i_m).$$

From (15) and $(\Sigma^w_{k,-1})^{-1} = 0$ we get

$$(\Sigma^w_{k,t})^{-1} = \sum_{m=0}^{t} I_{S_k}(i_m)\Xi^{-1}_{k,m}\phi(i_m)\phi(i_m)',$$

which is (19) of the RLS algorithm and part of the left-hand side of above optimality condition for the weight vector. That lets us derive

$$\begin{aligned}
(\Sigma^w_{k,t})^{-1}w_{k,t} &= I_{S_k}(i_t)\Xi^{-1}_{k,t}V(i_t)\phi(i_t) + \sum_{m=0}^{t-1} I_{S_k}(i_m)\Xi^{-1}_{k,m}V(i_m)\phi(i_m) \\
&= I_{S_k}(i_t)\Xi^{-1}_{k,t}V(i_t)\phi(i_t) + (\Sigma^w_{k,t-1})^{-1}w_{k,t-1} \\
&= I_{S_k}(i_t)\Xi^{-1}_{k,t}V(i_t)\phi(i_t) + (\Sigma^w_{k,t})^{-1}w_{k,t-1} \\
&\quad - I_{S_k}(i_t)\Xi^{-1}_{k,t}\phi(i_t)\phi(i_t)'w_{k,t-1} \\
&= (\Sigma^w_{k,t})^{-1}w_{k,t-1} + I_{S_k}(i_t)\Xi^{-1}_{k,t}\phi(i_t)(V(i_t) - w'_{k,t-1}\phi(i_t)),
\end{aligned}$$

which, pre-multiplied by $\Sigma^w_{k,t}$, gives the final weight update (18).

B.4 Incremental error update

As we assume a constant measurement noise variance Ξ_k for deriving the incremental error update, we will omit it from the update equation, which is equivalent to setting it to 1. This is valid as the noise variance is a constant factor in our minimization that does not influence the result.

The proof is based on the equalities

$$\begin{aligned}
\sum_{m=0}^{t} I_{S_k}(i_m)(w'_k\phi(i_m))^2 &= w'_{k,t}\left(\sum_{m=0}^{t} I_{S_k}(i_m)\phi(i_m)\phi(i_m)'\right)w_{k,t} \\
&= w'_{k,t}(\Sigma^w_{k,t})^{-1}w_{k,t} \\
&= w'_{k,t}\sum_{m=0}^{t} I_{S_k}(i_m)V(i_m)\phi(i_m),
\end{aligned}$$

that are used to manipulate the expression for $\varepsilon_{k,t}$ from the new applicable Principle of Orthogonality (Theorem 2). The manipulation is based on expressing $(c_{k,t} - L)\varepsilon_{k,t}$ in terms of $(c_{k,-1} - L)\varepsilon_{k,t-1}$ by reducing the scope of the sums from t to $t-1$. The algebraic details to get the final error update equation can be found in (Drugowitsch and Barry 2006a).

B.5 Optimal approximation for a sinusoid

Let us consider the function $V(i) = \sin(i)$ and feature vectors $\phi(i) = (1, i)'$, which are used by classifier k, matching range $S_k = [a, b)$ with $b > a$, to approximate function V by uniform sampling. According to Theorem 1, the optimal weight vector can be calculated by $w_k = \mathbb{E}_l(\phi\phi')^{-1}\mathbb{E}_k(V\phi)$. Hence, we require to know $\mathbb{E}(I_{S_k})$, $\mathbb{E}(I_{S_k}\phi\phi')$ and $\mathbb{E}(I_{S_k}V\phi)$.

As we only sample states from S_k, we have $\mathbb{E}(I_{S_k}) = 1$. For the other two expectations we get

$$\mathbb{E}(I_{S_k}\phi\phi') = \int_a^b \frac{1}{b-a}\begin{pmatrix}1\\ i\end{pmatrix}\begin{pmatrix}1 & i\end{pmatrix}\mathrm{d}i = \frac{1}{b-a}\begin{pmatrix}b-a & \frac{b^2-a^2}{2}\\ \frac{b^2-a^2}{2} & \frac{b^3-a^3}{3}\end{pmatrix},$$

$$\begin{aligned}\mathbb{E}(I_{S_k}V\phi) &= \int_a^b \frac{1}{b-a}\begin{pmatrix}1\\ i\end{pmatrix}\sin(i)\,\mathrm{d}i\\ &= \frac{1}{b-a}\begin{pmatrix}\cos(a) - \cos(b)\\ a\cos(a) - \sin(a) - b\cos(b) + \sin(b)\end{pmatrix}.\end{aligned}$$

By inverting $\mathbb{E}(I_{S_k}\phi\phi')$ we can calculate w_k, which gives

$$\begin{aligned}w_k = &\frac{2}{(b-a)^3}\begin{pmatrix}2(a^2+ab+b^2) & -3(a+b)\\ -3(a+b) & 6\end{pmatrix}\\ &\times\begin{pmatrix}\cos(a) - \cos(b)\\ a\cos(a) - \sin(a) - b\cos(b) + \sin(b)\end{pmatrix}.\end{aligned}$$

To get the mean-squared error $f_k(w_k)$, we again apply Theorem 1 to get

$$\begin{aligned}f_k(w_k) &= \mathbb{E}_k(V^2) - w_k'\mathbb{E}_k(\phi\phi')w_k\\ &= \frac{1}{2(b-a)}(\cos(a)\sin(a) - a - \cos(b)\sin(b) + b)\\ &\quad - \frac{w_k'}{b-a}\begin{pmatrix}\cos(a) - \cos(b)\\ a\cos(a) - \sin(a) - b\cos(b) + \sin(b)\end{pmatrix}.\end{aligned}$$

References

Anderson, B. D. O., & Moore, J. B. (1979). *Information and system sciences series. Optimal filtering*. Englewood Cliffs: Prentice-Hall.

Barry, A. M. (2002). The stability of long action chains in XCS. *Journal of Soft Computing*, *6*(3–4), 183–199.

Barry, A. (2003). Limits in long path learning with XCS. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, & H.-G. Beyer et al. (Eds.), *Lecture notes in computer science: Vol. 2724. Genetic and evolutionary computation (GECCO-2003)* (pp. 1832–1843). Berlin: Springer.

Barry, A., Holmes, J., & Llora, X. (2004). Data mining using learning classifier systems. In L. Bull (Ed.), *Applications of learning classifier systems*. Berlin: Springer.

Bernadó, E., Llorà, X., & Garrell, J. M. (2001). XCS and GALE: a comparative study of two learning classifier systems with six other learning algorithms on classification tasks. In *Proceedings of the 4th international workshop on learning classifier systems (IWLCS-2001)* (pp. 337–341).

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont: Athena Scientific.

Bishop, C. M. (2006). *Pattern recognition and machine learning*, *information science and statistics*. Berlin: Springer.

Booker, L. (2006). Personal communication, Mai 2006.

Bull, L. (2002). On accuracy-based fitness. *Journal of Soft Computing*, *6*(3–4), 154–161.

Bull, L. (Ed.). (2004). *Applications of learning classifier systems*. Berlin: Springer.

Bull, L. (2005). Two simple learning classifier systems. In L. Bull & T. Kovacs (Eds.), *Studies fuzziness and soft computing: Vol. 183. Foundations of learning classifier systems*. Berlin: Springer.

Butz, M. (2004). *Rule-based evolutionary online learning systems: learning bounds, classification, and prediction*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL.

Butz, M. (2005). Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In H.-G. Beyer, et al. (Eds.), *Proceedings of the genetic and evolutionary computation conference (GECCO-2005)* (Vol. 2, pp. 1835–1842). New York: ACM.

Butz, M. V. (2006). *Studies in fuzziness and soft computing: Vol. 191. Rule-based evolutionary online learning systems: a principled approach to LCS analysis and design*. Berlin: Springer.

Butz, M. V., & Pelikan, M. (2001). Analyzing the evolutionary pressures in XCS. In L. Spector, et al. (Eds.), *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)* (pp. 935–942). Los Altos: Kaufmann.

Butz, M., Kovacs, T., Lanzi, P.-L., & Wilson, S. W. (2004). Toward a theory of generalization and learning in XCS. In *IEEE transactions on evolutionary computation*.

Compiani, M., Montanari, D., Serra, R., & Simonini, P. (1990). Learning and bucket brigade dynamics in classifier systems. In S. Forrest, et al. (Eds.), *Emergent computation*, *proceedings of the ninth annual international conference of the center for nonlinear studies on self-organizing, collective, and cooperative phenomena in natural and artificial computing networks. A special issue of Physica D* (Vol. 42, pp. 202–212). Amsterdam: Elsevier Science.

Dixon, P. W., Corne, D. W., & Oates, M. J. (2002). A preliminary investigation of modified XCS as a generic data mining tool. In P. L. Lanzi, W. Stolzmann, & S. W. Wilson (Eds.), *Lecture notes in artificial intelligence: Vol. 2321. Advances in learning classifier systems* (pp. 133–150). Berlin: Springer.

Douglas, S. C. (1994). A family of normalized LMS algorithms. *IEEE Signal Processing Letters*, *SPL-1*(3), 49–51.

Drugowitsch, J., & Barry, A. M. (2006a). *A formal framework and extensions for function approximation in learning classifier systems* (Technical Report 2006-01). University of Bath, UK, January 2006.

Drugowitsch, J., & Barry, A. M. (2006b). *A formal framework for reinforcement learning with function approximation in learning classifier systems* (Technical Report 2006-02). University of Bath, UK, January 2006.

Drugowitsch, J., & Barry, A. M. (2006c). *Towards convergence of learning classifier systems value iteration* (Technical Report 2006-03). University of Bath, UK, April 2006.

Drugowitsch, J., & Barry, A. M. (2006d). *Mixing independent classifiers* (Technical Report 2006-13). University of Bath, UK, November 2006.

Drugowitsch, J., & Barry, A. M. (2007a). Mixing independent classifiers. In *Proceedings of the genetic and evolutionary computation conference (GECCO-2007)*, July 2007. New York: ACM.

Drugowitsch, J., & Barry, A. M. (2007b). *Generalized mixtures of experts, independent expert training, and learning classifier systems* (Technical Report 2007-12). University of Bath, UK, April 2007.

Drugowitsch, J., & Barry, A. M. (2007c). A principled approach to LCS. In *Proceedings of the tenth international workshop on learning classifier systems (IWLCS 2007)*, July 2007.

Eweda, E., & Macchi, O. (1987). Convergence of the RLS and LMS adaptive filter. *IEEE Transactions on Circuits and Systems*, *CAS-34*(7), 799–803.

Forrest, S., & Miller, J. H. (1990). Emergent behavior in classifier systems. In S. Forrest (Ed.), *Proceedings of the ninth annual international conference of the center for nonlinear studies on self-organizing, collective, and cooperative phenomena in natural and artificial computing networks. A special issue of Physica D* (Vol. 42, pp. 213–217). Amsterdam: Elsevier Science.

Graybill, F. A. (1961). *An introduction to linear statistical models* (Vol. 1). New York: McGraw-Hill.

Greenyer., A. (2000). The use of a learning classifier system JXCS. In: P. van der Putten & M. van Someren (Eds.), *CoIL challenge 2000: the insurance company case* (Technical report 2000-09). Leiden: Leiden Institute of Advanced Computer Science, June 2000.

Haykin, S. (1999). *Neural networks. A comprehensive foundation* (2nd ed). Upper Saddle River: Prentice Hall.

Haykin, S. (2002). *Information and system sciences series. Adaptive filter theory* (4th ed.). Upper Saddle River: Prentice Hall.

Hettich, S., & Bay, S. D. (1999). The UCI KDD Archive, 1999. http://kdd.ics.uci.edu.

Holland, J. H. (1985). Properties of the bucket brigade. In J. J. Grefenstette (Ed.), *Proceedings of the 1st international conference on genetic algorithms and their applications (ICGA85)* (pp. 1–7), July 1985. Pittsburgh: Lawrence Erlbaum Associates,

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering series D*, *82*, 35–45.

Kalman, R. E., & Bucy, R. S. (1961). New results in linear filtering and prediction theory. *Transactions ASME, Part D (J. Basic Engineering)*, *83*, 95–108.

Koller, D., & Parr, R. (2000). Policy iteration for factored MDPs. In *UAI '00: proceedings of the 16th conference on uncertainty in artificial intelligence* (pp. 326–334). San Francisco: Kaufmann Publishers.

Kovacs, T. (2002). *A comparison and strength and accuracy-based fitness in learning classifier systems*. PhD thesis, University of Birmingham.

Lanzi, P. L., Loiacono, D., Wilson, S. W., & Goldberg, D. E. (2005a). Extending XCSF beyond linear approximation. In H.-G. Beyer, U.-M. O'Reilly, D. V. Arnold, W. Banzhaf, C. Blum, & E. W. Bonabeau (Eds.) *Proceedings of the genetic and evolutionary computation conference (GECCO-2005)* (Vol. 2, pp. 1827–1834). New York: ACM.

Lanzi, P. L., Loiacono D., Wilson, S. W., Goldberg, D. E. (2005b). *Generalization in the XCSF classifier systems: analysis, improvement, and extension* (Technical Report 2005012). Illinois Genetic Algorithms Laboratory, March 2005.

Lanzi, P. L., Loiacono D., Wilson, S. W., Goldberg, D. E. (2005c). *XCS with computed prediction in continuous multistep environments* (Technical Report 2005018). Illinois Genetic Algorithms Laboratory, May 2005.

Lanzi, P. L., Loiacono, D., Wilson, S. W., & Goldberg, D. E. (2005d). XCS with computed predictions in multistep environments. In Beyer, H.-G., O'Reilly, U.-M., Arnold, D. V., Banzhaf, W., Blum, C., & Bonabeau, E. W. (Eds.), *Proceedings of the genetic and evolutionary computation conference (GECCO-2005)* (Vol. 2, pp. 1859–1866). New York: ACM.

Littman, M. (2005). Neural information processing systems workshop: RL comparisons. http://www.cs.rutgers.edu/~mlittman/topics/nips05-mdp/.

Loiacono, D., Drugowitsch, J., Barry, A. M., & Lanzi, P. L. (2006). Improving classifier error estimate in XCSF. In *Proceedings of the 9th international workshop on learning classifier systems*.

MacKay, D. J. C. (1991). Bayesian interpolation. *Neural Computation*, *4*(3), 415–447.

Maybeck, P. S. (1979). *Mathematics in science and engineering: Vol. 141. Stochastic models, estimation, and control*, *Vol. 1*. New York: Academic Press.

Mook, D. J., & Junkins, J. L. (1988). Minimum model error estimation for poorly modeled dynamic systems. *Journal of Guidance, Control and Dynamics*, *11*(3), 256–261.

Nedić, A., & Bertsekas, D. P. (2003). Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, *13*(1–2), 79–110.

Piater, J. H., Cohen, P. R., Zhang, X., & Atighetchi, M. (1998). A randomized ANOVA procedure for comparing performance curves. In *ICML '98: proceedings of the fifteenth international conference on machine learning* (pp. 430–438). San Francisco: Kaufmann.

Riolo, R. L. (1987a). Bucket brigade performance, I: long sequences of classifiers. In J. J. Grefenstette (Ed.), *Proceedings of the 2nd international conference on genetic algorithms (ICGA87)* (pp. 184–195). Cambridge: Lawrence Erlbaum Associates.

Riolo, R. L. (1987b). Bucket brigade performance, II: default hierarchies. In J. J. Grefenstette (Ed.), *Proceedings of the 2nd international conference on genetic algorithms (ICGA87)* (pp. 196–201). Cambridge: Lawrence Erlbaum Associates.

Rummery, G., & Niranja, M. (1994). *On-line q-learning using connectionist systems* (Technical Report 166). Engineering Department, University of Cambridge.

Saxon, S., & Barry, A. (2000). XCS and the Monk's problems In P. L. Lanzi, W. Stolzmann, & S. W. Wilson (Eds.), *Leture notes in artificial intelligence: Vol. 1813. Learning classifier systems. From foundations to applications* (pp. 223–242). Berlin: Springer.

Smith, R. E. (1994). Memory exploitation in learning classifier systems. *Evolutionary Computation*, *2*(3), 199–220.

Sutton, R. S. (1996). Generalization in reinforcement learning: successful examples using sparse coarse coding. In D. S. Touretzky, M. C. Mozer, & M. E. Hasselmo (Eds.), *Advances in neural information processing systems* (Vol. 8, pp. 1038–1044). Cambridge: MIT Press.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: an introduction*. Cambridge: MIT Press.

Tsitsiklis, J., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*(5), 674–690.

Wada, A., Takadama, K., Shimohara, K., & Katai, O. (2004). Is gradient descent method effective for XCS? Analysis of reinforcement process in XCSG? In W. Stolzmann, et al. (Eds.), *Lecture notes in artificial intelligence*. *Proceedings of the seventh international workshop on learning lassifier systems*. Seattle, WA, June 2004. Berlin: Springer.

Wada, A., Takadama, K., Shimohara, K., & Katai, O. (2005). Learning classifier system with convergence and generalization. In L. Bull & T. Kovacs (Eds.) *Foundations of learning classifier systems: Vol. 183*. *Studies in fuzziness and soft computing*. Berlin: Springer.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, University of Cambridge, Psychology Department.

Welch, G., & Bishop, G. (2004). *An introduction to the Kalman filter* (Technical Report TR 95-401). University of North Carolina at Chapel Hill, Department of Computer Science, April 2004.

Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. In *IRE WESCON convention revord part IV* (pp. 96–104).

Williams, G. (2005). *Linear algebra with applications* (5th ed.). Boston: Jones and Bartlett.

Wilson, S. W. (1994). ZCS: a zeroth level classifier system. *Evolutionary Computation*, *2*(1), 1–18. http://prediction-dynamics.com/.

Wilson S. W. (1995). Classifier fitness based on accuracy. *Evolutionary Computation*, *3*(2), 149–175.

Wilson, S. W. (2001). Function approximation with a classifier system. In Spector, et al. (Eds.). *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)* (pp. 974–981). Los Altos: Kaufmann.

Wilson, S. W. (2002). Classifiers that approximate functions. *Neural Computing*, *1*(2–3), 211–234.

Wilson, S. W. (2004). Classifier systems for continuous payoff environments. In K. Deb, R. Poli, W. Banzhaf, H. G. Beyer, E. K. Burke, P. J. Darwen (Eds.), *Lecture notes in computer science: Vol. 3103*. *Proceedings of the genetic and evolutionary computation conference (GECCO-2004)* (pp. 824–835). Berlin: Springer.