Model Failure and Context Switching Using Logic-based Stochastic Models

Nikita A. Sakhanenko · George F. Luger

Abstract We define a notion of context that represents invariant, stable-over-time behavior in an environment and we propose an algorithm for detecting context changes in a stream of data. A context change is captured through model failure when a probabilistic model, representing current behavior, is no longer able to fit the newly encountered data. We specify stochastic models using a logic-based probabilistic modeling language and use its learning mechanisms to identify context changes. We also discuss how our algorithm can be incorporated into a failure-driven context-switching probabilistic modeling framework and demonstrate several examples of its application.

Keywords Probabilistic reasoning · Context · Failure-driven online learning

1 Introduction to Context-Based Diagnostics

In real-time diagnosis, where observations are given as a data stream, reasoning often has to be performed under strict time constraints with limited amounts of data available at each time step. This diagnostic problem can be simplified by a contextualization approach where the data stream is partitioned into stable regions (contexts) and a separate model is built for each context. The complexity of a model representing stable behavior under a context is often considerably reduced since most of the contextually irrelevant information is left out during modeling. Reduced models often require less training data. In this paper we define the notion of context capturing stable data patterns and propose an algorithm for detecting when these contextual patterns change. A context change is identified by model failure when the current model no longer fits

N.A. Sakhanenko*
Institute for Systems Biology,
Seattle, WA 98103 USA
Tel.: +1-206-732-1433
Fax: +1-206-732-1299
E-mail: nsakhanenko@systemsbiology.org

G.F. Luger
*Computer Science Department, University of New Mexico
Albuquerque, NM 87131 USA
E-mail: luger@cs.unm.edu

the incoming data. Our representation is based on a first-order logic-based probabilistic modeling language that combines the power of first-order logic with the ability to handle uncertainty and noise.

Probabilistic modeling systems that dynamically represent changing data are important for carrying out complex diagnostic reasoning tasks. With the increasing use of remote sensing technology continuously and in parallel collecting large sets of data, it becomes more necessary to develop a methodology for processing noisy data in a timely manner. Since modern sensing systems are often supported by very large sensor networks, the standard approach of collecting and processing all data at a central location is rarely efficient and it becomes necessary to shift aspects of the computation to the sensors where the data are collected. This introduces additional constraints on the running time and memory of the modeling system.

The most suitable systems in these cases, we believe, are those that are able to *evolve* to handle rapidly changing pieces of information. There is a limitation, however, that makes current probabilistic modeling unable to support this evolution: many approaches are *static*, namely, they assume that modeling is done only once and that the entire dataset is available ahead of time. In this paper we define *context* and introduce *failure-driven context-switching* probabilistic modeling that incorporates ideas from developmental learning, including assimilation and accommodation (Piaget, 1983), to model streams of data from dynamic environments.

In Section 2 we overview related research. In Section 3 we describe generalized loopy logic (GLL), a first-order logic-based reasoning language we employ to specify contextual models and to perform inferencing over them. In Section 4 we give a definition of context and describe the context-sensitive modeling problem underlying this research. In Section 5 we propose an iterative algorithm for online detection of context transitions. In Section 6 we show how our context detection algorithm can be incorporated as a component of a larger context-sensitive modeling system and provide several examples. Finally, in Section 7 we give ideas for future research and conclude.

2 Related Research

In this section we discuss the three areas of related research that support our effort: logic-based probabilistic reasoning, the notion of context, and failure-driven context switching.

2.1 Logic-Based Probabilistic Reasoning

Logic-based representations for stochastic modeling have been proposed by a number of researchers. Poole (1993) was one of the first to develop an approximate inference algorithm for a Turing complete probabilistic logic language where uncertainty is expressed through sets of mutually exclusive predicates annotated with probabilities. Haddawy (1994) created a first-order probabilistic logic that he used to specify a static class of (propositional) Bayesian networks (BNs) as a knowledge base. Haddawy proposed a provably correct Bayesian network generation algorithm that was later adapted to focus the knowledge base on the relevant information (Ngo and Haddawy, 1997; Ngo et al., 1997).

Friedman et al. (1999) and, later, Getoor et al. (2001) proposed probabilistic relational models (PRMs) that differ from other approaches (Ngo and Haddawy, 1997; Kersting and DeRaedt, 2000; Richardson and Domingos, 2006) by specifying a probability model using classes of objects rather than simple attributes. For example, an explicitly identified relational structure of PRMs (similar to relational DBs) supports probabilistic dependencies between attributes of related objects. PRMs (Friedman et al., 1999; Getoor et al., 2001) use maximum likelihood parameter estimation for parameter learning, while structure learning is done through a heuristic search of the best scores in a hypothesis space.

Bayesian logic programs (BLPs) is another knowledge-based model construction appoach proposed by Kersting and DeRaedt (2000). This framework generates Bayesian networks specific for given queries using a set of first-order Prolog-like rules with uncertainty parameters. Richardson and Domingos (2006) propose Markov logic networks (MLNs), a probabilistic approach based on general first-order logic. This approach converts logic sentences into a conjunctive normal form (CNF) which is then mapped onto Markov random fields for inference.

In this paper, we choose a significantly different direction than the approach of Richardson and Domingos (2006) using both domain-dependent and query-dependent model construction. Even though mapping from the CNF sentences of MLNs to Markov fields is straightforward, the practical advantages over Horn-clause-based representations are not obvious: we argue that Horn clauses provide expressive power by preserving the generality and in the same time supporting embedding various heuristics. We use a stochastic language, called Generalized Loopy Logic (GLL), described in detail in the next section, that combines Horn clauses with BNs similarly to BLPs (Kersting and DeRaedt, 2000). Other stochastic logic-based methods can be seen as examples of knowledge-intensive modeling, however these are static, assuming all the data are given at the start of problem solving, and thus cannot be applied efficiently to dynamic problems that often have strict time and memory constraints.

2.2 The Specification of *Context*

One of the first attempts to explicitly use contextual information in probabilistic modeling was done in (Haddawy, 1994; Ngo and Haddawy, 1997). Their logic-based stochastic modeling approach utilizes explicit contextual information as a way to reduce the size of a model. Sanscartier and Neufeld (2007) propose another approach that uses context to refine a probabilistic model. They use context-specific independence to make a causal Bayesian network smaller and more accurate.

Exploiting the notion of context defined through conditional independencies to improve the performance of a model was investigated earlier by Turney (1996). In the area of supervised machine learning, Turney studies how features from a multidimensional feature space can be partitioned into different categories using context. Silver and Poirier (2007) applied context to adapt multiple task learning neural networks for learning. Silver and Poirier replaced multiple outputs of a neural network with a single one while adding a set of inputs that identify an example context.

There are a number of attempts to formalize the notion of context in order to carry out proofs of correctness for operations of context-aware systems where the goal is to make the software aware of the environment and adaptable to changing situations. Akman and Surav (1996) point out that a causal link between two events is only

relevant given a certain background and, thus, only in certain contexts. Akman and Surav (1996) discuss two directions for research on context formalization, one based on a logic approach proposed by McCarthy (1993), and the other based on situation theory proposed by Barwise (1986). McCarthy and Buvac (1998) further examined formalizing contexts as first class objects. The authors argue that using contexts as formal objects allows for generalization of axiomatizations in limited contexts to overcome the original limitations. Barlatier and Dapoigny (2007) propose the use of a logical framework based on intuitionism and type theory supported by ontological knowledge representation in order to define context. They argue that there is a strong connection between situations and contexts that define relevant information depending on the user's task.

Although our research is motivated by (Haddawy, 1994; Ngo and Haddawy, 1997), their contextual mechanism is too simple and discrete: it cannot reflect all the complexity of internal structures of data. In this paper, we provide a formal specification of context as truth assignments to a specific set of variables that we know about. The choice of variables is similar to the approach of Pearl (2000) and Halpern and Pearl (2001) that uses exogenous variables (that are not in the model) to identify a background for the possible causes of an event.

2.3 Detection of Failure and Context Switching

The concept of social context states that in order to interpret a text, the social environment must be taken into account as it influences the author of the discourse. As opposed to objectivistic social context, van Dijk (2006) argues that the relevant features of communicative situations influence understanding only through participants' subjective views of the situations. These views are represented and constantly updated in mental models of the speakers, so-called *context models*.

In psychology, a mental model is an interpretation of how something works in a surrounding environment that plays an important role in cognition and decision-making. Luquet, who first proposed this idea in 1927, argues that internal models are constructed by problem solving children (Jolley, 2004). Luquet's view strongly influenced Piaget, who proposed a developmental theory of learning (Piaget, 1983). When an unfamiliar situation is presented to a child, she tries to fit it into her current understanding of the world. When this fails, the normal child is able to form new cognitive structures to address the situation. The theory recognizes two forms of learning: *assimilation* and *accommodation* (Piaget, 1983). Piaget suggests that new information from the environment creates a state of disequilibrium in the mind of an individual. There are two typical responses. First, the new thoughts are incorporated into an existing mental schema and there is a return to the state of equilibrium (assimilation). However, if the new thoughts are inconsistent with the existing schema, the schema must be changed (accommodation).

Gopnik et al. (2004) argue that knowing about causal structure permits humans to make wide-ranging predictions about future events. Gopnik et al. (2004) suggest that causal maps, which can be deduced from correlation patterns among events, can be used to represent causal knowledge and causal learning. It is unlikely that children store large amounts of data in memory and then apply a learning procedure to the data. More likely, they argue, children use small samples of data to form hypotheses. They then forget the data and revise their hypotheses as suggested by new data. Moreover, often causal regularities learned from one context constrain causal regularities that

are learned in other contexts, supporting *learning by analogy*. Gopnik et al. (2004) suggest that a model based on Bayesian networks that uses the principles of dynamic programming can support research on learning in children.

Granott et al. (2002) give another psychological perspective on human learning. The key notion of their work is *bridging*, which is, as in dynamical systems, an attractor that draws development of a system toward more advanced and more stable levels. Bridging mechanisms are carried out by partially defined *shells* directing the development of new knowledge by providing a perspective for processing new experiences. Granott et al. (2002) argue that bridging is a transition mechanism that people use while learning.

The failure-driven approach presented in this paper is motivated by many of these concepts from developmental learning. In the next section we introduce generalized loopy logic, our logic-based stochastic modeling language. In Section 4 we define the notion of context and describe context-sensitive modeling.

3 Generalized Loopy Logic

Generalized Loopy Logic (GLL) is a logic-based probabilistic reasoning language. GLL is based on earlier work by Poole (1993), Haddawy (1994), Getoor et al. (2001), and Kersting and DeRaedt (2000). GLL is an extension of the basic language developed by Pless et al. (2006). GLL is a logic-based, first-order, Turing-complete stochastic modeling language that improves expressive and reasoning power by combining deterministic and probabilistic approaches. Note that the expressive power of traditional Bayesian networks is constrained to finite domains as in the propositional logic. GLL handles this representational shortcoming through variables that can capture general classes of events and relationships. As a first-order language (Pless et al., 2006) it combines Horn-clause logic with Bayesian networks in order to represent potentially infinite classes of stochastic relationships including Markov processes. Knowledge is represented as a set of rules describing the conditional dependences among random variables with stochastic distributions attached to facts and rules.

Specifically, a sentence in GLL is of the form

$$\texttt{head|body}_1\texttt{, ..., body}_k\texttt{ = [p}_1\texttt{, ..., p}_l\texttt{]},$$

where $\texttt{body}_i$, $1 \leq i \leq k$ are the variables of the system on which a variable `head` is conditionally dependent. Note that the size of the conditional probability table (l) is equal to $arity(\texttt{head}) \times \prod_{i=1}^{k} arity(\texttt{body}_i)$, where $arity(\texttt{x})$ is the number of states of a variable `x`. The probabilities are indexed over the states of `head` and $\texttt{body}_i$, $1 \leq i \leq k$. For instance, if `x` is a predicate defined over $\{low, avg, hi\}$ and `y` is a boolean predicate, then the sentence

```
x|y = [[0.5, 0.1, 0.4],[0.3, 0.6, 0.1]]
```

defines $Pr(\texttt{x}|\texttt{y})$.

In GLL, terms can be full predicates with structure and contain Prolog style variables. For instance, the sentence `b(N) = [0.5,0.5]` says that `b` is universally equally probable to take on either of two values. The domain of terms is specified using set notation: `b <- {hi, low}` indicates that `b` is either `hi` or `low`.

The following GLL program defines a hidden Markov model (HMM) with four observable time steps:

```
state <- {true, false}
emit <- {hi, low}
state(N+1)|state(N)=[[0.9,0.1],[0.01,0.99]]
emit(N)|state(N)=Emit
emit(0) = hi
emit(1) = hi
emit(2) = low
emit(3) = low
```

In this example, there are two states, `true` and `false`. The system can start with either one and at each time step either stay in the same state or transition to the other state. Note that if the system is in the state `true`, then there is a 90% chance that the system will stay in that state at the next time step; however, if the system is in the state `false`, there is only a 1% chance the system will stay in that state. In both states the system can output either `hi` or `low`. Note that the probability of these events is a *learnable distribution* (`Emit`). Note also how the recursive rule of GLL captures the Markov process between states of the HMM.

The learnable distribution `Emit` indicates that the conditional probability governing the system's output is to be fitted. The data for learning is obtained from GLL rules and facts (observations). The last four sentences in the program presented earlier are the GLL facts. Note that in each fact the variable `N` is bound. Generalized Loopy Logic uses the message-passing inference algorithm known as *loopy belief propagation* (Pearl, 1988), hence the name "Loopy". As opposed to its predecessor (Pless et al., 2006), GLL can also use other iterative inferencing schemes including *generalized belief propagation* and *Markov chain Monte-Carlo*.

To perform inference GLL converts (unrolls) its first-order program to a Markov randon field (Luger, 2009). Mapping into a Markov field handles the product distributions arising from goals that unify with multiple heads: if more than one rule unifies with the rule head, then the variable node is connected to more than one cluster node, which results in a product distribution. One feature of GLL is its support of dynamic contexts where models can be specified by using recursion and by controlling the depth of the unfolding of recursive rules when mapping into a Markov random field. Figure 1 demonstrates how the GLL program specifying an HMM presented earlier is converted into a bipartite Markov field. Here each ground instance of a GLL term corresponds to a variable node in the Markov field (ellipse), and each GLL rule with a probability distribution attached to it corresponds to a cluster node (rectangle).

During loopy belief propagation, nodes of a Markov field exchange messages that are initially set randomly. On update, a message from a cluster node C to a variable node V (a message E_1 in figure 1) is the product of the conditional probability table (called a local potential) at C and all the messages to C except the message from V. In the other direction, the message from a variable node V to a cluster node C (a message E_2 in figure 1) is the normalized product of all the messages to V except the message from C. This process, iterating until convergence, has been found to be effective for stochastic inference (Murphy et al., 1999) and when applied to an acyclic graph is proved to converge to an optimal solution (Pearl, 1988).

A major feature of GLL is its natural support for parameter learning by the assignment of learnable distributions to rules of a GLL program. These parameters are estimated using a variant of the Expectation Maximization (EM) algorithm (Dempster et al., 1977) implemented through the message passing of loopy belief propagation algorithm. EM estimates learning parameters iteratively, alternating between an ex-

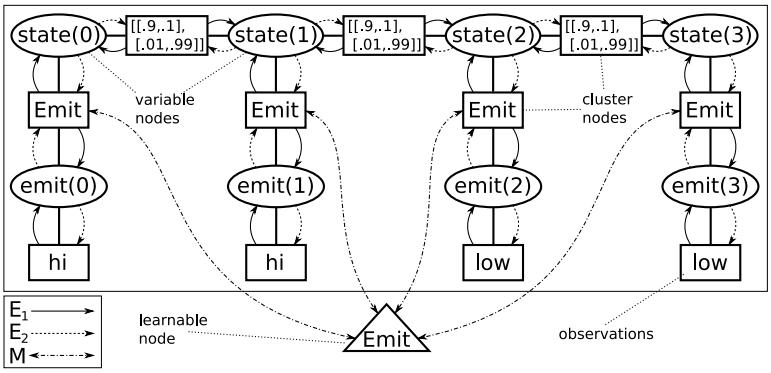


Fig. 1 A Markov random field produced by unrolling the GLL program defining a hidden Markov model.

pectation (E) step and a maximization (M) step. In the E step, the distribution for the hidden variables is based on their known value and the current estimate of the parameters is found. In the M step, these parameters are re-estimated. Assuming the distribution estimated in the E step is correct, each EM iteration increases the probability of reaching maximum likelihood (Dempster et al., 1977).

More specifically, GLL utilizes the EM algorithm by adding a special kind of node, a learnable node, to a Markov random field (the triangular node in figure 1). Each instance of the cluster node that is to be fitted is connected to the learnable node. By inferencing over the cluster and variable nodes of a Markov field (using loopy belief propagation) GLL computes the messages for the learnable nodes (a message M in figure 1). Applying the propagation algorithm until convergence is equivalent to the E step of the EM algorithm, since it produces an approximation of the expected values. The averaging over all the cluster nodes connected to the learnable node yields a maximum likelihood estimate of the parameters in a learnable node, which is equivalent to the M step of EM. Therefore, inferencing over the variable and cluster nodes followed by updating the learnable nodes and iterating this process is equivalent to the full EM algorithm.

4 Formal Specifications for Context-Sensitive Modeling

We next introduce the general problem of learning with context-sensitive probabilistic models by first introducing a formal notation. Italic uppercase letters (X, Y,Z) denote *variables*, and italic lowercase letters (x, y, z) represent their instantiated *values*. Similarly, the bold uppercase letters ($\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$) represent sets of variables, and bold lowercase letters ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$) denote their instantiations.

The *probability distribution of* a set of variables $\mathbf{X}$ is denoted with $Pr(\mathbf{X})$ whose elements are $Pr(\mathbf{x})$. For example, using this notation we can write $\sum_{\mathbf{x}} Pr(\mathbf{x}) = 1$. Similarly, $Pr(\mathbf{X} \mid \mathbf{Y})$ denotes the *conditional probability of* $\mathbf{X}$ *given* $\mathbf{Y}$, which is a table of probability distributions indexed by the instantiations of $\mathbf{Y}$: every $Pr(\mathbf{X} \mid \mathbf{y})$ is a probability distribution over $\mathbf{X}$, each element of which is depicted by $Pr(\mathbf{x} \mid \mathbf{y})$.

Selected features of knowledge and beliefs about a domain are encoded in a *model*, which is a partial view of total information about the domain. We use probabilistic graphical models (Pearl, 1988) as suitable representations for a model.

Definition 1 Given a (*universal*) set of variables $\mathbf{V}$, a *model* $\mathcal{M}$ imposed on $\mathbf{U} \subseteq \mathbf{V}$ is a graphical model defined on $\mathbf{U}$. Similar to Halpern and Pearl (2001), the variables in $\mathbf{U}$ are called *endogenous variables*, given $\mathcal{M}$, and denoted as $En(\mathcal{M})$. All the variables that are not in $\mathcal{M}$ are called *exogenous variables* and denoted as $Ex(\mathcal{M})$. Formally, we have $Ex(\mathcal{M}) = \mathbf{V} - En(\mathcal{M})$. Recall that $\mathcal{M}$ has a structural component, a graph $G_{\mathcal{M}}$, and a parametric component, a set of probability distributions $\Theta_{\mathcal{M}}$.

Definition 2 A conjunction of truth assignments to some exogenous variables of a model $\mathcal{M}$ is called a *context* $\mathcal{C}$ of $\mathcal{M}$: $\mathcal{C} \equiv V_1 \wedge \ldots \wedge V_n$, where $\{V_1, \ldots, V_n\} \subseteq Ex(\mathcal{M})$. Note that to make the definitions simpler, we assumed that all variables of our models are boolean; this can be relaxed by using general variable assertions instead of truth assignments.

The idea of a context is to capture the stable invariant behavior of the specified set of exogenous variables of a model: assuming the model fits a data set well, its context logically holds under the available data.

We next define a function estimation problem in an environment of changing situations. This can be seen as a multidimensional optimization problem: find an optimal collection of probabilistic models that represent a system in particular situations (contexts) accurately and efficiently. In general, the collection of contexts may not be known a priori, therefore we have to find an *optimal* set of contexts improving function estimation. Two properties of the set of contexts are accounted for during optimization: (a) context stability and (b) a rate of change of contexts. Consequently, we search for a set of contexts by minimizing the error representing how well each context from the set agrees with associated data and how many context changes are present. The search space is a collection of all possible sets of contexts.

Let $\mathbf{D}$ represent a set of observed data. Naturally we assume that the data set is ordered: $\mathbf{D} = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_m\}$, where each $\mathbf{d}_i$ is a vector of observations recorded at the ith time step ($i \leq m$) for all observable variables of the system. Given recent observations $\mathbf{d}_i$ for some $1 \leq i < m$, we refer to the successive data vector using the following notation: $s(\mathbf{d}_i) = \mathbf{d}_{i+1}$.

Consider a set of contexts $\mathbf{H} = \{\mathcal{C}_1, ..., \mathcal{C}_k\}$ from the search space. Each context from $\mathbf{H}$ represents invariant behavior in a possibly non-continuous subset of data. Therefore, $\mathbf{H}$ corresponds to some decomposition of a data stream. There are many possible decompositions of $\mathbf{D}$ into k mutually exclusive subsets, which we denote as $\rho(\mathbf{D})$. Consider an element $\rho_i \in \rho(\mathbf{D})$ that decomposes $\mathbf{D}$ into $\mathbf{D}_1, \ldots, \mathbf{D}_k$, where each $\mathbf{D}_j$ corresponds to observations of the stable behavior described by a context $\mathcal{C}_j$. Note that each $\mathbf{D}_i$ consists of data vectors that may not form a continuous time range of observations. We define two error scores associated with data decomposition ρ_i:

$$error'_j(\rho_i) = Pr_{\mathbf{x} \in \mathbf{D}_j}[\mathcal{C}_j(\mathbf{x}) = false],$$
$$error''_j(\rho_i) = Pr_{\mathbf{x} \in \mathbf{D}_j}[\mathcal{C}_j(s(\mathbf{x})) = false \mid \mathcal{C}_j(\mathbf{x}) = true],$$

where $\mathcal{C}_j(\mathbf{x})$ is an instantiation of context $\mathcal{C}_j$ on a data vector $\mathbf{x}$.[1] Informally, score $error'_j(\rho_i)$ indicates the error rate we expect when applying $\mathcal{C}_j$ to instances drawn

[1] $Pr_{\mathbf{x} \in \mathbf{D}_j}$ indicates that the probability is taken over the instance distribution D_j.

from the probability distribution $\mathbf{D}_j$. It captures how much context $\mathcal{C}_j$ disagrees with data set $\mathbf{D}_j$ from data decomposition ρ_i. Given a successful application of $\mathcal{C}_j$ to an instance, score $error''_j(\rho_i)$ indicates the expected error rate when applying $\mathcal{C}_j$ to the next instance. Note that when $error'_j(\rho_i)$ is minimal, $error''_j(\rho_i)$ denotes the amount of instability in the system's behavior described by context $\mathcal{C}_j$ and sampled with data $\mathbf{D}_j$. By summing these two scores across ρ_i we obtain an error score for the data decomposition given the context partition:

$$error(\rho_i) = \sum_{j=1}^{k} [error'_j(\rho_i) + error''_j(\rho_i)].$$

Minimizing $error(\rho_i)$ across all data decompositions yields a score $Error_{\mathbf{D}}(\mathbf{H})$ for a context set $\mathbf{H}$ given a data stream $\mathbf{D}$:

$$Error_{\mathbf{D}}(\mathbf{H}) = \min_{\rho_i \in \rho(\mathbf{D})} [error(\rho_i)].$$

Note that the problem of estimating the error score of $\mathbf{H}$ is essentially the problem of clustering the data according to some stable contiguous patterns.

By minimizing $Error_{\mathbf{D}}(\mathbf{H})$ over all possible sets of contexts we find an optimal collection of contexts that represents the stable invariant behavior (with the smallest number of context changes) of the observed system: $\min_{\mathbf{H}}[Error_{\mathbf{D}}(\mathbf{H})]$.

Recall that each element of $\mathbf{H}$ (a context $\mathcal{C}$) corresponds to some model $\mathcal{M}$: there is a connection between $\mathcal{M}$ and $\mathcal{C}$. We look at context $\mathcal{C}$ as a condition that *constrains* the set of all possible models (structurally and parametrically). In other words, $\mathcal{C}$ constrains $\Theta_{\mathcal{M}}$, the parameters of $\mathcal{M}$, and $G_{\mathcal{M}}$, the structure of $\mathcal{M}$.

We next search for a context partition that most accurately represents the data. Ideally, these models should be as small as possible to reduce the cost of inference over them. Therefore, while minimizing $Error_{\mathbf{D}}(\mathbf{H})$, we want to maximize the probability for each $\mathcal{C} \in \mathbf{H}$:

$$\max_{\mathcal{C} \in \mathbf{P}} [Pr(G_c \mid \mathbf{D})] \propto \max_{\mathcal{C} \in \mathbf{P}} [Pr(G_c) Pr(\mathbf{D} \mid G_c)].$$

The prior probability distribution $Pr(G_c)$ reflects our belief before seeing any data that the structure G_c imposed by the context $\mathcal{C}$ is correct. Simultaneously, we minimize the structural complexity of a model to ensure that the structure of models is parsimonious:

$$\min_{\mathcal{C} \in \mathbf{P}} [size(G_c) + \max_{V \in G_c} [degree(V)]].$$

Here $size(G_c)$ stands for a number of edges in G_c, and $degree(V)$ denotes the number of other vertices connected to V by edges (minimizing the fan-in/fan-out problem). If there is no dependency between contexts and models (if contexts do not constrain corresponding models), then the entire optimization problem described above can be reduced to a traditional structure search and parameter estimation for a single model.

In the next section we analyze the model failure phenomenon and introduce algorithms for context transitions.

5 The Detection of Context Transitions

Since contexts correspond to invariant behavior of a system over periods of time, modeling context yeilds a very focused representation of a specific operational mode of the system. It is important to note that context change is closely related to model failure, i.e., to a new situation when a model no longer fits recent data. A model corresponding to an active context is less robust to context changes than the full model of the system. During a context transition event, when the observed data undergoes a significant qualititative or quantitative change, the current model fails. Thus, we consider model failure to be an indication of a context transition event.

In this section we demonstrate that the failure-driven approach is suitable for switching between contextual models. If the data change is not severe, a modification of the parameters of the current model can account for the new data. Otherwise we must remove the present model as it is no longer relevant, store it, and then assume that we are operating in a new context. Thus, contextualization and failure detection are used to perform context-sensitive probabilistic modeling.

5.1 Failure-driven Model Revision

Our approach to the probabilistic modeling of changing contexts is based on ideas from developmental human learning, see section 2.3 for more details. We argue that probabilistic inference systems will benefit greatly by emulating these mechanisms.

Our failure-driven context-switching approach addresses two related and common problems in machine learning: the problems of *over-fitting* and *over-generalization*. When single models are learned on a data set that is not diverse, models tend to become too specific and are said to over-fit and unable to generalize and account for slightly varying datasets. The converse problem of over-generalization is when a very general model is learned from well distributed and possibly sparse data in the learning stage and, therefore, performs badly on all types of data in the operational stage. When used in probabilistic systems, the mechanisms of assimilation and accommodation along with the notion of context and context change, helps minimize these problems.

Context switching mechanisms that swap the models during context-sensitive modeling employ these two forms of learning within a *failure-driven approach*. When new data are available, we check whether the current model fits the dataset well. If it does, the data are incorporated into the model by updating its probability distribution. Otherwise, if the model fails to fit the data, we save the current model and choose a new version that accounts for the new data. Here learning by assimilation happens when the model is consistent with new data and it is fine-tuned by assimilating the dataset. Learning by accommodation is when the model is inconsistent with new data, and, in order to account for the dataset, we have to reorganize our model.

5.2 An Example of Model-Failure and Context-Revision

Identification of model failure is crucial in context sensitive modeling. Assuming a continuous stream of data, the notion of failure represents the situation when new data are inconsistent with the current model. Essentially, model failure can be identified by estimating the likelihood of the data given the current model. When this likelihood

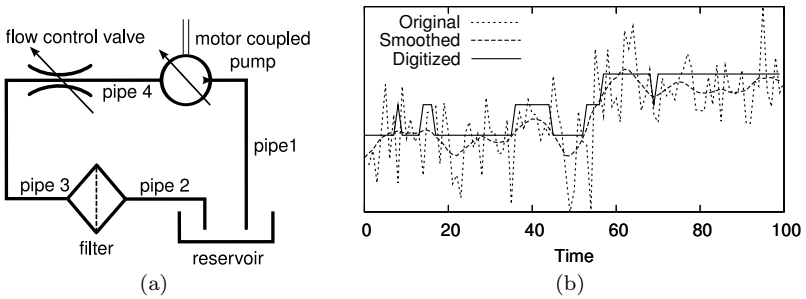


Fig. 2 (a) The diagram representing a simplified pump system. (b) The time series of the pressure generated by the pump (*OutPr*) and its smoothed and digitized versions.

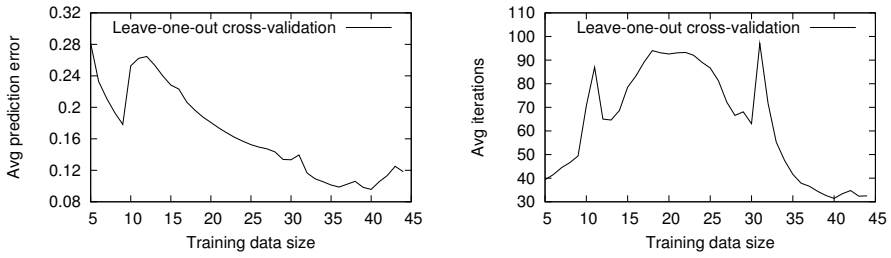
is below a certain threshold, then the model fails. Even though the models of the probabilistic system described in this paper are specified by the first-order stochastic language – Generalized Loopy Logic (Pless et al., 2006), the notion of failure can be extended to any probabilistic graphical model specifying a full joint probability distribution (see section 4 definition 1).

To illustrate our failure detection method we consider temporal data obtained from multiple sensors installed on the *mechanical pump system* schematically depicted in figure 2(a). A water pump draws liquid from a reservoir through a pipe (`pipe1`) and ejects the liquid into another pipe (`pipe4`). The pump is driven by an electrical motor. The liquid, containing contaminants is cleared by a filter and then deposited back into the reservoir. The flow control modulates the liquid flow.

In order to diagnose the system, we install a number of sensors that detect current pressure, flow, the emission state of the liquid at different locations, as well as *indicating parameters* such as the rotation rate of the pump and vibration near the motor. One important task is to detect when the filter gets clogged leading to possible cavitation in the system. In order to perform such diagnostic tasks, the knowledge about the system is transformed into a stochastic model using the GLL tool.

The sensory data consists of a time series of three parameters: pressure coming into the pump (*InPr*), pressure generated by the pump (*OutPr*), and voltage at the motor driving the pump (*Volt*). In order to estimate the behavior of the pump system depending on how clogged the filter is, we control the valve regulating the amount of fluid coming into the pump (as opposed to literally contaminating the system). During the experiment the pump system starts normal operation with the valve fully open. As the time passes a certain point (around the 53d time step), we partially close the valve to limit the flow of the fluid coming into the pump. A series of 100 data steps is recorded during the experiment. Each signal is then smoothed using a sliding window and digitized. Figure 2(b) illustrates the time series of one of the paprameters (*OutPr*) of the pump system.

We selected 35 time steps to train a stochastic model, each time slice of which contains 2 hidden variables (resistance at the pump, *Resist*, and torque of the motor, *Torque*) and 3 observable variables (*InPr*, *OutPr*, *Volt*). To select an appropriate size of training data we performed a leave-one-out cross-validation for each model trained on the first K time steps of the training data (here K takes values between 5 and 45 since we know that the first 45 time steps came from the same stationary distribu-



(a) (b)

Fig. 3 A leave-one-out cross-validation analysis across models trained on data sets with gradually increasing size. (a) Dependance between a model prediction error and the size of the training data. The error is averaged across the range of predicted parameters as well as across iterations of the cross-validation. Notice that the error decreases as the window gets larger than 15, and the error is minimal at around 35. (b) The number of iterations a model takes on average to converge versus the size of training data. Notice the amount of iterations stabilized to a minimum when the size of the training data is greater than 35.

tion). Figure 3(a) shows that the average prediction error decreases as the size of the training dataset increases and becomes minimal at around 35. Recall that GLL uses the EM learning algorithm (Dempster et al., 1977) implemented using loopy belief propagation (Pearl, 1988) to learn model parameters. Figure 3(b) demonstrates the dependency of the iterations of the learning algorithm on the training dataset. The fact that learning the model from the training dataset with 35 time steps requires a considerably smaller number of iterations is another indicator of the appropriate size of the training dataset.

5.3 Detecting model failure

The problem of identifying model failure is a special case of a statistical problem of detecting the distribution change from a stream of observations (Pollak, 1985). There are a number of approaches to this problem (Dayanik et al., 2007; Song et al., 2007; Steyvers and Brown, 2006). In this paper we provide a method that naturally fits into the iterative framework of our context-sensitive probabilistic modeling. Table 1 outlines our failure detection algorithm. The idea of the algorithm is to monitor a

1.		model $M \leftarrow$ *train_model*(training_data)
2.		window_params $\leftarrow$ *find_window*(training_data, model M)
3.		threshold $\leftarrow$ *find_threshold*(training_data, model M, window_params)
4.		current_data $\leftarrow$ *slide_window*(window_params)
5.		*for each* (trigger_params of model M)
	5.1.	current_param $\leftarrow$ trigger_param
	5.2.	new_param $\leftarrow$ *learn_param*(model M, current_data)
	5.3.	difference $\leftarrow$ *frobenius_norm*(current_param, new_param)
	5.4.	*if* (difference $>$ threshold) *then* failure $\leftarrow$ true
6.		*if* (not failure) *then* go to 4

Table 1 An algorithmic description of failure detection.

selected subset of model parameters (triggers) and signal about model failure when the parameters of the previously trained model are considerably different than these learned with new data. Note that the function *learn_param* (in step 5.2) estimates a single parameter given new data and the model, with the rest of the parameters learned from the training data. This is different from the function *train_model* (in step 1) that estimates all model parameters given data.

In Table 1 failure detection, initialized in steps 1 through 3, is an ongoing iterative process: it checks for failure in the specified data window and, if no break-down is detected, slides the data window further along the data stream (steps 4 and 5). Note that not only is the detection algorithm in Table 1 controlled by the size of the data window and the size of the window shift, but it is also regulated by the threshold indicating model failure.

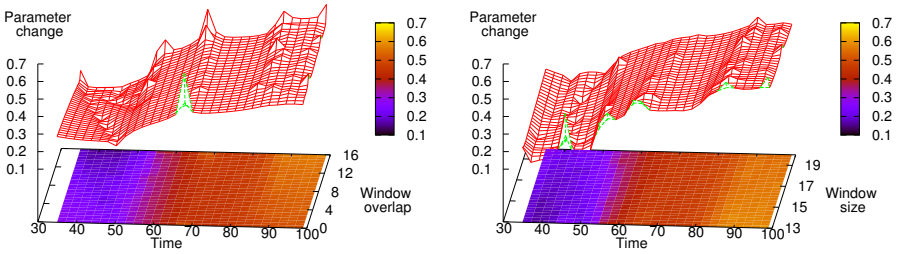
In general the problem of finding the appropriate window and threshold parameters can be seen as a two-dimensional error minimization problem. Consider $error_1 = Pr[FD(M, \Theta, D_{nofail}) = true]$, a Type I error representing the rate with which our failure detector FD signals about the failure of model M given window/threshold parameters Θ on the data (D_{nofail}) from the same stationary distribution (thus no failure is expected). $error_2 = Pr[FD(M, \Theta, D_{fail}) = false]$ is a Type II error that shows how frequently FD misses model failure. Ultimately, we would like to find parameters Θ that would minimize $error_1$ and $error_2$.

Minimizing $error_1$ is relatively easy: we partition the training dataset into two subsets, use the first subset to train the model, and employ the second subset to determine window parameters such that the failure detector finds no failure on the second subset. Note that the minimization of $error_1$ returns a subset of possible window/threshold parameters. Given the third subset of the training data on which the detector is *expected* to signal failure, we can perform a similar minimization routine to further constrain the parameter set.

Figure 4(a) shows the performance of the failure detector for a model (trained on data from the pump system) parameter corresponding to resistance of the pump (*Resist*). The detection algorithm slides a window of 17 data points through the data stream starting from the 35th time step (since we used the first 35 data points of the stream to train the model). The data stream has a real break-down at around time step 54, when a valve of the pump system is closed causing less flow coming into the pump and increasing pump resistance. Figure 4(a) shows that choosing an overlap between consecutive windows affects the choice of the threshold: selecting 0.3 as a threshold in case of overlap 12 accurately captures the break-down, however the same threshold does not work for overlap 10.

Figure 4(b) shows how the failure detector performance changes depending on the sliding window size, assuming the consequtive windows overlap by 12 points. It can be seen that the smaller the window, the more prone to data noise the failure detection becomes. On the other hand, larger windows produce smoother, more stretched out results.

We see from figures 4(a) and 4(b) that larger windows produce a failure detection *lag*, when model failure is identified long after the break-down has occured. Additionally, larger windows demand more computational power. On the other hand, smaller windows result in a higher likelihood of a false positive error.



(a) (b)

Fig. 4 Performance of the failure detector for a single model parameter (*Resist*) across (a) different window overlaps given a window of size 17, and (b) different window sizes given a window overlap of 12 data points. The X axis corresponds to time, the Y axis corresponds to the size of (a) the window overlap and (b) the sliding window, and the Z axis is the difference between the expected and the predicted values of the model parameter (*Resist* in this case) measured by the Frobenius norm (see footnote 2).

5.4 Constraining window/threshold parameters using variance

In general, without an appropriate data set, minimizing Type II error ($error_2$) is a challenging problem. The problem becomes even more difficult if the difference between distributions, transition between which creates model failure, is small. A possible way of selecting the window/threshold parameters without a training set for failure detection is to employ data variance.

Intuitively, we would like to know the size of a *representative* subset of the training data, a data window, variance of which is close to the true variance of the training data. A steep change in variance of such a data window would be a good indicator that the data came from a new distribution. Consider a window with size K and draw N subsets of data by randomly sliding the window along the training dataset. Computing an average variance over N data subsets for a large enough N produces an estimate of our confidence that a window of K elements drawn from the training dataset captures the underlying dependencies observed in the entire training dataset. Figure 5(a) shows the average variance of data windows with increasing size randomly selected from the training dataset. Figure 5(b) demonstrates that at some moment error bars of the variance monotonically decrease as the window size increases: the more data we take, the less changes in the data variation we get. Thus, we can set the window size to 12 or larger (25 is the optimal). Automatically, this can be done by selecting the window as soon as the error bars drop below a certain level, as the window size increases.

Once the window size is set, the failure threshold can be found by computing an average difference (Frobenius norm[2]) between the current value of a model parameter and its estimate computed from the window of the training data. Essentially, we can execute the failure detection algorithm using the window of training data and employ the computed difference as a failure threshold.

[2] A Frobenius norm of a matrix $A = (a_{ij})_{kl}$ is defined as $\|A\|_F = \sum_{i=1}^{k} \sum_{j=1}^{l} |a_{ij}|^2$.

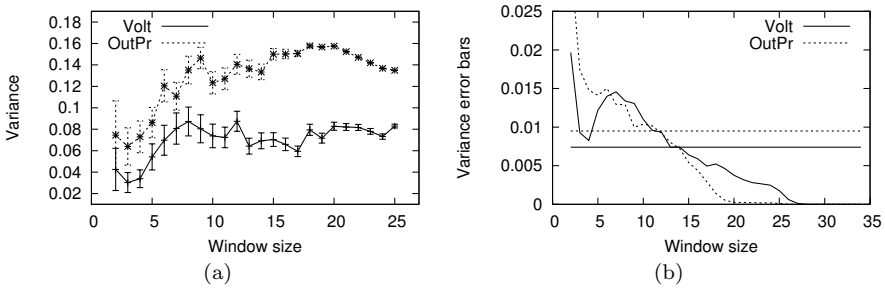


Fig. 5 Average variance (a) and the corresponding error bars (b) of subsets of the training data of the pump model plotted for various subset size (between 2 and 25). Two plots are shown for random variables *Volt* (voltage at the motor of the pump system) and *OutPr* (pressure coming out of the pump). We would like to select a window big enough for the changes in variance to be below the level depicted in (b) by the horizontal lines for each variable.

5.5 Application of failure detection

Figure 6 illustrates the performance of the failure detector on the sensory data for the pump model (plotted for three model parameters: motor voltage (*Volt*), pump resistance (*Resist*), and motor torque (*Torque*). Note that in this example each parameter has its own failure threshold, which brings more flexibility into the detection process, since some parameters change less gradually (such as *Volt*), while other deviate considerably (like *Torque*). The thresholds were automatically identified using the method described above.

Recall that the real model break-down happens around time step 54, when the valve of the pump system is partially closed. By monitoring the parameter *Resist* the failure can be identified at step 59 after 4 window shifts, where as by monitoring parameters *Volt* and *Torque* the failure is idetified much later, at about step 69.

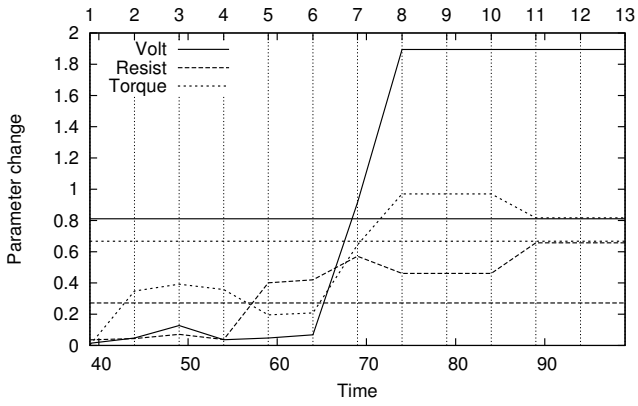


Fig. 6 Model failure detection for the pump model. Each horizonatal line corresponds to a failure threshold: once a corresponding distribution change goes above this threshold, the failure detector signals a model break-down. The grid corresponds to window shifts.

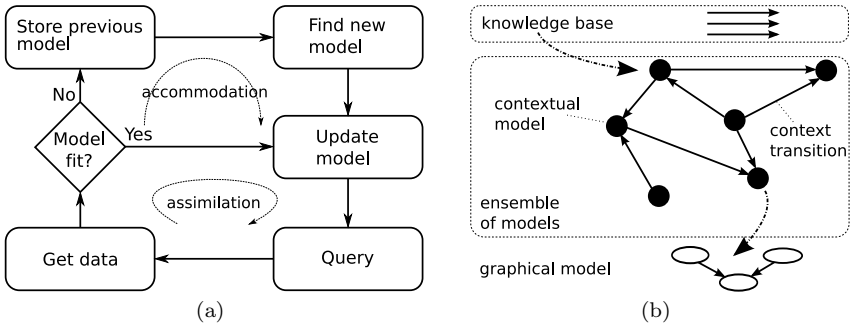


Fig. 7 (a) The flow chart of the failure-driven architecture. (b) A diagram of the relationship between domain knowledge, an ensemble of contextual models, and graphical models in a context-sensitive probabilistic modeling system.

When the model is large, failure detection in general can be very expensive. We specify a small subset of *trigger* parameters, whose changes are seen as most important by the domain experts and indicative of model failure. Instead of checking for failure in the entire model, only this small set of trigger parameters is monitored. Full-fledged failure detection is engaged once a change in a trigger parameter is discovered. Since different parameters give different detecting performance, it might be useful to employ a combination of these. Two-layerd failure detection can be used, for example, where a parameter that is sensitive to data noise but useful in detecting early failure (*Resist* in figure 6) can trigger an alert mode, in which case a more stable parameter (such as *Voltage*) is analyzed to confirm the detected model break-down.

In the next section we show how our method for the detection of context changes supports a framework for context-sensitive probabilistic modeling.

6 Applications to Dynamical Long-term Modeling

Probabilistic modeling systems that dynamically represent frequently changing data are important for monitoring complex tasks. Dynamical systems employing distributed sensing technology also introduce additional constraints on the running time and memory of the modeling system. The most suitable systems in these cases, we believe, are those that are able to *evolve* to handle rapidly changing pieces of information.

The failure-driven architecture of our context-sensitive probabilistic modeling system is described by the flow chart diagram in figure 7(a). Context switching mechanisms, which are among the main components of the system, employ two forms of learning within the architecture – *assimilation* and *accommodation*. When new data are available, the system checks whether the current model fits the dataset well. If it does, the data are incorporated into the model by updating its probability distributions. Otherwise, if the model still fails to fit the data, the system saves the current model and searches for a new version of the model that will account for the new data.

The operation of the system, see figure 7(a), when the condition "*Model fit?*" holds, corresponds to learning by assimilation: the model is consistent with the new data and it is fine-tuned by assimilating the dataset. Conversely, when this condition does not

hold, the system employs learning by accommodation: the model is inconsistent with new data, and in order to account for the dataset, we have to reorganize the model.

Thus a key component of our context-sensitive modeling system is an ensemble of contextual models, figure 7(b). Interconnected contextual models managed by domain knowledge, the top layer, correspond to vertices and edges of the structure of the ensemble. Graphical models, the vertices of the ensemble's structure, constitute the lowest layer. The system incrementally populates the ensemble of models by applying our failure-driven methodology. The implementation architecture may be understood as a production system. Each set of assertions to exogenous variables corresponds to a condition in a production system. When an assertion is satisfied, the set of activated probabilistic relationships consitutes a new model representing that context.

6.1 Testing: switching contexts in a pump system

The method for context change detection is implemented in Scheme. The prototype is applied to the data obtained from the mechanical pump system shown in figure 2. In particular, two types of tests were performed: *Test A*, a test on detecting a transition from a normal behavior to a context when the filter is clogged, and *Test B*, a test on choosing an appropriate model in the ensemble of models once the context change is detected.

In *Test A* an active model was trained on the data from the mechanical pump system operating normally. The normal behavior of the pump system continued until the 48th time step when the flow valve (see figure 2) was partially closed to imitate a clogged filter. The system was subsequently halted after a tatal of 100 data steps. During this time the active model was continuously checked for failure. The results for the three model parameters, voltage at the motor, resistance at the pump, and torque at the motor, are plotted in figure 8(a).

Failure was detected using a window of 17 time steps that was sliding 5 steps at a time. As seen in figure 8(a), the modeling system successfully identified the context change. Note however that the context change was captured after the actual breakdown had occured, at steps 55-60 as opposed to the 48th step, due to the choice of conservative window parameters from the set of possible parameters trained earlier. An attempt to capture the context change sooner by employing a smaller window (size 15) with a smaller overlap (10) was not successful (data are not shown): our failure detection method prematurely, around the 25th time step, signals a context change due to the high noise level in the data stream. A possibly more robust extension to the context switching mechanism might be to consider several windows of different sizes: small windows can be used to alert the system of a possible context change, while large windows can then confirm this change.

In *Test B* we trained 6 distinct models, corresponding to various operational contexts of the pump system. The contexts consist of normal operational behavior, behavior under a highly clogged filter, under a slightly clogged filter, behavior when a pump is misaligned, when a shaft between a motor and the pump is misaligned, and when one of the gears has a chipped tooth.

We then considered the data stream of the pump system operating normally until the 70th time step when the flow valve was almost closed (simulating a highly clogged filter) and reopened again after the 120th time step. The failure detection method was invoked for each of the six trained models. Figure 8(b) illustrates the results of

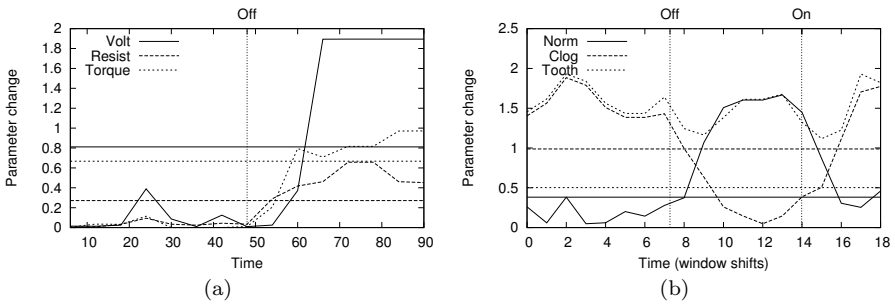


Fig. 8 (a) Failure detection on three model parameters (*Volt*, *Resist*, *Torque*) in the data stream when the pump system initially operates normally, but then breaks down at the 48th time step (when the flow valve is partially closed). The prediction is performed using a sliding window of 17 points with 12 point overlaps. A vertical line *Off* shows an actual system break down, while horizontal lines correspond to failure thresholds for the corresponding (same style) model parameters. Using *Resist* the context change is detected at the 55th step, using *Torque* the change is detected at the 60th step, and using *Volt* the change is detected at the 65th step. (b) An illustration of an experiment when the pump system starts operating normally and the valve is partially closed (at the *Off* time step), which is then opened back (at the *On* time step). The figure shows the difference between the true *Voltage* parameter and the parameter estimated on the sliding window for various models: a model (*Norm*) trained on data from regular conditions, a model (*Clog*) corresponding to a partially closed flow valve, and a model (*Tooth*) capturing the situation when a gear tooth is chipped.

failure detection for a model parameter corresponding to voltage at the motor (*Volt*) for three models *Norm*, *Clog*, and *Tooth*, corresponding to contexts of normal operational behavior, behavior when a flow valve is partially closed, and behavior when a gear tooth is chipped. Note that the model *Norm* shows the smallest difference between predicted and true model parameters before time *Off* (when the flow valve is turned), which then peeks after the turn, and drops down after time *On* (when the valve is reopened). On the other hand, the model *Clog* presents almost the opposite behavior: it shows a very large parameter difference before the turn of the valve, which steeply reduces once the valve is partially closed, and increases back after the valve is reopened. Note that other models do not show such distinct behavior, e.g., the model *Tooth* in figure 8(b) exhibits a constant large (above the corresponding threshold) difference between true and predicted parameters.

Since the model *Norm* has the smallest error at the beginning of the data stream, the context-sensitive modeling system employing our context change detection method selects this model as initially active. Once the first context change (*Off*) is identified, the system then switches the active model to *Clog*, since it has the smallest error among the six models. Consequently, the modeling system returns the active model back to *Norm* after the second context change is detected.

7 Conclusions and Future Directions

In this paper we specify the notion of context and propose a framework that uses model failure to represent a complex real-time diagnostic problem as a set of simpler context-specific knowledge-focused reasoning tasks. The framework uses a failure-

driven approach of switching between minimal models corresponding to contexts, which is motivated by research in developmental psychology (Piaget, 1983).

There are a number of important advantages of using the context-based failure-driven modeling approach to real-time diagnostic reasoning. First, there is no need to reconstruct a complete distribution of all possible data, thus less data is needed for training the diagnostic model. Usually, during probabilistic reasoning, system dynamics are assumed to be stationary and invariant over the *entire* training data set. While reasoning about a system whose dynamics change according to states of the external environment and where little a priori knowledge is given, every possible aspect of the world must be explicitly represented for the training data and learning algorithm to capture all hidden relationships.

One implication of minimal model context switching is that when a traditional knowledge base changes, the learned general model is discarded as no longer true and a new one must be constructed from scratch. Using our approach, by splitting the domain into contexts we are able to construct smaller models with reduced complexity capturing only relevant, currently present-in-the-data relationships. Such small models assume stationary behavior and require only a minimum amount of training data. This fact also helps to reduce the overfitting problem.

A second advantage of our approach is that non-stationary behavior is handled by using context transitions and the swapping of models. The different operational contexts of a system are captured by different small models representing local (referring to a context) stationary behavior in the data. Combining these models together by swapping a currently active model with another contextual model provides a way to handle global non-stationary behavior in the data.

Finally, for the domain expert, there can be more meaningful diagnoses due to domain focusing. Since the contextual models are knowledge-focused (representing only relevant information), the analysis of these models and their contextual differences can be much more meaningful to a domain expert.

There are a number of directions for future research. Even though it is linked to a probabilistic model, only deterministic context changes are currently allowed. A major extension would be to allow stochastic context transitions that will potentially increase reasoning power of our framework. Other directions of future research include extending the method for detecting model failure, e.g., by using several sliding windows of various sizes: a smaller window can be used to detect early failure, whereas a larger window can then confirm that the failure is not triggered by noise.

Interested readers can find the GLL software supporting failure-driven context-modeling at `http://www.cs.unm.edu/~sanik/Support/gll.tgz`.

Acknowledgements The research presented in this paper is part of the PhD dissertation of the first author under the supervision of the second. We thank Carl Stern of Management Sciences, Inc. and Roshan Rammohan, a PhD coleague at the University of New Mexico, for their continuous support and numerous discussions. We also acknowledge the advice of Tom Caudell and Lance Williams, who served on the PhD committee. This research was funded by an Air Force Research Laboratory SBIR contract (FA8750-06-C0016).

References

Akman, V. and Surav, M. (1996). Steps toward formalizing context. *AI Magazine*, 17(3):55–72.

Barlatier, P. and Dapoigny, R. (2007). Using Contexts to Prove and Share Situations. pages 448–453. AAAI Press. Proceedings of FLAIRS-07 Conference.

Barwise, J. (1986). Conditionals and Conditional Information. *On Conditionals*, pages 21–54.

Dayanik, S., Goulding, C., and Poor, H. V. (2007). Joint Detection and Identification of an Unobservable Change in the Distribution of a Random Sequence. *Information Sciences and Systems*, pages 68–73. Issue of 41st Annual Conference on Volume.

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B (Methodological)*, 39(1):1–38.

Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. (1999). Learning Probabilistic Relational Models. pages 1300–1307. Proc. of 16th Intl. Joint Conf. on AI (IJCAI).

Getoor, L., Friedman, N., Koller, D., and Pfeffer, A. (2001). Learning Probabilistic Relational Models. *Relational Data Mining*, pages 307–335.

Gopnik, A., Glymour, C., Sobel, D. M., Schulz, L. E., Kushnir, T., and Danks, D. (2004). A theory of causal learning in children: Causal maps and Bayes nets. *Psychological Review*, 111(1):3–32.

Granott, N., Fischer, K. W., and Parziale, J. (2002). Bridging to the unknown: a transition mechanism in learning and development. *Microdevelopment: transition processes in development and learning*.

Haddawy, P. (1994). Generating Bayesian Networks from Probability Logic Knowledge Bases. pages 262–269. Morgan Kaufmann. Proc. of 10th Conf. on Uncertainty in AI.

Halpern, J. and Pearl, J. (2001). Causes and Explanations: A Structural-Model Approach — Part 1: Causes. pages 194–202. San Francisco, CA: Morgan Kaufmann. Proc. of 17th Conf. on Uncertainty in AI (UAI-01).

Jolley, R. P. (2004). Review of G-H Luquet (1927, 2001) Children's Drawings (Le Dessin Enfantin). *History and Philosophy of Psychology*, 6:81–83. Translation and introduction by A. Costall.

Kersting, K. and DeRaedt, L. (2000). Bayesian Logic Programs. pages 138–155. Proc. of 10th Int. Conf. on ILP.

Luger, G. F. (2009). *Artificial intelligence: Structures and Strategies for Complex Problem Solving*. Addison-Wesley.

McCarthy, J. (1993). Notes on Formalizing Context. pages 555–560. Proceedings of 13th IJCAI.

McCarthy, J. and Buvac, S. (1998). Formalizing Context (Expanded Notes). *Computing Natural Language*, 81:13–50.

Murphy, K. P., Weiss, Y., and Jordan, M. (1999). Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *Uncertainty in Artificial Intelligence*, pages 467–475.

Ngo, L. and Haddawy, P. (1997). Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171(1–2):147–177.

Ngo, L., Haddawy, P., Krieger, R. A., and Helwig, J. (1997). Efficient Temporal Probabilistic Reasoning via Context-Sensitive Model Construction. *Computers in Biology and Medicine*, 27(5):453–476.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Pearl, J. (2000). *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Piaget, J. (1983). Piaget's theory. *Handbook of Child Psychology*, 1.

Pless, D. J., Chakrabarti, C., Rammohan, R., and Luger, G. F. (2006). The Design and Testing of a First-Order Stochastic Modeling Language. *International Journal on Artificial Intelligence Tools*, 15(6):979–1005.

Pollak, M. (1985). Optimal Detection of a Change in Distribution. *The Annals of Statistics*, 13(1):206–227.

Poole, D. (1993). Logic Programming, Abduction and Probability: a top-down anytime algorithm for estimating prior and posterior probabilities. *New Generation Computing*, 11(3–4):377–400.

Richardson, M. and Domingos, P. (2006). Markov Logic Networks. *Machine Learning*, 62(1–2):107–136.

Sanscartier, M. J. and Neufeld, E. (2007). Identifying Hidden Variables from Context-Specific Independencies. pages 472–477. AAAI Press. Proceedings of FLAIRS-07 Conference.

Silver, D. L. and Poirier, R. (2007). Context-Sensitive MTL Networks for Machine Lifelong Learning. pages 628–633. AAAI Press. Proceedings of FLAIRS-07 Conf.

Song, X., Wu, M., Jermaine, C., and Ranka, S. (2007). Statistical Change Detection for Multi-Dimensional Data. pages 667–676. Proceedings of the 13th Intnl. Conf. on Knowledge Discovery and Data Mining (KDD'07).

Steyvers, M. and Brown, S. (2006). Prediction and Change Detection. *Advances in Neural Information Processing Systems*, 18:1281–1288.

Turney, P. (1996). The Identification of Context-Sensitive Features: A Formal Definition of Context for Concept Learning. pages 53–59. Proceedings of Workshop on Learning in Context-Sensitive Domains at the 13th ICML.

van Dijk, T. A. (2006). Discourse, context and cognition. *Discourse Studies*, 8(1):159–177.