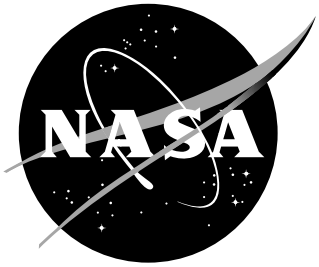
NASA/TM–2015–218927



ACCEPT: Introduction of the Adverse Condition and Critical Event Prediction Toolbox

Rodney A. Martin
NASA Ames Research Center, Mail Stop 269-1, Bldg. N-269, Rm. 260-17, P.O. Box 1, Moffett Field, CA 94035-0001, `rodney.martin@nasa.gov`*, ph. +1 (650) 604-1334, fax +1 (650) 604-3594*

Santanu Das
University Affiliated Research Center, currently Verizon
Vijay Manikandan Janakiraman
University Affiliated Research Center, NASA Ames Research Center, Mail Stop 269-1, Bldg. N-269, Rm. 260-23, P.O. Box 1, Moffett Field, CA 94035-0001, `vijaymanikandan.janakiraman@nasa.gov`
Stefan Hosein
NASA I^2 Internship Program, The University of the West Indies, St. Augustine Trinidad, `stefan.hosein2@my.uwi.edu`

November 2015

NASA STI Program ... in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

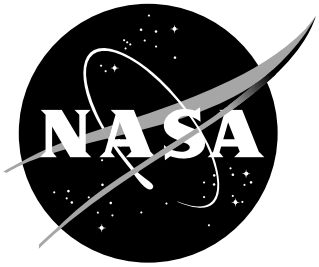
- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.
- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.
- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.
- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.
- TECHNICAL TRANSLATION. English- language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI Program, see the following:

- Access the NASA STI program home page at ***http://www.sti.nasa.gov***
- E-mail your question to help@sti.nasa.gov
- Phone the NASA STI Help Desk at 757-864-9658
- Write to:
  NASA STI Information Desk
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681–2199

NASA/TM–2015–218927



ACCEPT: Introduction of the Adverse Condition and Critical Event Prediction Toolbox

Rodney A. Martin
NASA Ames Research Center, Mail Stop 269-1, Bldg. N-269, Rm. 260-17, P.O. Box 1, Moffett Field, CA 94035-0001, `rodney.martin@nasa.gov`*, ph. +1 (650) 604-1334, fax +1 (650) 604-3594*

Santanu Das
University Affiliated Research Center, currently Verizon
Vijay Manikandan Janakiraman
University Affiliated Research Center, NASA Ames Research Center, Mail Stop 269-1, Bldg. N-269, Rm. 260-23, P.O. Box 1, Moffett Field, CA 94035-0001, `vijaymanikandan.janakiraman@nasa.gov`
Stefan Hosein
NASA I^2 Internship Program, The University of the West Indies, St. Augustine Trinidad, `stefan.hosein2@my.uwi.edu`

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, CA 94035-0001

November 2015

Acknowledgments

We would like to thank the Integrated Vehicle Health Management Project and the Systemwide Safety Assurance Technologies Project of the Aviation Safety Program for providing support during the period of time that this research was conducted. We would also like to thank Dr. Nikunj Oza for reviewing the paper, Dr. Kamalika Das for reviewing the paper and also providing valuable insights during discussions throughout the development of the ACCEPT framework, and finally Robert Lawrence for his invaluable domain expertise.

This report is available in electronic form at
http://

Abstract

The prediction of anomalies or adverse events is a challenging task, and there are a variety of methods which can be used to address the problem. In this paper, we introduce a generic framework developed in MATLAB® called ACCEPT (Adverse Condition and Critical Event Prediction Toolbox). ACCEPT is an architectural framework designed to compare and contrast the performance of a variety of machine learning and early warning algorithms, and tests the capability of these algorithms to robustly predict the onset of adverse events in any time-series data generating systems or processes.

Contents

Acronyms

ACCEPT	Adverse Condition and Critical Event Prediction Toolbox
MSET	Multivariate State Estimation Technique
SVR	Support Vector Regression
ELM	Extreme Learning Machine
BNN	Bagged Neural Network
k-NN	k-Nearest Neighbor
NMSE	Normalized Mean Squared Error
RMS	Root Mean Square
KS	Kolmogorov-Smirnov
PCA	Principal Components Analysis
DBM	Deep Boltzmann Machine
DBN	Deep Belief Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory Network
LDS	Linear Dynamical System
EM	Expectation-Maximization
N4SID	Numerical Algorithms for Subspace State-Space System Identification
AIC	Akaike Information Criterion
ROC	Receiver Operating Characteristic
AUC	Area Under the ROC Curve
SPRT	Sequential Probability Ratio Test

Nomenclature

k	Time index
T	Total number of time indices spanned by data
d	Prediction horizon
L	Critical threshold
$\mathbf{u}_k$	Feature vector
$\mathbf{u}_i$	Support vector
p	Number of features/parameters
z_k	Target parameter
v_k	Measurement Noise
$f(\cdot)$	Support Vector Regression (SVR) nonlinear mapping function
m	Number of support vectors
$\alpha_i, \hat{\alpha}_i$	Support vector weighting coefficients
ρ	SVR bias offset parameter
C	SVR user-specified regularization parameter
η	SVR regularization parameter
$\phi(\cdot)$	SVR image transformation
$\langle \cdot, \cdot \rangle$	Kernel function
σ	Kernel width
$\mathbf{q}$	SVR primal optimization variable

ξ_k^+, ξ_k^-	SVR slack variables
y_k	SVR residual and Linear Dynamical System (LDS) output
$\hat{y}_{k+i\|k}$	LDS output value prediction i time steps into the future
$\hat{z}_k$	Target parameter estimate
$\mathbf{w}_k$	Process/input noise
$\mathbf{x}_k$	State vector
$\mathbf{A}$	LDS system matrix
$\mathbf{C}$	LDS output matrix
$\mathbf{Q}$	Process noise covariance matrix
R	Measurement noise variance
$\hat{\mathbf{x}}_{k\|k}$	State estimate
n	State size
$\|\cdot\|$	Euclidean norm
$\succeq$	Positive semi-definite
$\mathcal{I}$	Universe of all possible events
$(\cdot)'$	Not (Set complement)
$(\cdot)^\top$	Transpose
$P(\cdot)$	Probability
$E[\cdot]$	Expected Value
$\mathcal{N}(\mu, \Sigma)$	Gaussian distribution with mean μ and covariance Σ
$\mathcal{N}(\mathbf{x}; \mu, \Sigma)$	Gaussian distribution evaluated at $\mathbf{x}$ with mean μ and covariance Σ

1 Introduction

One of the objectives we planned to achieve in the process of developing a framework to test adverse event prediction algorithms was to allow for the prediction to occur within a reasonable time horizon of an actual adverse event, with low false positive and missed detection rates. Thus, we introduce a generic framework developed in MATLAB®, ACCEPT (Adverse Condition and Critical Event Prediction Toolbox), which is explicitly geared to providing a comparative performance assessment of results from the application of a variety of algorithmic methods.

An initial step towards the goal of developing a new, state-of-the-art forecasting technology based upon this framework was presented in [1]. It is thus our aim in this paper to document continued steps towards achieving these goals, but with a broader focus on performance assessment. Here we use a variety of regression techniques in the context of a rigorous, formal analysis that involves cross validation. The regression techniques are used within an architectural framework to be described shortly, which act as a preprocessor for transformation of data into a form that is amenable to modeling for use in the context of applying hypothesis tests to the distribution formed by the resulting residual.

The rest of this paper will be organized as follows. Sec. 2 details ACCEPT's overall architectural framework. In Sec. 3 we review the architecture in the context of signal flow. In Sec. 4 we provide a discussion of optimization methods used for cross-validation and a detailed background of all selected regression methods. In Sec. 5, we discuss optimization for all detection methods through validation for the best possible performance in generation of the results. We then offer concluding re-

marks in Sec. 6. Details on various supplementary toolbox requirements are provided in Appendices A and B. Configuration and set-up of ACCEPT are provided in Appendix C. A developer's guide is provided in Appendix D. Finally, demonstration of ACCEPT on a sample problem is provided in Appendix E.

2 The Architectural Framework for ACCEPT

The architecture we use is patterned after MSET (Multivariate State Estimation Technique), as documented in [2]. MSET represents the current state-of-the-art in prediction technologies and is used ubiquitously in nuclear applications, as well as aviation and space applications [3]. The architecture of ACCEPT has the same basic structure as MSET (as shown in Fig. 1), but involves both a regression step and a detection step (shown in the dotted boxes, respectively). The regression step is typically implemented with the aid of a machine learning technique and the detection step tests a set of fixed hypotheses relating to the statistical properties of the resulting residual.

ACCEPT tests theoretical assumptions that differ from those made in MSET by appealing to the use of various detection and regression techniques used to parameterize the underlying models shown in the framework architecture (see Fig. 1). MSET is restricted to the use of a set of proprietary nonlinear kernel-based regression operators that yield specific properties necessary for the statistical hypothesis tests governing early detection. As such, for MSET, detection or classification which appeals to hypothesis testing methods is not model-based, nor does it rely on the use of machine learning techniques. In contrast, our aim is to consider detection techniques less restrictive in their assumptions.

As shown in Fig. 1, all data will be preprocessed and filtered using z-score normalization and feature selection, respectively. We employ to the idea of "boosting" in order to allow for the use of the residual generated from the base model by the Kalman filter. In Fig. 1, the regression toolbox shown within the dotted box on the left contains many alternative methods from which to generate the base model, which processes a select number of parameters (the "multivariate time series") and maps them to a distinct target parameter. This static nonlinear mapping characterizes the basic relationship between the features or input variables and the target parameter or response variable.

It is important to note that an unsupervised machine learning approach is employed for this architecture, meaning that no labeled data is used to supervise the process of model learning. As such, all training data associated with the regression step is by definition nominal data. Anomalous data is reserved solely for validation and testing purposes, and does not influence the model characterized by the regression step described above. In this way, two distinct classes of machine learning

algorithms, regression and classification, are employed within ACCEPT. Classification methods based upon hypothesis tests are used to determine if any novel, anomalous data is out of family with respect to the regression model characterizing the nominal training data.

2.1 Step 1: Regression Toolbox

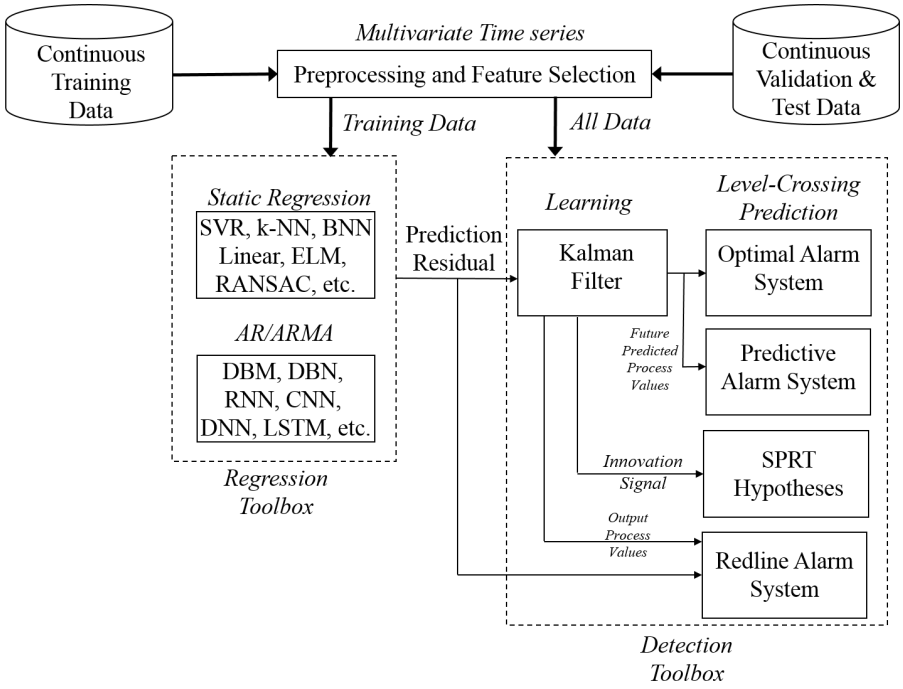


Figure 1: Functional Architecture, ACCEPT (Adverse Condition and Critical Event Prediction Toolbox)

The residual output from the regression toolbox quantifies the difference between the actual value of the target parameter and the value predicted by the base model[1]. The regression techniques are applied in the context of a rigorous, formal analysis that involves cross validation for an objective function that represents regression performance as quantified by the NMSE (Normalized Mean Squared Error) of resulting residuals. They also act as a preprocessor for transformation of data into a form that is amenable to modeling in the context of applying hypothesis tests to the distribution formed by the resulting residual.

All detection methods will conform to a rigorous process that is based upon examples of the candidate adverse event contained in validation datasets. This is followed by alarm system design and final testing with

[1]This mapping should have a functional basis specific to the adverse event scenario defined within the context of a particular application, for which any reasonably robust regression approach can be used.

a hold out data set, which in theory should be drawn from the same distribution as the validation data.

A single target parameter that acts as a global health indicator for a subject system or process rarely exists in reality. In fact, there may be multiple such indicators for each adverse event or anomalous operation that is a candidate for prediction. Thus, we may train as many regressors as there are available to characterize target parameters. It is equally important to ensure that the regressors are adequate predictors of the target parameters through a rigorous feature selection process that is based upon assessing the strength of correlations between these sets of parameters.

2.2 Step 2: Detection Toolbox

The dotted box on the right of Fig. 1 represents the detection portion of the architecture, which tests the residuals that are produced from Step 1. The residual may be distributed in such a way that is amenable to modeling as a Gaussian distribution. It can be used as the basis for learning a linear dynamical system, which is the first block in the detection portion of the architecture (labeled as "Kalman filter"), and needed for all but one of the prediction methods to be tested. The linear dynamical system implicitly characterizes the unmodeled dynamics unable to be captured by the regression method, which can subsequently be used for the design of an alarm system based upon different statistical hypotheses.

The optimal alarm system tests the level-crossing prediction hypothesis, *i.e.* extreme values associated with the tails of the distribution formed by the residual. The predictive alarm system also relies on using the model provided by the linear dynamical system. The other alarm systems requiring the linear dynamical system belong to the class of detection methods that test shifts in the first and second moments of the distribution formed by the residual. An empirical variant of the only other alarm system (the "redline alarm system") does not require the linear dynamical system, however, there is a model-based variant which does rely on it. Both variants are illustrated by two inputs into the block. Also, two equivalent variants are available for the optimal and predictive methods, although only one arrow is shown in the diagram for clarity.

3 Signal Flow

Fig. 1 is a *functional* representation of the architecture, meaning that the arrows in the diagram represent batch data transfer, rather than signal flow or real-time signal processing. This paradigm is used to emphasize the fact that learning takes place in two stages, due to the serial nature of the architecture. The architecture involves two distinct algorithms which both involve machine learning, and are necessary to fulfill the

desired objective of recasting the adverse event prediction problem into a form whose solution is accessible as a hypothesis-testing problem.

Fig. 2 depicts a signal flow representation of the problem. The linear dynamical system shown in the dotted portion of the diagram can loosely be thought of as an alternate representation of the regression block. More explicitly, it is trained to mimic the statistical properties of the regression residual (the unmodeled dynamics), such that $y_k = z_k - \hat{z}_k$, where y_k represents both the residual and the output at time k, and z_k represents a target parameter that characterizes a specifically designated adverse event or anomaly. Note that the regression block takes as input a multivariate vector of features, $\mathbf{u}_k$, and outputs an estimate of this target parameter, $\hat{z}_k$. Note also that the Kalman filter block takes as input the input or process noise, $\mathbf{w}_k$, and measurement noise, v_k.

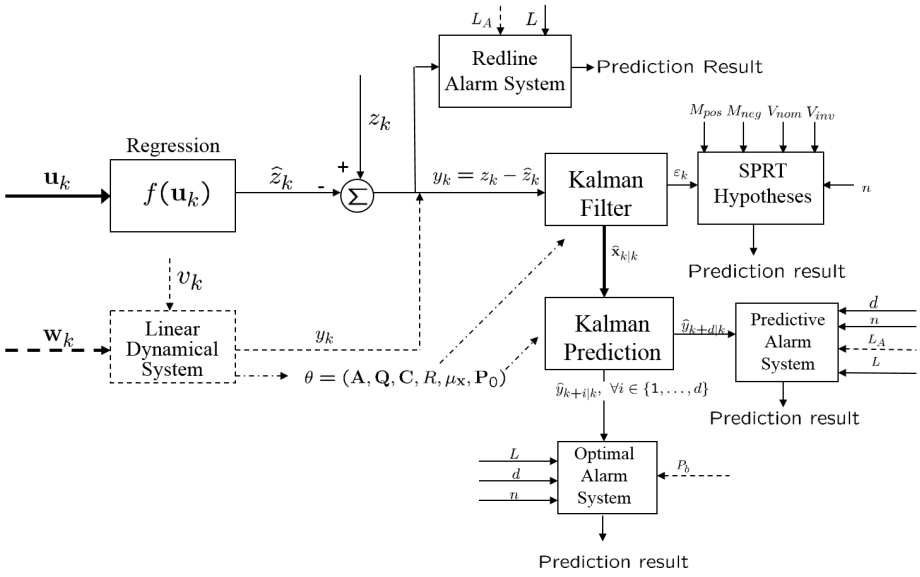


Figure 2: Signal Flow Diagram

The Kalman filter block accepts input from the regression block as y_k, processes these residuals as observables, and computes state estimates, $\hat{\mathbf{x}}_{k|k}$, based upon these observations during the correction step. These state estimates can then be used to form forward projected residual value predictions, $\hat{y}_{k+i|k}$, spanning the prediction horizon, $\forall i \in \{1, \ldots, d\}$. These are used either by the predictive or optimal level-crossing alarm system to initiate an alarm to forewarn of a level-crossing associated with the threshold value L. The alarm design parameters for both the predictive and optimal level-crossing alarm systems are the threshold L_A and border probability P_b, respectively. The redline alarm system can also be designed with the threshold L_A as an early warning for the more consequential level-crossing event associated with the threshold, L. Both alarm system design parameters are represented with dotted lines in Fig. 2. More detail on these detection methods will be provided in Sec. 5.

The Kalman filter block also generates an innovation signal, ε_k, which represents a secondary residual in the context of "boosting" mentioned previously. Due to the spectral properties of this innovation signal (*i.e.* whiteness), it can also be used for testing various Sequential Probability Ratio Test (SPRT) hypotheses. This class of prediction methods tests shifts in the first and second moments of the distribution formed by the innovation signal, however it requires the residual to exhibit the specific properties of whiteness and Gaussianity.

The SPRT tests, predictive, and optimal level-crossing event predictors, as well as the Kalman filter and Kalnab predictor all fundamentally use the LDS parameters, θ, which have been learned during the training process. The only detection method shown in Fig. 2 not using the linear dynamical system parameters is an empirical variant of the redline alarm system, which uses only the current residual value, y_k, and relies on use of the alarm design threshold L_A, similar to the one used for the predictive alarm system. Note that there are two variants for the redline, predictive, and optimal alarm systems, and the distinctions between these two will be provided in further detail in Sec. 5.

4 Regression Toolbox

Recall that there are a variety of regression techniques to be tested in the context of a rigorous, formal analysis that uses nominal training data which is partitioned into multiple folds for the purposes of f-fold cross validation. As previously mentioned, the NMSE will act as the objective function used for cross validation. It is a traditional metric for assessing the goodness-of-fit of a regression algorithm via the residual and is often a better choice than using the RMS (root-mean square) due to the ability to compare regression algorithms more equitably.

The ability to capture system dynamics is governed by the NMSE metric. For infinitesimally small NSME values, the spectral characteristics of the resulting residual should be white, and all of the dynamics embedded in the residual can be explained by the regression model. Thus, in a sense model fidelity characterizes the ability of the regression algorithm to learn all system dynamics (both linear and nonlinear). All prediction methods with the exception of the standard exceedance prediction method attempt to exploit the fact that the regression method will most likely not be able to explain all of the dynamics embedded in the residual. The residual itself will most likely contain linear dynamics that are driven by Gaussian noise. However it is still possible that the unexplained dynamics may contain nonlinearities which are unable to be captured, which is one limitation of assuming linear dynamics driven by Gaussian noise.

The NMSE governs model fidelity, but it has also been empirically observed that it is also heavily correlated to a metric called the Kolmogorov-

Smirnov (KS) statistic. The KS statistic is fundamentally a measure of Gaussianity of a distribution, by finding the maximum difference between an empirical cumulative distribution function based upon the residual and the normal distribution with mean and standard deviation given by the same residual data. This empirical observation will be studied in a more theoretically rigorous manner in a subsequent paper. Since the KS statistic governs a property assumed by almost all of the prediction methods to be tested, it is very convenient that it is very well correlated to the NMSE metric.

Formally, the optimization used for regression can be posed as shown in Eqn. 1, with the NMSE metric as the objective. The optimization problem shown here is essentially the result of a f-fold cross validation. Note that the index i is used for an individual validation time-series data record, where the index k is the time index, as previously referenced. A standard grid search is used to optimize the NMSE over the hyperparameter specific to the regression method being evaluated.

$$\begin{aligned} &\text{minimize} && J = \frac{\sum_{i=1}^{f}\sum_{k=1}^{T_i} y_{i,k}^2(\lambda)}{\sum_{i=1}^{f}\sum_{k=1}^{T_i} \tilde{z}_{i,k}} \\ &\text{subject to} \\ &\lambda \in \mathcal{S} \end{aligned} \tag{1}$$

where

$$\begin{aligned} f &= \text{number of folds used for cross-validation} \\ &\quad \text{(number of records in validation dataset)} \\ T_i &= \text{number of time points in } i^{th} \text{ time-series data record} \\ y_{i,k}(\lambda) &= z_{i,k} - \hat{z}_{i,k}(\lambda) \\ \tilde{z}_{i,k} &= z_{i,k} - \bar{z} \\ \bar{z} &= \frac{\sum_{i=1}^{f}\sum_{k=1}^{T_i} z_{i,k}}{\sum_{i=1}^{f} T_i} \\ \hat{y}_{i,k}(\lambda) &= f(\mathbf{u}_{i,k}, \lambda) \\ \lambda &= \text{Regression-specific hyperparameter} \\ \mathcal{S} &= \text{Regression-specific hyperparameter tuning domain} \end{aligned}$$

A review of all the regression methods to be used are provided in the list below, and is associated with the highlighted block of the architecture previously shown as Fig. 1, shown again below as Fig. 3.

- Support vector regression (SVR, the technique adopted in [1] introduced by [4])
- k-nearest neighbor regression (k-NN)

- ℓ_2 regularized linear regression (LR)
- Bagged neural nets (BNN)
- Extreme Learning Machines (ELM)
- RANdom SAmple Consensus (RANSAC)

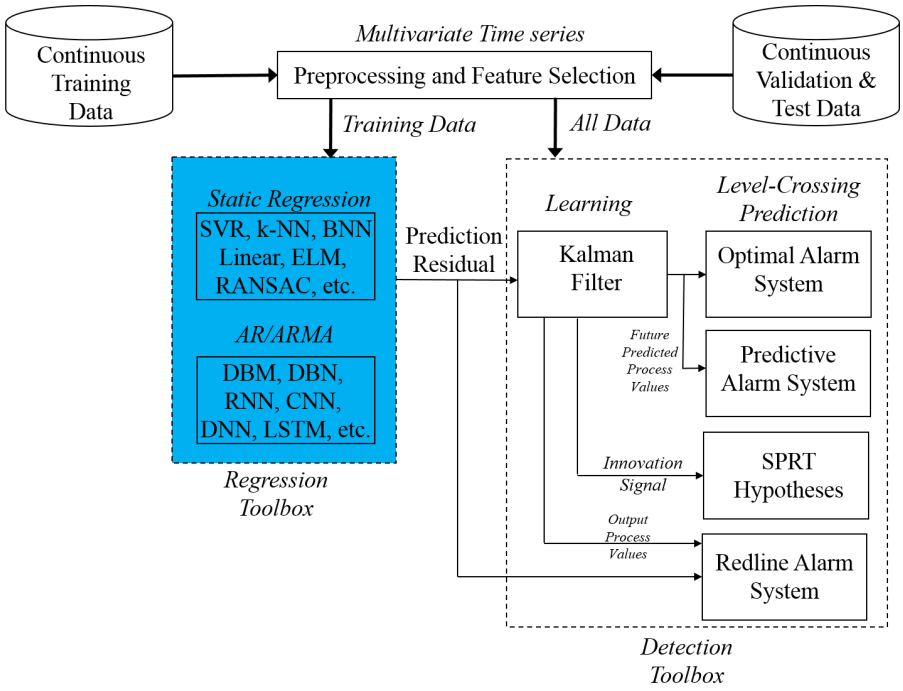


Figure 3: Functional Architecture, ACCEPT (Adverse Condition and Critical Event Prediction Toolbox), Regression Block Highlighted

4.1 Support Vector Regression (SVR)

In this subsection, we provide a brief description of the $\eta-$support vector regression algorithm that is used for target prediction as documented in [4]. Recall that MSET contains a proprietary set of kernel-based nonlinear operators designed to yield residuals with white noise spectral characteristics. However, support vector regression offers an improved ability to reduce target value prediction error compared to MSET. It also offers better built-in controls for complexity and numerical stability. This was found in a nuclear application studied in [5]. Given a finite set of multivariate observations, it is possible to reconstruct an input and target set that takes the form as shown in Eqn. 2, where $\mathbf{U} \stackrel{\triangle}{=} [\mathbf{u}_0 \dots \mathbf{u}_T]^\top$ is an input data matrix of size $(T \times p)$ and the corresponding output is denoted by $\mathbf{z} \stackrel{\triangle}{=} [z_0 \dots z_T]^\top$, termed as the target vector. Thus, there are

p parameters and T observations. Once the $\eta-$support vector regression algorithm is appropriately trained, it is possible to estimate a target function $f(\mathbf{u}_k)$ that has at the most a deviation of η from the actual observed targets $\{z_k\}_{k=0}^{T}$ for all the input data $\{\mathbf{u}_k \in \mathbb{R}^p\}_{k=0}^{T}$.

$$\mathbf{z} = f(\mathbf{U}) \tag{2}$$

The target function $f(\cdot)$ is basically a linear combination of weighted similarities between some chosen training points and the test points with an additional offset which is often known as bias, ρ. The chosen training instances with m non-zero weights are called *support vectors* (SVs, $\mathbf{u}_i$) and they are the representatives of the model. This implies that, given the model, any training points apart from the SVs are of no importance and can be thrown out without changing the performance of the algorithm. The target function is shown in Eqn. 3.

$$f(\mathbf{u}_k) = \sum_{i=1}^{m} (\alpha_i - \hat{\alpha}_i) \langle \mathbf{u}_i, \mathbf{u}_k \rangle + \rho \tag{3}$$

The support vectors and their corresponding weights, α_i and $\hat{\alpha}_i$ result from the solution of a quadratic programming optimization problem in dual form. The expression of the primal problem is shown in Eqn. 4. Further details on the cost function and optimization problem can be found in [4].

$$\begin{aligned} \text{minimize} \quad & P\left(\mathbf{q}, C, \xi_k^+, \xi_k^-\right) = \frac{1}{2}\mathbf{q}\mathbf{q}^\top + C\sum_{k=0}^{T}\left(\xi_k^+ + \xi_k^-\right) \\ \text{subject to} \quad & \left(z_k - \mathbf{q}^\top \phi(\mathbf{u}_k) - \rho\right) \leq \eta + \xi_k^+ \\ & \left(z_k - \mathbf{q}^\top \phi(\mathbf{u}_k) - \rho\right) \geq \eta + \xi_k^- \\ & \xi_k^+, \xi_k^- \geq 0 \\ & C > 0 \end{aligned} \tag{4}$$

C and η are user specified regularization and precision parameters respectively. They are chosen according to the practical guidelines set forth in [6]. ξ^+, ξ^- are non-zero slack variables, $\mathbf{q}$ is the weight vector normal to the separating hyperplane, ρ is the offset parameter, $\phi(\mathbf{u}_k)$ represents the transformed image of $\mathbf{u}_k \in \mathbb{R}^p$ in the same Euclidean space, and $k \in [0, \ldots, T]$. Throughout this research we have used the Radial Basis Function (RBF) as the mapping function given in Eqn. 5, where σ represents the hyperparameter of the Gaussian function.

$$\langle \mathbf{u}_i, \mathbf{u}_k \rangle = \exp\left(-\frac{1}{2}\frac{\parallel \mathbf{u}_k - \mathbf{u}_i \parallel^2}{\sigma^2}\right) \tag{5}$$

The parameter σ, also known as the "kernel width" parameter, controls the overall scale in horizontal variations and acts as the hyperparameter associated with the optimization problem posed in Eqn. 1. The implementation of SVR in ACCEPT is based on the libsvm package [7] and the default settings for gaussian kernel parameter is used. The cost parameter C is considered the only hyper-parameter for SVR, and is optimized based on cross-validation.

4.2 k-nearest neighbor regression (k-NN)

As described in [8], k-NN, or k-nearest neighbor regression is most useful for mapping data that is often represented in high-dimensional spaces to lower dimensional manifolds. As such, it is often used for contrast to linear dimensionality reduction techniques such as PCA (Principle Components Analysis). k-NN regression assumes that points in the data space that are located in close proximity to each other have similar output values. As such, for novel data presented to the regression algorithm, the output values must be located in close proximity to those k nearest points having similar patterns in higher dimensional space, and so the hyperparameter used for k-NN regression is the number of nearest neighbors. More details on this regression method can be found in [9].

4.3 ℓ_2 regularized linear ridge regression (LR)

One of the regression methods in ACCEPT's regression toolbox is linear ridge regression. Tikhonov regularization is used, and the associated regularization coefficient is used as a hyperparameter to optimize the NMSE as a function of how well conditioned the solution should be. A well-posed or well-conditioned problem is one that yields a solution that meets the criteria of existence, uniqueness, and robustness (*e.g.* low sensitivity to natural variation in the data). The linear regression techniques to be evaluated are linear in the parameters to be estimated only, however basis functions for the regressors themselves to be studied will involve both affine and quadratic regressors in the same vein as was presented in [8], [10], and [11]. Let $\mathbf{u}_i$ represent a vector of regressors for observation i, then scalar elements of the vector can represent linear regressors, u_i, and/or quadratic regressors, u_i^2. Note that the use of quadratic regressors include the u_i^2 regressors as well as the product of all $\binom{N}{2}$ quadratic pairs of parameters, u_i and u_j.

4.4 Bagged neural nets (BNN)

In this study we only consider traditional neural networks with a single layer perceptron as distinct from "deep learning" techniques which exploit the use of multiple hidden layers[2]. In this work, a single layer

[2]In future releases of our regression toolbox within ACCEPT, many variants of deep learning will be considered, including *e.g.* DBMs (Deep Boltzmann Machines), DBNs

perceptron refers to a network architecture that contains a single hidden layer in addition to separate input and output layers, similar to the structure illustrated in the subsequent subsection (*cf.* Fig. 4). Fundamentally, neural networks offer the ability to capture nonlinearities in data by learning weights associated with nodes in a network that are linearly combined and ultimately transformed through a nonlinear mapping.

More details on neural networks and the "bagging" process which aims to aggregate ensembles of neural networks trained on datasets generated from the same source can be found in other work [12]. "Bagging," or bootstrap aggregation, is meant to address deficiencies in stability and accuracy of the neural net performance and also reduces variance to help prevent overfitting. However, due to the lack of complexity for this particular dataset, a single base model was used and no bagging was necessary. The hyperparameter used for neural nets (NN) regression is the number of hidden neurons in a single layer perceptron.

4.5 Extreme Learning Machines (ELM)

Extreme Learning Machines (ELMs) are emerging as an alternate learning paradigm for multi-class classification and regression problems [13, 14], and have outperformed some state of the art algorithms such as backpropagation neural nets, support vector machines, *etc.* ELMs have a basic architecture similar to that of a single layer feedforward neural network, but the input layer parameters are assigned randomly (sampled from any continuous random distribution). In such a setup, the output layer parameters are the only unknowns of the model, and can be analytically determined using a linear least squares approach (See Fig. 4), making ELM training extremely fast. Some of the attractive features of ELM include the universal approximation capability, better generalization, the convex optimization problem of ELM resulting in the smallest training error without getting trapped in local minima, and availability of a closed form solution eliminating iterative training [13].

Consider the following data set

$$\{(x_1, y_1), ..., (x_N, y_N)\} \in \big(\mathcal{X}, \mathcal{Y}\big), \tag{6}$$

where N denotes the number of training samples, $\mathcal{X}$ denotes the space of the input features and $\mathcal{Y}$ denotes labels whose nature differentiate the learning problem in hand. For instance, if $\mathcal{Y}$ takes integer values {1,2,3,..} then the problem is referred to as classification and if $\mathcal{Y}$ takes real values, it becomes a regression problem. ELMs are well suited for solving both regression and classification problems and training is faster than state of the art algorithms [14]. ELM has been applied to several

(Deep Belief Networks), DNNs (Deep Neural Networks), CNNs (Convolutional Neural Networks), RNNs (Recurrent Neural Networks - AR/ARMA-style), and LSTMs (Long Short Term Memory Network - AR/ARMA-style)

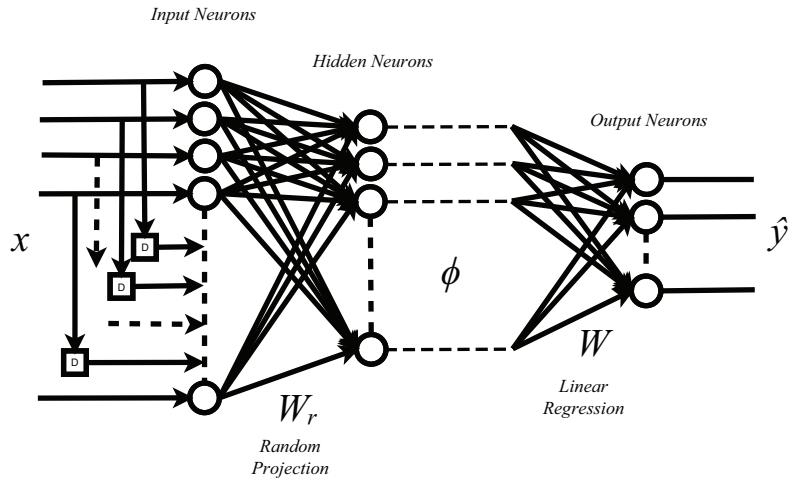


Figure 4: Extreme learning machine model structure.

benchmark problems [13, 14] , system identification [15], diagnostics [16, 17] and control problems [18] with good success. When training data is available and a model is required to be learned using all the training data, a batch training approach is adopted which involves solving the following optimization problem

$$\min_{W} \left\{ \|HW - Y\|^2 + \lambda \|W\|^2 \right\} \tag{7}$$

$$\phi = H^T = \psi(W_r^T x(k) + b_r) \in \mathbb{R}^{n_h \times 1}, \tag{8}$$

where λ represents the regularization coefficient, Y represents the vector of outputs or targets, ψ represents the hidden layer activation function (sigmoidal, sinusoidal, radial basis etc [14]) and $W_r \in \mathbb{R}^{n_i \times n_h}, W \in \mathbb{R}^{n_h \times y_d}$ represents the input and output layer parameters respectively. Here, n_i represents the dimension of inputs $x(k)$, n_h represents the number of hidden neurons of the ELM model, H represents the hidden layer output matrix and y_d represents the dimension of outputs Y. The matrix W_r consists of randomly assigned elements that maps the input vector to a high dimensional feature space while $b_r \in \mathbb{R}^{n_h}$ is a bias component assigned in a random manner similar to W_r. The number of hidden neurons determines the expressive power of the transformed feature space. The elements can be assigned based on any continuous random distribution [14] and remains fixed during the training process. Hence, training reduces to a single step calculation given by equation (9). The ELM decision rule can be expressed as in equation (10) for classification and equation (11) for regression.

$$W^* = \left(H^T H + \lambda I\right)^{-1} H^T Y \tag{9}$$

$$f(x) = sign\left(W^T[\psi(W_r^T x + b_r)]\right). \tag{10}$$

$$f(x) = W^T[\psi(W_r^T x + b_r)] \tag{11}$$

Since training involves a linear least squares solution with a convex objective function, the solution obtained by ELM is extremely fast and is a global optimum for the chosen n_h, W_r and b_r. The number of hidden neurons n_h is considered the only hyper-parameter that is optimized based on cross-validation. The other parameters of ELM such as the regression coefficient λ, random parameters W_r and b_r are fixed to default values. The user is free to modify these values to adapt to the data in ACCEPT.

4.6 RANdom SAmple Consensus (RANSAC)

Typical regression methods are based on assumptions which meet with success if they hold true. However, if these fundamental assumptions no longer hold, they can lead to misleading results as a consequence of the fragility of a particular regression method. RANSAC is a robust regression method that is not excessively affected when gross errors and outliers exist in the data [19]. These outliers may be introduced as a result of incorrect measurements or excessive noise. These errors should be rejected since they do not fit the model and can cause improper parameter estimation, and is a technique that the RANSAC regression algorithm employs. The RANSAC algorithm is iterative and comprised of two main steps which are repeated until a sufficient model is generated [19].

1. **Create a Hypothesis**

   RANSAC randomly selects N data points from a uniform distribution to instantiate a subset, S_1, from the input data set $\mathcal{U}$, defined as $\mathcal{U} = \{\mathbf{u}_1, \ldots, \mathbf{u}_N\}$. Elements of the the input vector $\mathbf{u}_k$ are defined as u_k^j, and each individual element is distributed as $u_k^j \sim \mathbb{U}[0, 1]$. Contrary to orthodox sampling methods, RANSAC uses the smallest subset of data (Minimal Sample Set (MSS)) possible to construct a model M_1 by calculating the model parameters using the subset S_1. For example, using a simple deterministic linear model representation, $u_k^j = ax_1 + bx_2 + c$, $M_1 = [a, b, c]^T$ and $N = 3$ since three data points are required to uniquely determine the parameters of a linear model [20].

2. **Evaluate the Hypothesis**

   A point is considered an inlier if it agrees with the hypothetical model within an error threshold, δ. Using more data points than the MSS for this model would decrease the probability that the chosen points are all inliers. A new subset(consensus set), S_1^*, is produced by computing all the points that agree with the current

model, M_1. RANSAC now assess the value of the model created on the complete data set using a cost function. RANSAC seeks to minimize the cost function [20] [21]:

$$C_\Lambda(\mathcal{U};\boldsymbol{\theta}) = \sum_{i=1}^{N} \rho(\mathbf{u}_i, \Lambda(\boldsymbol{\theta})) \tag{12}$$

using

$$\rho(\mathbf{u}, \Lambda(\boldsymbol{\theta})) = \begin{cases} 0 & |\tilde{z}_\Lambda(\mathbf{u}, \boldsymbol{\theta})| \leq \delta \\ \text{constant} & otherwise \end{cases} \tag{13}$$

where

$$\begin{aligned} \Lambda &= \text{the model space} \\ N &= \text{number of data points} \\ \mathcal{U} &= \text{input data set } \{\mathbf{u}_1, \ldots, \mathbf{u}_N\} \\ \boldsymbol{\theta} &= (\{\mathbf{u}_1, \ldots, \mathbf{u}_h\}), \text{estimated parameter vector} \\ h &\geq \text{the cardinality of the MSS} \\ \tilde{z}_\Lambda(\mathbf{u}, \theta) &= \text{the error associated with a data point } \mathbf{u} \end{aligned}$$

If the cost is below a predefined value, δ, *i.e.*, the current model comprises a sufficient number of inliers, then a new model M_1^* is generated using the new subset, S_1^*. If the consensus set has the lowest cost, $C_\Lambda(\mathcal{U};\boldsymbol{\theta})$, it is used as the leading candidate for this iteration of the algorithm. This process is repeated for a certain number of iterations, I. Given that φ is the probability that the algorithm will fail at least once, ω is the probability a point is an inlier and M is the number of points required to generate hypothesis [20] [19]:

$$I = \frac{\log \varphi}{\log(1 - \omega^M)} \tag{14}$$

After I iterations, the model with the lowest cost out of all leading candidates is selected as the final model. The hyperparameter used for RANSAC is the error threshold, δ.

5 Detection Toolbox

As distinct from regression, all detection methods will conform to a rigorous validation process that involve *both* nominal training data and validation data containing examples of the adverse event in question. The detection methods can be split into the three types listed below.

1. Methods that use a Monte-Carlo style implementation to empirically generate relevant alarm system statistics

2. Methods that rely on a model-based approach to generate these same performance metrics

3. A method that uses an optimal stopping rule based upon the test of hypotheses associated with abrupt changes in residuals of the modeled process output

For the detection methods that fall into one of the first two categories listed above, ROC curve analysis will be used to enable the design of tradeoffs between false alarm and missed detection probabilities. A user-specified condition of acceptance based upon missed detection and/or false alarm rates will be established, and it will be used to design an alarm system based on the training or validation ROC curve. The resulting threshold will be used to implement the detection method using the test data, illustrating all observed false positives and missed detections. These methods are based upon the use of a decision rule or hypothesis test which involves level-crossing behavior of the modeled process (see Sec. 5.3 for further detail). Strict definitions for the detection performance metrics are provided in the list below. These definitions may vary slightly from standard definitions that are provided in the context of single time points alone. The definitions provided are based upon ground truth that spans a prediction horizon with multiple time points.

False Alarm An alarm is triggered at a time point that does not contain an example of a confirmed anomalous event in at least one time point in the next d time steps, in a manner similar to the level-crossing definition provided in Sec. 5.3.

Missed Detection No alarm is triggered at a time point where an example of a confirmed anomalous event exists in at least one time point in the next d time steps, in a manner similar to the level-crossing definition provided in Sec. 5.3.

For the detection methods that involve level-crossing prediction of the modeled process, it is important to select the threshold associated with those level-crossings according to some quantifiable metric. Formulating an optimization problem emerges as a natural way to address the need to "tune" the threshold associated with the level-crossing event to the adverse events found in the validation data. Furthermore, both the prediction horizon associated with the level-crossing event and the dimension of the state-space, where applicable, are parameterized as an implicit function of the objective function associated with this optimization problem. In that way, design of the alarm system can be based upon LDS model parameters derived from training data and the level-crossings that provide the best representation of violations. Alternately,

these level-crossings be identified in advance, and act as adverse events to be derived from validation data.

A formal representation of the objective function is provided in Eqn. 15, which is based upon the existence or absence of anomalous events, and accompanying alarms associated with the level-crossing threshold L, both encoded as binary vectors for all records in the entire validation data set.

Note that the objective function in Eqn. 15 involves the AUC and is optimized over a countably finite two dimensional grid $n, d \in \mathbb{Z}^+$. The AUC is based upon using validation data as the ground truth and values of $|y_k|$ as a source of candidate thresholds to yield L. It is also important to note that the AUC optimization procedure is only the first step in the threshold selection process, and involves only finding the state dimension and prediction horizon that yield the highest AUC value. Ultimately the goal is to select a threshold which yields the most accurate representation of the ground truth. Thus, the second step is to use the respective ROC curve as the basis of this selection process. There are many different criteria which can be used in the context of ROC curve analysis, however we employ the EER (equal error rate, where $P_{fa} = P_{md}$) as the criterion for acceptance.

$$\begin{array}{cc} \text{maximize} & \text{AUC} \\ \text{subject to} & \\ n, d \in \mathbb{Z}^+ \equiv [1, 2, \ldots] & \end{array} \tag{15}$$

A summary of all the prediction methods to be discussed in this section are provided in the list below, and can be referenced in the highlighted block of Fig. 5.

- Standard exceedance
- Redline alarm system
- Predictive alarm system (Monte Carlo simulation)
- Predictive alarm system (Numerical integration)
- Optimal alarm system (Monte Carlo simulation)
- Optimal alarm system (Numerical integration)
- SPRT tests

5.1 Standard Exceedance and Predictive Alarm System

The objective of the simplest standard exceedance detection method is to obtain a threshold, L_A from alarm system design based upon a Monte-Carlo style implementation to empirically generate relevant alarm

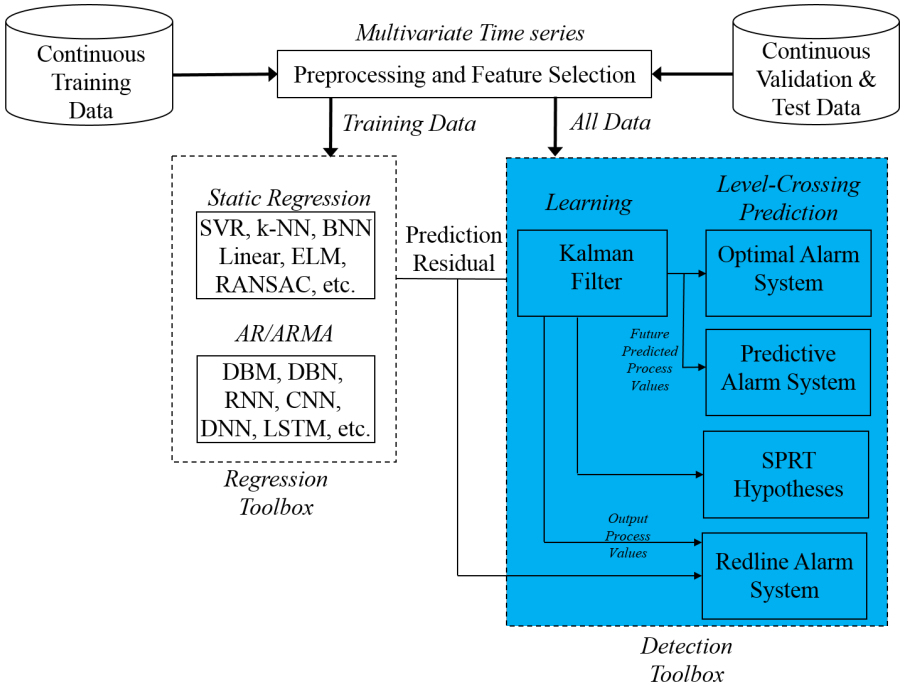


Figure 5: Functional Architecture, ACCEPT (Adverse Condition and Critical Event Prediction Toolbox), Detection Block Highlighted

system statistics (first from the preceding categorical list) to use for final testing. The standard exceedance method is also the only detection method not using LDS parameters, but shares a similar alarm design threshold, L_A, with the predictive alarm system that does require them. As such, the predictive alarm system requires the optimization problem posed in Eqn. 15. Both the standard exceedance and predictive detection methods also share a common alarm system design philosophy. Their design is based upon a Monte-Carlo simulation to empirically generate relevant alarm system statistics from realizations that use real validation in the attempt to define an envelope, $[-L_A, L_A]$, outside of which an alarm will be triggered to forewarn of the impending level-crossing event. Formally, the predictive alarm system is given by $\{|\hat{y}_{k+d|k}| > L_A\}$, such that it is based upon a predicted future process value, and the standard exceedance method is given by $|y_k| > L_A$.

A "redline" alarm system can also be designed using the same alarm rule: $|y_k| > L_A$. However, the distinction between the redline alarm system and the standard exceedance method is that the former relies on sufficient statistics used to estimate the LDS parameters that come strictly from training data. In this case, an alarm design threshold, L_A, is used as a predictor of a second more critical threshold, L. Here, L is the same threshold tuned to the observed adverse event via the optimization

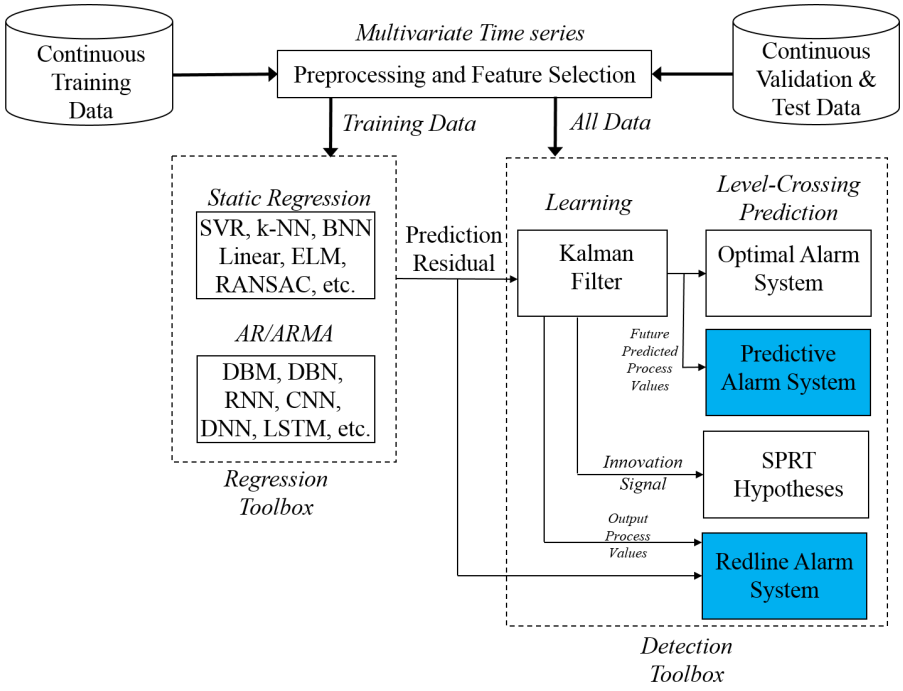


Figure 6: Functional Architecture, ACCEPT (Adverse Condition and Critical Event Prediction Toolbox), Redline and Predictive Block Highlighted

problem shown in Eqn. 15[3]. Similarly, the predictive alarm system can also be based on sufficient statistics used to estimate the LDS parameters that come strictly from training data. More detail on both these redline and predictive alarm system variants is provided in [22].

5.2 Linear Dynamical System

With one exception all detection methods in the detection toolbox require the use of the linear dynamical system. Linear dynamical systems evolve according to Eqns. 16 - 18, demonstrating propagation of the state, $\mathbf{x}_k \in \mathbb{R}^n$ which is corrupted by process noise $\mathbf{w}_k \in \mathbb{R}^n$. For convenience of presentation, it will be assumed that propagation of all state covariance matrices can reasonably well be approximated by their steady-state counterparts. This approximation, while it introduces error with regards to the probability of a level-crossing event at a specific point in time, is ostensibly negligible and will provide for a great computational advantage in the design of an alarm system. Instead of designing an alarm system for each time step, a single alarm system can be designed for all time steps. The approximation is based upon the limiting statis-

[3]Note that the standard exceedance method uses L_A for tuning rather than L. For the redline alarm system, L_A is used solely for alarm system design.

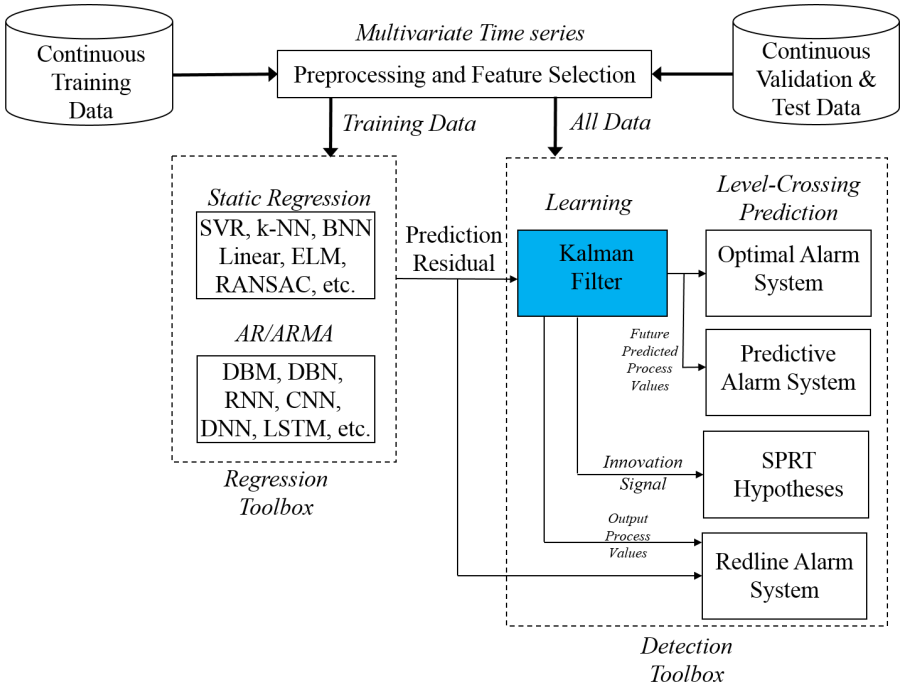


Figure 7: Functional Architecture, ACCEPT (Adverse Condition and Critical Event Prediction Toolbox), Kalman Filter Block Highlighted

tics that are reached at steady-state, which greatly reduces the computational burden, as previously identified [22]. As such, the solution of the steady-state Lyapunov function, $\mathbf{P_{xx}}$, suffices for evolution of the unconditional state covariance matrix. The output, $y_k \in \mathbb{R}$ is univariate, and is corrupted by measurement noise $v_k \in \mathbb{R}$.

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{w}_k && (16) \\ y_k &= \mathbf{C}\mathbf{x}_k + v_k && (17) \\ \mathbf{P_{xx}} &= \mathbf{A}\mathbf{P_{xx}}\mathbf{A}^\top + \mathbf{Q} && (18) \end{aligned}$$

where

$$\begin{aligned} \mathbf{w}_k &\sim \mathcal{N}(0, \mathbf{Q}), \quad \mathbf{Q} \succeq \mathbf{0} \\ v_k &\sim \mathcal{N}(0, R), \quad R > 0 \end{aligned}$$

Implementation of the optimal alarm system hinges on use of standard Kalman filter and predictor equations which are omitted for the sake of brevity. However it is important to introduce relevant predicted future process output values, covariances and cross-covariances, given below as Eqns. 19- 21, respectively. These equations rely on Kalman filter formalisms, and will be used in subsequent formulae. $\mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}}$ is the solution to the discrete algebraic Riccati equation (Eqn. 22), and $\bar{\mathbf{P}}_{\hat{\mathbf{x}}\hat{\mathbf{x}}}$ is

the steady-state *a posteriori* covariance matrix given in Eqn. 23, which both rely on the Kalman gain defined in Eqn. 24. The approximations shown in Eqns. 20 and 21 will provide for a great computational advantage in design of the optimal alarm system and its corresponding approximations for reasons stated previously. The assumption of stationarity is also required for the design of an optimal alarm system using this modeling paradigm (*cf.* Theorem from [22]), and holds here as well.

$$\begin{aligned}
\hat{y}_{k+j|k} &= \mathbf{C}\mathbf{A}^{j}\hat{\mathbf{x}}_{k+j|k} && (19)\\
\mathbf{P}_{k+j|k} &\approx \mathbf{A}^{j}(\bar{\mathbf{P}}_{\hat{\mathbf{x}}\hat{\mathbf{x}}} - \mathbf{P}_{\mathbf{xx}})(\mathbf{A}^{\top})^{j} + \mathbf{P}_{\mathbf{xx}} && (20)\\
\mathbf{P}_{k+i,k+j|k} &\approx \mathbf{A}^{j}(\bar{\mathbf{P}}_{\hat{\mathbf{x}}\hat{\mathbf{x}}} - \mathbf{P}_{\mathbf{xx}})(\mathbf{A}^{\top})^{i} + \mathbf{A}^{j-i}\mathbf{P}_{\mathbf{xx}} && (21)\\
\mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}} &= \mathbf{A}\left(\mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}} - \mathbf{F}_{\mathbf{xx}}\mathbf{C}\mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}}\right)\mathbf{A}^{\top} + \mathbf{Q} && (22)\\
\bar{\mathbf{P}}_{\hat{\mathbf{x}}\hat{\mathbf{x}}} &\triangleq \mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}} - \mathbf{F}_{\mathbf{xx}}\mathbf{C}\mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}} && (23)\\
\mathbf{F}_{\mathbf{xx}} &\triangleq \mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}}\mathbf{C}^{\top}(\mathbf{C}\mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}}\mathbf{C}^{\top} + R)^{-1} && (24)
\end{aligned}$$

The parameters to be learned are specified in Eqn. 25, as the parameter θ.

$$\theta = (\mu_{\mathbf{x}}, \mathbf{P}_0, \mathbf{A}, \mathbf{C}, \mathbf{Q}, R) \quad (25)$$

where

$$\mathbf{P}_k = E[(\mathbf{x}_k - \mu_{\mathbf{x}})(\mathbf{x}_k - \mu_{\mathbf{x}})^{\top}]$$

There are three important considerations in learning θ. The first consideration relates to the method used for learning these parameters. One such data-driven approach incorporates the use of the Expectation-Maximization (EM) algorithm, which is an iterative maximum likelihood estimation-based approach that is ultimately an alternating nonlinear optimization problem. As such, it is possible to arrive at a solution which is only a local optimum, and there may be better solutions based upon the location of the initial parameters.

In previous work [23], the EM algorithm was used to learn the model parameters using a variety of initialization techniques. Furthermore, the fidelity of resulting models was assessed via a derivative of the Akaike Information Criteria (AIC), such as one presented by Bengtsson and Cavanaugh [24], in addition to being used for model order selection.[4] In this paper we consider alternative approaches to initialization of the EM algorithm, based in part by a method known as Numerical Algorithms for Subspace State-Space System Identification (N4SID), documented in Overschee and De Moor [26–28] and Favoreel *et al.* [29]. However, using this approach for initialization does not guarantee a globally optimal

[4]Note that here, the model order selection problem is handled entirely by the detection algorithm optimization problem cited in [25].

solution. It is often suggested to use "random restarts" as a strategy for overcoming this limitation to find a near global optimum. Furthermore, any of the methods listed as candidates for global optimization in [25] would work just as well to overcome this limitation. These will be studied in earnest in future papers. N4SID will be the approach used for initialization here, as it is often advocated to avoid undesirable local minima [30], even though there is no guarantee of achieving a globally optimal solution.

The second consideration relates to the enforcement of constraints for certain parameters in θ. For any useful linear dynamical system model, an important constraint on system stability is required, which is not guaranteed by any of the methods described in this section. Thus, the M-step of the EM algorithm must include a constraint such that the eigenvalues of $\mathbf{A}$ fall within the open unit disk, which can also be represented as a constraint on the spectral radius, such that $\rho(\mathbf{A}) < 1$. Recent work by Boots [31] will allow for enforcement of stability throughout the entire identification and learning procedure in the context of the EM algorithm, as originally suggested by Siddiqi *et al.* [32]. The constraint is enforced by the use of `cvx` [33], [34], a package for specifying and solving convex programs, in the context of solving a quadratic optimization problem as posed in [31]. In some cases the data used for initialization of the EM algorithm may also yield an unstable linear dynamical system model such that $\rho(\mathbf{A}) < 1$. As such, using N4SID for initialization will fail, and as an alternative the method by Siddiqi *et al.* [32] will be implemented.

In a practical application on learning linear dynamical system parameters by Derek *et al.* [35] it was found that numerical instabilities cause convergence problems when obtaining M-step estimates for all covariance matrices $\theta_{cv} = (\mathbf{P}_0, \mathbf{Q}, R)$. This can in part be assisted by enforcing symmetry with the approximation $\mathbf{X} \approx \frac{\mathbf{X}+\mathbf{X}^\top}{2}$. However, in addition to enforcement of symmetry and stability constraints, numerical issues can also be addressed by enforcing constraints on the positive definiteness of these matrices (θ_{cv}), which is also a requirement for important control theoretic properties to hold true. This is easily achieved by using the following generic convex optimization formulation in Eqn. 26 below, also using `cvx`. The constraint is enforced only at the final step of the EM algorithm, following the recommendation of [31], which cited no observed additional increase in model fidelity or accuracy by performing this often computationally intensive computation at each iteration of the EM algorithm.

$$\begin{aligned} &\text{minimize} \quad \operatorname{tr}(\mathbf{SX}) - \log\det(\mathbf{X}) \\ &\mathbf{X} \succ \mathbf{O} \end{aligned} \tag{26}$$

where for $\mathbf{P}_0$

$$\begin{aligned}\mathbf{X} &= \mathbf{P}_0^{-1} \\ \mathbf{S} &= \mathbf{P}_{0|T}\end{aligned}$$

and for R

$$\begin{aligned}\mathbf{X} &= R^{-1} \\ \mathbf{S} &= \mathbf{S} = \frac{\Gamma_{\mathbf{y}} - \Gamma_{\mathbf{xy}}\Gamma_{\mathbf{xx}}^{-1}\Gamma_{\mathbf{xy}}^{\top}}{T+1}\end{aligned}$$

and finally for $\mathbf{Q}^{-1}$

$$\begin{aligned}\mathbf{X} &= \mathbf{Q}^{-1} \\ \mathbf{S} &= \frac{\Gamma_{\mathbf{xx}} - \left[\left(\mathbf{P}_{0|T} + \hat{\mathbf{x}}_{k|T}\hat{\mathbf{x}}_{k|T}^{\top}\right) + \Gamma_{\mathbf{xx'}}\hat{\Gamma}_{\mathbf{xx}}\Gamma_{\mathbf{xx'}}^{\top}\right]}{T}\end{aligned}$$

where

$$\begin{aligned}\Gamma_{\mathbf{y}} &\triangleq \sum_{k=0}^{T}\mathbf{y}_k\mathbf{y}_k^{\top} \\ \Gamma_{\mathbf{xy}} &\triangleq \sum_{k=0}^{T}\mathbf{y}_k\hat{\mathbf{x}}_{k|T}^{\top} \\ \Gamma_{\mathbf{xx}} &\triangleq \sum_{k=0}^{T}\mathbf{P}_{k|T} + \hat{\mathbf{x}}_{k|T}\hat{\mathbf{x}}_{k|T}^{\top}\end{aligned}$$

and

$$\begin{aligned}\Gamma_{\mathbf{xx'}} &\triangleq \sum_{k=0}^{T-1}\mathbf{P}_{k+1,k|T} + \hat{\mathbf{x}}_{k+1|T}\hat{\mathbf{x}}_{k|T}^{\top} \\ \hat{\Gamma}_{\mathbf{xx}} &\triangleq \Gamma_{\mathbf{xx}}^{-1} + \Gamma_{\mathbf{xx}}^{-1}(\mathbf{P}_{T|T} + \hat{\mathbf{x}}_{T|T}\hat{\mathbf{x}}_{T|T}^{\top})\Gamma_{\mathbf{xx}}^{-1}\end{aligned}$$

where

$$\begin{aligned}\hat{\mathbf{x}}_{k|T} &\triangleq E[\mathbf{x}_k|y_0,\ldots,y_T] \\ \mathbf{P}_{k|T} &= E[(\mathbf{x}_k - \mu_{\mathbf{x}})(\mathbf{x}_k - \mu_{\mathbf{x}})^{\top}|y_0,\ldots,y_T]\end{aligned}$$

The third consideration relates to scalability and computational complexity of the E-step within the EM algorithm. Machine learning techniques used for adverse event prediction should be highly scalable and capable of supporting a fleetwide analysis for future deployment. This is important when considering the vast quantities of fleet data that can be used to train a linear dynamical system model. However, it is equally important when training a model based upon a smaller representative

sample from the fleet. Both sample sizes are used in the context of the optimization problem for detection algorithms that rely on this mathematical construct.

The optimization requires the pre-computation of LDS parameters with as many different model orders, $n, d \in \mathbb{Z}^+ \equiv [1, 2, \ldots]$ as are practical to learn. Practically, constraints are often applied such that $n \leq 10$. The optimization problem (*cf* [25]) can then be performed without having to learn LDS parameters redundantly, since many iterations are often required to arrive at a near global optimum as a function of model order in addition to other relevant tuning parameters, as previously suggested.[5] However, finding LDS parameters for model orders ranging from $n \in [2, \ldots, 10]$ is still a computationally burdensome task, and as such complexity is important here as well.

Martens [36] developed a novel algorithm to specifically address this issue, by making approximations based upon rigorous statistical and control theoretic principles. These approximations are enabled in part by a set of recursions established for the aggregate sufficient statistics that are used as part of the M-step in the EM algorithm. Per iteration computational complexity is consequently reduced from $\mathcal{O}(n^3 T)$ to $\mathcal{O}(n^3 k_{lim})$, where $k_{lim} \ll T$ and k_{lim} is a user selected value. This provides an extremely valuable tool in surmounting an otherwise intractable computational bottleneck for very large datasets, and where applicable the approach can be used for this study.

5.3 Optimal Level-Crossing Prediction

As with the redline and predictive methods, the optimal alarm system also requires parametric tuning of n, L, and d using Eqn. 15 to find the best final outcome. The ultimate goal of these optimizations is to close the gap between the level-crossing prediction hypothesis being tested and prediction of the actual adverse event. Details on alarm system design have been given in [22], and are based upon appropriate threshold selection of P_b, given pre-established user-specified conditions of acceptance based upon missed detection and/or false alarm rates, and the resulting ROC curve. Just as with the predictive alarm system, the optimal alarm system can either use a Monte-Carlo style implementation to empirically generate relevant alarm system statistics or use the estimated LDS parameters that come strictly from training data as a function of sufficient statistics to compute those alarm system statistics numerically without relying on Monte Carlo simulations.

Recall that the adverse event detection problem can be cast as an optimal level-crossing prediction problem. In this section, we provide an overview of some of the fundamental theoretical and implementation details of this method. A level-crossing event, C_k, is defined with a critical

[5] See footnote 4.

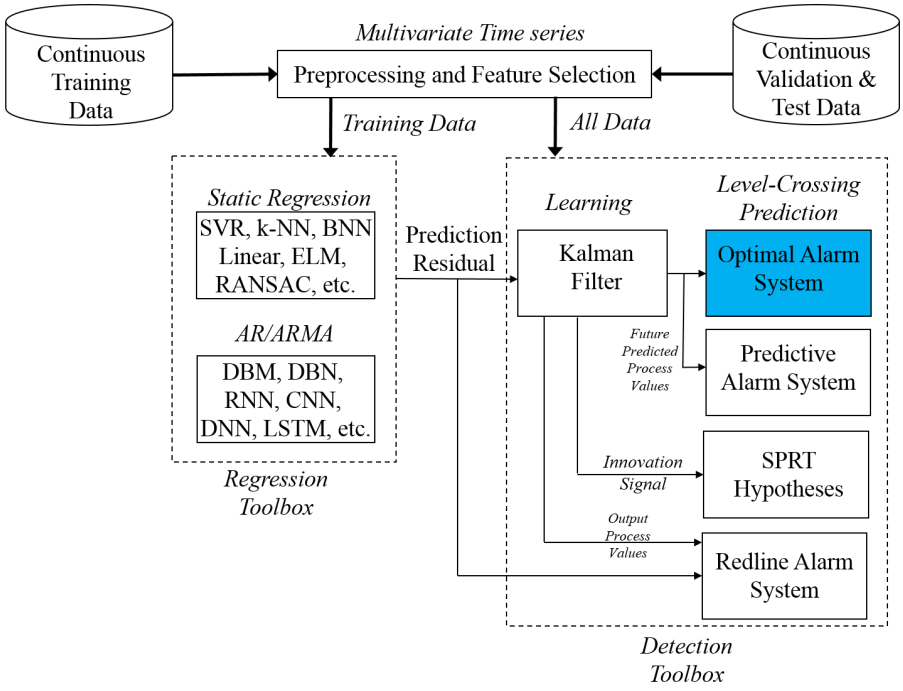


Figure 8: Functional Architecture, ACCEPT (Adverse Condition and Critical Event Prediction Toolbox), Optimal Alarm System Block Highlighted

level, L, that is assumed to have a fixed, static value. The level defines an envelope outside of which a critical parameter, y_k may experience several excursions. This parameter can be represented by a dynamic process, and is modeled as a zero-mean stationary linear dynamic system driven by Gaussian noise. The theoretical underpinnings of this approach are based upon this standard representation of the optimal level-crossing problem.

The essence of the optimal alarm system is derived from the use of the likelihood ratio resulting in the conditional inequality: $P(C_k|y_0, \ldots, y_k) \geq P_b$. This basically says "give alarm when the conditional probability of the event, C_k, exceeds the level P_b." Here, P_b represents some optimally chosen border or threshold probability with respect to a relevant alarm system metric. P_b is an extremely important parameter, as it effectively defines the space spanned by the alarm region and therefore controls the tradeoff between false alarms and missed detections. It is thus chosen with respect to these metrics and is the key parameter used for design of an optimal alarm system.

It is necessary to find the alarm regions in order to design the alarm system. The event, C_k, can be chosen arbitrarily, and is usually defined with respect to a prediction window, d, as well as the critical threshold, L. In this paper, the event of interest is shown in Eqn. 27, and repre-

sents at least one exceedance outside of the threshold envelope specified by $[-L, L]$ of the process y_k within the specified look-ahead prediction window, d. Mathematically, it can also be represented as the union of disjoint subevents, $\bigcup_{j=1}^{d} S_{k+j}$, or as the union of overlapping subevents, $\bigcup_{j=1}^{d} E'_{k+j}$.

E_{k+j} represents an exceedance at the j^{th} time step, and S_{k+j} is a sequence of such subevents, of which only the j^{th} represents an exceedance.

$$C_k \triangleq \bigcup_{j=1}^{d} S_{k+j} \tag{27}$$

$$= \bigcup_{j=1}^{d} E'_{k+j} \tag{28}$$

$$= \mathcal{I} \setminus \bigcap_{j=1}^{d} E_{k+j} \tag{29}$$

where

$$E_{k+j} \triangleq \{|y_{k+j}| < L\},\ \forall j \geq 1$$

$$S_{k+j} \triangleq \begin{cases} E'_{k+j} & j = 1 \\ \bigcap_{i=1}^{j-1} E_{k+i}, E'_{k+j} & \forall j > 1 \end{cases}$$

Previous work [22] provides the mathematical underpinnings for the optimal alarm condition corresponding to the level-crossing event, shown here as Eqn. 30. Alternatively, the optimal alarm condition derived in [22] can be expressed in terms of the subevents E_{k+j}, as shown in Eqn. 31.

$$P(C_k|y_0, \ldots, y_k) \geq P_b \tag{30}$$

$$P(\bigcap_{j=1}^{d} E_{k+j}|y_0, \ldots, y_k) \leq 1 - P_b \tag{31}$$

As was discussed in [22], it is not possible to obtain a closed-form representation of the parametrization for the optimal alarm region resulting from Eqn. 30. Monte Carlo simulation is thus often used to generate alarm system design statistics to be estimated empirically. These estimates are generated by using validation data consisting of example adverse events and the conditional probability expressed in Eqn. 30. However, alarm system design statistics can also be obtained by computing relevant multivariate normal probabilities which approximate the optimal alarm region, evaluated by numerically integrating expressions

as provided in [22]. Although there were two such approximations studied in [22], using the one requiring the least computational effort was shown not to lose any appreciable accuracy as compared to the other approximation. We refer the interested reader to [1], [22], or [37] for further detail.

5.4 SPRT Hypothesis Tests

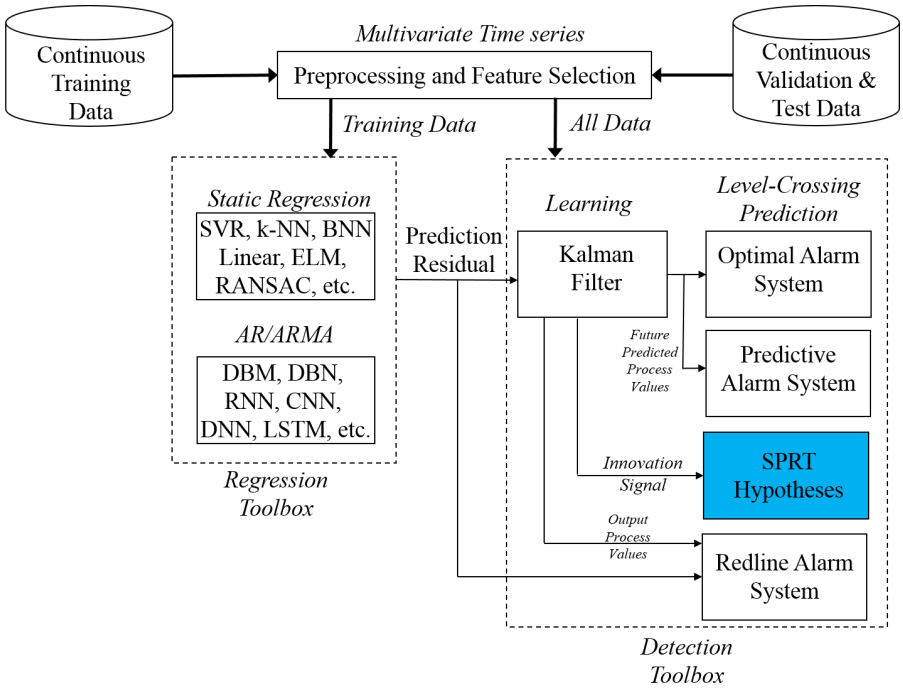


Figure 9: Functional Architecture, ACCEPT (Adverse Condition and Critical Event Prediction Toolbox), SPRT Block Highlighted

As mentioned in Sec. 1, SPRT tests are a central part of MSET, and have met with quite favorable results. Here we provide more detail on SPRT, but full details on SPRT can be found in [38]. The SPRT test statistic is a cumulative log-likelihood ratio between the probability distributions characterizing anomalous and nominal behavior. The SPRT test represents the last item in the preceding categorical list, representing the method using an optimal stopping rule based upon the test of hypotheses associated with abrupt changes in residuals of the modeled process output.

There are four distinct alternative hypotheses shown as Eqns. 34-37, which are tested using the SPRT statistics that are computed with MSET to provide comprehensive coverage. These statistics test the hypotheses for both positive and negative mean drifts, as well as nominal and inverse variance shifts. The denominator represents the null hypoth-

esis, $\mathcal{H}_0$ shown as Eqn. 33 and characterizes nominal behavior, while the numerator represents the alternative hypothesis, $\mathcal{H}_j$ and characterizes anomalous behavior.

$$
\begin{aligned}
S_k^j &= S_{k-1}^j + \sum_{i=1}^{k} \log \frac{p(\varepsilon_i|\mathcal{H}_j)}{p(\varepsilon_i|\mathcal{H}_0)}, \quad j \in \{1,\ldots,4\} && (32)\\
p(\varepsilon_i|\mathcal{H}_0) &= \mathcal{N}(\varepsilon_i; 0, \mathbf{C}\mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}}\mathbf{C}^\top + R) && (33)\\
p(\varepsilon_i|\mathcal{H}_1) &= \mathcal{N}(\varepsilon_i; M_{pos}, \mathbf{C}\mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}}\mathbf{C}^\top + R) && (34)\\
p(\varepsilon_i|\mathcal{H}_2) &= \mathcal{N}(\varepsilon_i; -M_{neg}, \mathbf{C}\mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}}\mathbf{C}^\top + R) && (35)\\
p(\varepsilon_i|\mathcal{H}_3) &= \mathcal{N}(\varepsilon_i; 0, V_{nom}(\mathbf{C}\mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}}\mathbf{C}^\top + R)) && (36)\\
p(\varepsilon_i|\mathcal{H}_4) &= \mathcal{N}\left(\varepsilon_i; 0, \frac{\mathbf{C}\mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}}\mathbf{C}^\top + R}{V_{inv}}\right) && (37)\\
\mathbf{P}_{\hat{\mathbf{x}}\hat{\mathbf{x}}} &\in \mathbf{R}^{n\times n}(\text{solution to DARE}) && (38)
\end{aligned}
$$

A theoretically optimal stopping rule, $S_k^j > \log \frac{1-P_{md}}{P_{fa}}$ provides the best threshold, given user-specified tolerances as established targets for both missed detection and false alarm probabilities (as P_{md}, P_{fa} respectively). This class of prediction methods tests shifts in the first and second moments of the distribution formed by the innovation signal. Positive or negative shifts in the first moment or mean are given by parameter M_{pos} and M_{neg}, respectively, and increases or decreases in the second moment or variance are given by parameters V_{nom} and V_{inv}, respectively.

As with the other detection methods, alarm systems based upon testing the SPRT hypotheses will require parametric tuning given by the modified objective function in Eqn. 39.

$$
\begin{array}{c}
\text{minimize} \\
\\
\text{subject to} \\
\mathbf{s} \in \mathbb{R}^4 \\
n \in \mathbb{Z}^+ \equiv [1, 2, \ldots]
\end{array}
\qquad
\frac{\sum_{i=1}^{f}\sum_{k=1}^{T_i} \mathcal{G}(i,k) \bigwedge \bigcup_{j=1}^{4} \left\{ S_k^j > \log \frac{1-P_{md}}{P_{fa}} \right\}}{\sum_{i=1}^{f} T_i} \qquad (39)
$$

where

$$
\begin{aligned}
\mathcal{G}_(i,k) &= \text{Ground truth classification for the } k^{th} \\
&\quad \text{time point in the } i^{th} \text{ validation record} \\
\mathbf{s} &\triangleq \begin{bmatrix} M_{pos} \\ M_{neg} \\ V_{nom} \\ V_{inv} \end{bmatrix}
\end{aligned}
$$

Note here that the optimization function is now based upon a modified Hamming distance metric and expressed as a function of n, M_{pos},

$M_{neg}, V_{nom}, V_{inv}$ as distinct from AUC expressed as a function of n and d in Eqn. 15. Any number of optimization techniques can be used to solve this problem, however in an attempt to find a near global optimum, a number of different approaches have been tested, which are provided in the list below.

- Simulated Annealing
- Genetic Algorithm
- Multi Start - choosing multiple local initial starting points uniformly, followed by running local optimizations from each of the points
- Global Search - use a serial scatter-search method to find initial starting points from which to run local optimizations, followed by elimination of the unlikely candidates
- Pattern Search - gradient-free pattern search via direct objective function evaluation

In order to allow for a fair comparison of all prediction methods, we will implement a prewhitening filter for the SPRT-based hypotheses. This is required in order to meet the assumptions implicit in using the SPRT, which is that the residuals are white (*i.e.* not serially correlated), and Gaussian. The Gaussian nature of the residuals has already been accounted for in part, based upon the discussion of the relationship between the KS statistic and optimization of the NMSE metric that takes place by applying the f-fold cross validation procedure discussed in Sec. 1. Since we are not using the same state estimation procedure as MSET, which are specifically designed to yield residuals that are white as discussed in [2], this prewhitening step is compulsory. Conveniently, prewhitening occurs naturally via the Kalman filter residual. Thus, the associated filter parameters are based upon the same parameters subsequently used for testing the level-crossing prediction hypothesis. This lends itself nicely to a well balanced basis for comparison.

Appropriately tuned parameter values can be selected by solving the constrained optimization problems posed in either Eqn. 15 or Eqn. 39 for all relevant detection methods, given the appropriate set of optimization arguments. After this step, the corresponding optimized parameter values[6] can be used for final testing and implementation. This step is seeded by a disjoint hold out dataset which was not used for either training or validation, and presumably contains real examples of adverse events that are similar in nature to those used for validation, or more formally, derived from the same distribution and data-generating process. The same testing regimen is applied for all selected regression and

[6] and where applicable, alarm design thresholds

prediction methods for comparison. From this we can ascertain the best combination of regression and detection algorithms to use as dictated by the given performance metrics.

Table 3: Tunable Regression Hyper-parameters

Regression Method	**Regression Description**	**Hyper-parameter**	**Hyper-parameter description**
SVR	Support Vector Regression	σ or C	Kernel Width or cost parameter
k-NN	k-NN Regression	k	Number of nearest neighbors
LR1	Ridge (linear) regression using linear regressors	λ	ℓ_2 regularization coefficient
LR2	Ridge (linear) regression using quadratic regressors	λ	ℓ_2 regularization coefficient
ELM	Extreme Learning Machines	n_h	number of hidden neurons
BNN	Bagged Neural Networks	n_h	number of hidden neurons
RANSAC	RANdom SAmple Consensus	δ	cost threshold

6 Conclusions

We have introduced a new architectural framework that can be run in MATLAB® called ACCEPT (Adverse Condition and Critical Event Prediction Toolbox). An open source release of this package will be made publicly available before the summer of 2015[7]. The fundamental purpose of releasing this software package is to provide a tool which can be used specifically for the prediction or forecasting of adverse events in time series data. It offers a single, unifying framework in which to compare a variety of combinations of algorithmic approaches, and provides a platform to act as a catalyst in advancing the state of the art in technologies related to this problem.

[7]Source code will be posted at the *DASHlink* URL: `https://c3.nasa.gov/dashlink/projects/10/resources/`.

In future work we will also extend the scope of the software to incorporate alternate classes of regression methods (*e.g.* streaming, AR/ARMA-based, *etc.*) in the context of the same architectural framework. As such, the framework has been designed with flexibility and modular extensibility to accommodate future innovations. Appendix D provides a developer's guide which will provide a way forward towards that end. We will also augment the architectural framework to more seamlessly test a variety of explanatory feature sets, which will aid in event localization and isolation efforts.

References

1. Martin, R.; and Das, S.: Near Real-Time Optimal Prediction of Adverse Events in Aviation Data. *Proceedings of the AIAA Infotech@Aerospace Conference*, Atlanta, Georgia, April 2010.

2. Bickford, R.: MSET Signal Validation System Final Report. Technical report, NASA Contract NAS8-98027, August 2000.

3. Hines, J. W.; and Seibert, R.: Technical Review of On-Line Monitoring Techniques for Performance Assessment Volume 1: State-of-the-Art. NUREG/CR-6895, University of Tennessee, January 2006.

4. Smola, A. J.; and Schölkopf, B.: A Tutorial on Support Vector Regression. , Statistics and Computing, 2003.

5. Zavaljevski, N.; and Gross, K. C.: Support Vector Machines for Nuclear Reactor State Estimation. , Argonne National Laboratory, 2000.

6. Cherkassky, V.; and Ma, Y.: Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, vol. 17, January 2004, pp. 113–126.

7. Chang, C.-C.; and Lin, C.-J.: LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, vol. 2, 2011, pp. 27:1–27:27.

8. Martin, R.; and Kaul, U. K.: Optimization of Perturbation Parameters for Simulated Free Shear Layer Flow. *7th AIAA Flow Control Conferences*, Atlanta, GA, USA, June 2014.

9. Krämer, O.: Dimensionality Reduction by Unsupervised K-Nearest Neighbor Regression. *IEEE 10th International Conference on Machine Learning and Applications*, Honolulu, HI, USA, December 2011.

10. Pressburger, T.; Hoelscher, B.; Martin, R. A.; and Sricharan, K.: Simple Sensitivity Analysis for Orion GNC. *Proceedings of the AIAA*

Guidance, Navigation, and Control Conference, Boston, MA, USA, August 2013.

11. Chu, E.; Gorinevsky, D.; and Boyd, S. P.: Detecting Aircraft Performance Anomalies from Cruise Flight Data. *Proceedings of the AIAA Infotech@Aerospace Conference*, Atlanta, Georgia, April 2010.

12. Hastie, T.; Tibshirani, R.; and Friedman, J.: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 2009.

13. Huang, G.-B.; Zhu, Q.-Y.; and Siew, C.-K.: Extreme Learning Machine: Theory and Applications. *Neurocomputing*, vol. 70, 2006, pp. 489–501.

14. Huang, G.-B.; Zhou, H.; Ding, X.; and Zhang, R.: Extreme Learning Machine for Regression and Multiclass Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 42, no. 2, 2012, pp. 513–529.

15. Janakiraman, V. M.; Nguyen, X.; and Assanis, D.: Nonlinear identification of a gasoline HCCI engine using neural networks coupled with principal component analysis. *Applied Soft Computing*, vol. 13, no. 5, 2013, pp. 2375 – 2389.

16. Janakiraman, V.; Nguyen, X.; Sterniak, J.; and Assanis, D.: Identification of the Dynamic Operating Envelope of HCCI Engines Using Class Imbalance Learning. *Neural Networks and Learning Systems, IEEE Transactions on*, vol. PP, no. 99, 2014, pp. 1–1.

17. Janakiraman, V.; Nguyen, X.; and Assanis, D.: Stochastic Gradient Based Extreme Learning Machines For Online Learning of Advanced Combustion Engines. *Neural Networks and Learning Systems, IEEE Transactions on (In Review)*, vol. PP, no. 99, 2014, pp. 1–1.

18. Janakiraman, V.; Nguyen, X.; and Assanis, D.: Nonlinear Model Predictive Control of Gasoline HCCI Engines Using Extreme Learning Machines. *Neural Networks and Learning Systems, IEEE Transactions on (In Review)*, vol. PP, no. 99, 2014, pp. 1–1.

19. Fischler, M. A.; and Bolles, R. C.: Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 1981, pp. 381–95.

20. Choi, Sunglok, T. K.; and Yu, W.: British Machine Vision Conference (BMVC). *Performance Evaluation of RANSAC Family*, 2009.

21. Zuliani, M.: RANSAC for Dummies. `http://vision.ece.ucsb.edu/~zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf`, 7 2014. Accessed: 2014-11-17.

22. Martin, R.: A State-Space Approach to Optimal Level-Crossing Prediction for Linear Gaussian Processes. *IEEE Transactions on Information Theory*, vol. 56, no. 10, October 2010, pp. 5083 –5096.

23. Martin, R.: An Investigation of State-Space Model Fidelity for SSME Data. *Proceedings of the International Conference on Prognostics and Health Management*, IEEE, October 2008.

24. Bengtsson, T.; and Cavanaugh, J. E.: An Improved Akaike information criterion for state-space model selection. *Computational Statistics and Data Analysis*, vol. 50, no. 10, 2006, pp. 2635–2654.

25. Martin, R.; Das, S.; Janakiraman, V.; Nielsen, D.; and Hosein, S.: An Architectural Framework for the Prediction of Adverse Events in Aviation Data using Optimal Alarm Systems. *Reliability Engineering & System Safety*, vol. XX, no. X, 2015, pp. YY – YY.

26. Overschee, P. V.; and Moor, B. D.: *Subspace Identification for Linear Systems, Theory, Implementation, Applications*. Kluwer Academic Publishers, 1996.

27. Overschee, P. V.; and Moor, B. D.: Subspace algorithms for the stochastic identification problem. *Automatica*, vol. 29, no. 3, 1993, pp. 649–660.

28. Overschee, P. V.; and Moor, B. D.: N4SID: Subspace algorithms for the identification of combined deterministic-stochastic systems. *Automatica*, vol. 30, no. 1, January 1994, pp. 75–93.

29. Favoreel, W.; Moor, B. D.; and Overschee, P. V.: Subspace state space system identification for industrial processes. *Journal of Process Control*, vol. 10, no. 2-3, 2000, pp. 149 – 155.

30. Smith, G.; and Robinson, A.: A Comparison Between the EM and Subspace Identification Algorithms for Time-Invariant Linear Synamical Systems. CUENDF-INFENG/TR.345, Cambridge University, Cambridge University Engineering Department Trumpington Street Cambridge, CB2 1PZ England, November 2000.

31. Boots, B.: Learning Stable Linear Dynamical Systems. Master's Thesis, Carnegie Mellon University, May 2009.

32. Siddiqi, S.; Boots, B.; and Gordon, G.: A Constraint Generation Approach to Learning Stable Linear Dynamical Systems. *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, and S. Roweis, eds., MIT Press, Cambridge, MA, 2008, pp. 1329–1336.

33. Grant, M.; and Boyd, S.: CVX: Matlab Software for Disciplined Convex Programming, version 1.21. `http://cvxr.com/cvx`, Apr. 2011.

34. Grant, M.; and Boyd, S.: Graph implementations for nonsmooth convex programs. *Recent Advances in Learning and Control*, V. Blondel, S. Boyd, and H. Kimura, eds., Lecture Notes in Control and Information Sciences, Springer-Verlag Limited, 2008, pp. 95–110. `http://stanford.edu/~boyd/graph_dcp.html`.

35. Derek, M.; Isaacs, K.; McElfresh, D.; Murguia, J.; Nguyen, V.; Shao, D.; Wright, C.; and Bremer, M.: Anomaly Detection with Multi-dimensional State Space Models. , San Jose State University, February 19, 2010.

36. Martens, J.: Learning the Linear Dynamical System with ASOS. *Proceedings of the 27th International Conference on Machine Learning (ICML)*, Haifa, Israel, 2010.

37. Martin, R. A.: Extreme value analysis of optimal level-crossing prediction for linear Gaussian processes. *Journal of Time Series Analysis*, 2012, pp. no–no. URL `http://dx.doi.org/10.1111/j.1467-9892.2012.00791.x`.

38. Basseville, M.; and Nikiforov, I. V.: *Detection of abrupt changes: theory and application*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

Appendix A

MATLAB Toolboxes Required for Full Functionality

- Statistics Toolbox (MATLAB)
  - Basic requirement
- Optimization Toolbox (MATLAB)
  - Basic requirement
- Parallel Computing Toolbox (MATLAB)
  - Required for running distributed jobs on cluster (if available) to reduce processing time
- MATLAB Compiler (MATLAB)
  - Required for running distributed jobs on cluster (if available) to reduce processing time, if Parallel Computing Toolbox is not available

- Global Optimization Toolbox (MATLAB)
  - Option to run global optimization routines in support of regression or SPRT detection optimization
- Control Systems Toolbox (MATLAB)
  - Required for any detection method using an LDS
- Neural Network Toolbox (MATLAB)
  - Required for testing BNN (Bagged neural networks) as one the regression methods

Appendix B

Auxiliary Toolboxes Required for Full Functionality

Please note that all of these package folders must be installed according to these directions:

- Third Party Toolboxes
  - KPMstats and KPMtools Toolboxes (Kevin Murphy, MIT/GPL: `http://code.google.com/p/bnt/source/browse/trunk/?r=36`)
    - Basic requirement, place anywhere on your Matlab path
  - SMO code for SVR routine
    - Place the files listed below under `$ACCEPT_DIR/regressopt/svropt`
    - The following files are required for running SVR as one of the regression methods
      - `gsmo.m` (and also equivalent mex-compiled version for your OS, *e.g.* `gsmo_mex.mexa64`, link: `https://searchcode.com/codesearch/view/29473653/`)
      - `kernel.m` (and also equivalent mex-compiled version for your OS, *e.g.* `kernel.mexa64`, link: `http://cmp.felk.cvut.cz/cmp/software/stprtool/`)
  - LibSVRcode for SVR routine (Chih-Chung Chang and Chih-Jen Lin, `https://github.com/cjlin1/libsvm`)
    - Install in `$ACCEPT_DIR/regressopt/libsvropt`

- RANSAC code for regression (Marco Zulianis RANSAC toolbox, released under GNU LGPL at `https://github.com/RANSAC/RANSAC-Toolbox`, and Laurens van der Maatens Matlab Toolbox for Dimensionality Reduction released under the FreeBSD License at `http://lvdmaaten.github.io/drtoolbox/`)
  - Install RANSAC toolbox in `$ACCEPT_DIR/regressopt/ransac`
  - In the Matlab Toolbox for Dimensionality Reduction, locate `/drtoolbox/techniques/pca.m`, place in `$ACCEPT_DIR/regressopt/ransac`, and rename it `RANSACpca.m`.
- Kalman Filter Toolbox (Kevin Murphy, MIT/GPL `http://code.google.com/p/bnt/source/browse/trunk/?r=36`), required for any detection method using an LDS, the following files need to be placed under `$ACCEPT_DIR/ldslearn`
  - `smooth_update.m`
  - `kalman_update.m`
  - `kalman_filter.m`
  - `kalman_smoother.m`
- ASOS Toolbox (James Martens, Apache 2.0, obtained here: `http://www.cs.toronto.edu/~jmartens/ASOS/`)
  - Option for any detection method using an LDS
  - Should be installed under `$ACCEPT_DIR/ASOS`
- CVX toolbox (Michael Grant & Stephen Boyd, GNU General Public License 2.0, obtained here: `http://cvxr.com/cvx/`)
  - Option for any detection method using an LDS
  - Should be installed under `$ACCEPT_DIR/cvx`, using cvx installation directions
  - Follow directions in `Readme.docx` for additional installation instructions

- Toolboxes developed in-house at NASA Ames Research Center
  - Regression Wrapper Code (released w/ACCEPT)
    - Basic requirement
  - SMO code for SVR routine
    - `svmregSMO.m` is released w/ACCEPT under `$ACCEPT_DIR/regressopt/svropt` (required for running SVR as one of the regression methods)
  - Kalman filter augmentation ARC 17528-1 (Rodney Martin, not released w/ACCEPT and can be accessed here: `https://c3.nasa.gov/dashlink/members/10/resources/?type=al)` (required for any detection method using an LDS, the following files need to be placed under `$ACCEPT_DIR/ldslearn`)

    - `learn_kalman.m` (use modified version released separately)
    - `em_converged.m` (use modified version released separately)
  - CAMCOS LDS Initialization script (SJSU released to NASA w/ ACCEPT)
    - Option for any detection method using an LDS
  - Stanford cvx utility calls (Eric Chu, released to NASA under NRA NNX07AE11A)
    - Option for positive definite matrix constraint enforcement
  - ROC Curve Augmentation ARC 17529-1 (Rodney Martin, not released w/ACCEPT and can be accessed here: `https://c3.nasa.gov/dashlink/members/10/resources/?type=al`)
    - Required for all detection methods except for SPRT
    - Install in `$ACCEPT_DIR/detectopt`
  - Optimal Alarm Toolbox (Rodney Martin, NOSA, released w/ ACCEPT, but can be accessed here: `https://c3.nasa.gov/dashlink/resources/119/`)
    - Required for running Optimal Alarm System as one of the detection methods
    - If not already part of package, extract under `$ACCEPT_DIR/optalarm_osrelease`
  - MCR Scheduler toolbox to set up distributed computing w/ Matlab compiler (released w/ACCEPT). Option for distributed computing only if a computing cluster is available. Set the following fields of the `sched` struct in `createJobsNew.m` accordingly
    - `sched.Type='';`
    - `sched.JobFolder='';`
    - `sched.MCRRoot='../MATLAB_Compiler_Runtime/v713/';`
    - `sched.ServerName='';`
    - `sched.SubmitArguments= '-l walltime=168:00:00 -l nodes=1:ppn=1';`
    - `sched.ResourceTemplate='-l nodes=N';`

- Uncategorized Toolboxes
  - LDS stability constraint toolbox (Siddiqi *et al.* [32], no formal license other than a requirement to acknowledge researchers by citation of corresponding paper, obtain here: `http://www.select.cs.cmu.edu/projects/stableLDS/`) (required for any detection method using an LDS, extract package into existing folder `$ACCEPT_DIR/stablelds`
    - Keep `learnCGModelEM.m`! (modified version which should already be included in `$ACCEPT_DIR/stablelds`)

    - Keep `learnCGModelSS.m`! (modified version which should already be included in `$ACCEPT_DIR/stablelds`)
  - N4SID Toolbox (Overschee & DeMoor, International, no formal license, obtained here: `http://homes.esat.kuleuven.be/~smc/sysid/software/`)
    - Required for any detection method using an LDS
    - Install in `$ACCEPT_DIR/subfun`
    - Use existing modified version of `subid.m` included in ACCEPT package

Appendix C

ACCEPT set-up and configuration guide

Dependent open sourced packages are not included with ACCEPT. Instructions for installing the relevant packages and files for all auxiliary packages can be found in Appendix B.

1. ACCEPT set-up
   (a) Environment variables to set
      i. `DATA_DIR` = local path containing data repository
      ii. `ACCEPT_DIR` = local path where you've checked out the ACCEPT package
   (b) Add entire repository to matlab path (exercise caution with the `cvx` branch: you should only add the root and `/functions` folders)
   (c) Follow directions for installation of auxiliary packages in Appendix B.
2. Customized functions (auxiliary files to edit and create). In the file `User_input_accept_generic.m` you will find the following lines - modify as follows and save as `User_input_accept.m` (without the "`_generic`" suffix):
   (a) `params.acceptpath=getenv('ACCEPT_DIR');` % No modification necessary
   (b) `params.datapath=[getenv('DATA_DIR') '/Data'];` % Enter location (path) of data corresponding to adverse events and their nominal data counterparts
      i. Partition these datafiles into `/Training`, `/Validation`, and `/Testing` sub-directory structures.
      ii. There must be as many training files as there are validation files.

iii. These datafiles should be in `*.mat` format.

iv. The `*.mat` file must contain a struct with the following fields at the very minimum: `data`, `header`.

v. The data field must represent a matrix of numerical elements, with columns corresponding to the header fields, and rows corresponding to the time index.

vi. The header field must be a cell array containing the names of all parameters.

(c) `params.anomalytype='Adverse Event 1','Adverse Event 2','Adverse Event 3';` % Contains the names of the adverse events to predict

(d) `params.loadfcn={'Function_1','Function_2', 'Function_3'};` % Contains names of the loading functions for data corresponding to a particular scenario (*e.g.* operating regime)

i. These functions partition the datasets provided at the location(s) provided in `params.datapath` according to the corresponding scenario.

ii. Example call:
```
[header,data,message] =
Function_1('nomdatafile.mat');
```

iii. The function must take in as its sole argument the filename of the `*.mat` file (*e.g.* `nomdatafile.mat`)

iv. The function must return as output arguments header, data, and message. The first two outputs are self-explanatory, and the last argument can be used to report errors.

v. The body of the function must include the logic necessary to temporally partition out the specific section of the data file that corresponds to the scenario of interest.

vi. Please note that there is no need to partition based upon the parameters, partitioning here is only time-based.

(e) `params.truthfcn={'GTFunc_1','GTFunc_2','GTFunc_3'};` % Contains names of the functions used to establish the ground truth for a specific adverse event that occurs within the scenario

i. Example call:
```
subevent =
GTFunc_1(Idx,params,obsval,rawdata_val);
```

ii. The input arguments are as follows:

A. `Idx` - integer index of the type of file being loaded (either validation or test), *i.e.* File # Idx out of X total validation or Y total test files

B. `params` - a struct containing all relevant fields for the specific run configuration. It needs to be passed in as the second argument and named exactly as "`params`."

C. `obsval` - indexed structure containing all validation or test files, created by an auxiliary ACCEPT function. It needs to be passed in as the second argument and named exactly as "`obsval`." It contains the unnormalized residual formed by the regression stage, contained in the field `obsval.data`. The other two fields contain ground truth data. `obsval.subevent` contains the "raw" ground truth derived from the logical construction of this function. `obsval.event` contains the ground truth vector that is constructed based upon the given prediction horizon.

D. `rawdata_val` - cell array containing all validation or test files, also created by an auxiliary ACCEPT function. It needs to be passed in as the fourth and last argument and named exactly as "`rawdata_val`." It contains the raw parametric data in engineering units, prior to normalization. The first argument, `Idx`, indexes the `rawdata_val` cell array.

iii. The output argument is a binary vector representing the ground truth status for each time step (0 - adverse event absent, 1 - adverse event present).

iv. The body of the function must include the logic necessary to establish the ground truth for a given validation or test data file.

(f) In the file `User_input_ressarch_generic.m`, you will find only the first section needs to be modified if desired, which should be clearly demarcated. Modify the paths accordingly and save as `User_input_ressarch.m` (without the "`_generic`" suffix):

(g) Use the following format and rules for creating the file `Adverse_Event_X_Parameterlist.txt`:

i. All parameter names must be encapsulated with double quotes as such "Parameter 1"

ii. Each line of the file should only contain a single parameter name, followed by a carriage return.

iii. The listed parameter name must match the name of a parameter in the `*.mat` data file.

3. Running ACCEPT

(a) Matlab session

i. `>> ressarch` (runs ACCEPT)

ii. `>> combine_ressarch` (combines results from previous ACCEPT runs)

Appendix D

ACCEPT developer's guide

Steps to add a new regression method in ACCEPT using local resources:

1. In the `regressopt` folder
   - Create new folder for the regression method in the `regressopt` folder, `$FOLDERNAME` (the convention for naming is all common letters).
   - There should be at least two functions in this new regression methods folder. They should be named `buildTEMP.m` and `evalTEMP.m` where you exchange 'TEMP' for the name of your regression method *e.g* `buildELM`, `evalELM` (the convention is all capital letters).
   - `buildTEMP` should take in `trainData` and `runOptions` as parameters and return's a variable that stores the model which was produced by the regression method.
   - `evalTEMP` should take in the model (the variable that was returned from the `buildTEMP` function above) and `tst` as parameters and return the predicted value(s).
2. In `mainREGcode_ressarch.m`
   - Add the name of the regression method to the end of the `algo_name` cell array (the convention is all common letters, no spaces).
   - Add a new `case` block for the new regression method in the switch statement. It should follow the pattern of the others changing the names to be the new regression method where appropriate. Here is where you would place a hyperparameter to tune (see Step 3) and also where to include the `buildTEMP` and `evalTEMP` functions.
3. In `aux_input.m`
   - Add the name of the new regression method to the `algotypes` cell array, it should be the same name as in the `algo_name` array in Step 2.

- Add the name of the hyper-parameter that you want to tune into the `tuneparamtypes` cell array (the convention is all common letters WITH spaces as necessary).
- Add all the other regression's hyper-parameters at the end of the last `if` statement (if any). You must create an `if` statement that follows the pattern of the previous regression methods to add the hyper-parameters. To add the hyper-parameters it should be of the form `params.temp_hyper` where you replace `temp_hyper` with the actual name of the hyper-parameter.

4. In `test_loop_ressarch.m`
   - Add the name of the new regression method (same name in `algo_name` and `algotypes`) to the appropriate `if` or `elseif` statement in the `if` block `if params.regress.optIdx==7`
5. In `reg_ranges.m`
   - Add a value to the end of the `tuneminrange` array (this the minimum value for the hyper-parameter that you want to tune)
   - Add a value to the end of the `tunemaxrange` array (this the maximum value for the hyper-parameter that you want to tune)
   - Add a value to the end of the `tunevals` array (this is a default value that ACCEPT would use when necessary)

To add a new regression method in ACCEPT using a distributed processing configuration, include the following steps:

1. `$CONFIG.Values.job.FileDependencies = 'regressopt', 'regressopt$FOLDERNAME'.`
2. Save as `*.mat` (R2011b or earlier) or `*.settings` (R2012a or later) in `$ACCEPT_DIR/ACCEPT/Distributed Processing/PCTconfigs`

Appendix E

Demo of ACCEPT on a sample problem

E.1 Sample Problem Description

Sustainability Base is a 50,000 square foot LEED Platinum certified building located at NASA Ames Research Center. It seeks to expand the

possibilities of building sustainably on Earth by utilizing NASA's own innovations and technology. Sustainability Base generates data from many systems in the building via sensors and power ports. The data being used as the basis for the example problem in this tutorial is motivated by a scenario devoted to anomalous drops in room temperature in Sustainability Base, and can be found at the DaSHlink website here `https://c3.nasa.gov/dashlink/resources/936/`.

E.2 Tutorial

Preprocessing

As stated in Appendix C, ensure that the `DATA_DIR` and `ACCEPT_DIR` environment variables are set to the directories where you have stored the data and ACCEPT code respectively, *e.g.*

```
> setenv('DATA_DIR','/home/username/ACCEPT_data')
> setenv('ACCEPT_DIR','/home/username/ACCEPT')
```

Next, create `params.txt` and place it in the data directory. All sensors, including both target and input parameters, need to be separated by a carriage return (i.e. the enter/return button on the keyboard). The sensor names should not be separated by a space. For example:

```
DC114T
DCRCS103
DCTN240T
```

In this introductory example we only use continuous-valued data from two randomly selected sensors to predict the target (room temperature) sensor. The two parameters are used to predict the air temperature outside of a conference room on the second floor (sensor label: DCTN240T, Room 227). One of the two parameters corresponds to a sensor measuring the CO_2 concentration for a large conference room on the first floor (sensor label: DCRCS103, Room 103), which is served by a traditional HVAC system. The other parameter corresponds to a sensor measuring the room temperature of an electrical closet on the first floor (sensor label: DC114T, Room 114).

Next, create a name for the ground truth function, *e.g.* `ground_truth.m` and specify that any temperature value falling below 68.1 deg F for the target sensor is anomalous. For example:

```
temp_col = strmatch('DCTN240T ', ...
params.header,'exact');
data = rawdataVal{idx};
truth = data(:,temp_col) < 68.1;
anomIdx = find(truth==1,1); %find the first anomaly
```

Lastly, create a name for the load function, *e.g.* `load_function.m`. Each `*.mat` file contains a structure called `Info`. `Info` contains two fields: one "header" fields containing all sensor names stored in a cell array called `colheaders`, and another field called `data`, containing all the sensor data in a matrix format.

```
load(filelist);
header = Info.colheaders();
data = Info.data();
message = true;
```

Running ACCEPT

Ensure that the correct paths are added to MATLAB's path and run the `ressarch.m` function.

```
> ressarch
```

A dialog box will appear; select `No` so that new results can be created.

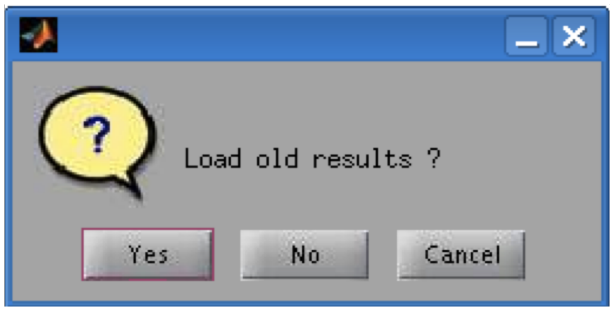


Figure E10: Loading Previous Results

Next, select `Yes` to create a new configuration file.

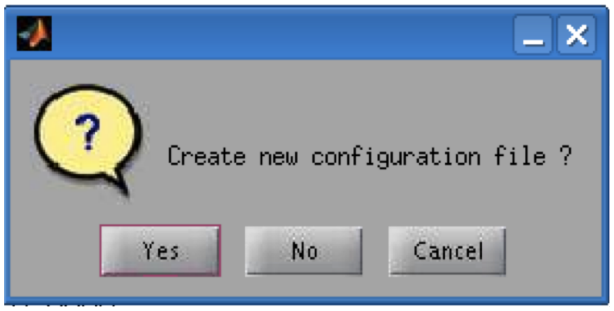


Figure E11: New Configuration File

When asked to pick an anomaly candidate, select `Cold Complaints`, and when asked to select a target parameter, select `DCTN240T`.

When asked to either use the system's local resources or distributed processing via a cluster, select your system's local resources for the purposes of this tutorial (input 2).

```
> 1- Use cluster scheduler 2 - Use local resources :  2
```

When asked to perform regression only, select No (2).

```
> Use regression only ?  1 - Yes, 2 - No :  2
```

When prompted to select the optimization method, select fixed point in order to set pre-optimized values. Selecting any other option would require a great deal more computational effort, and the purpose of this simple example is primarily to get the user up and running with an example that will complete in a short period of time.

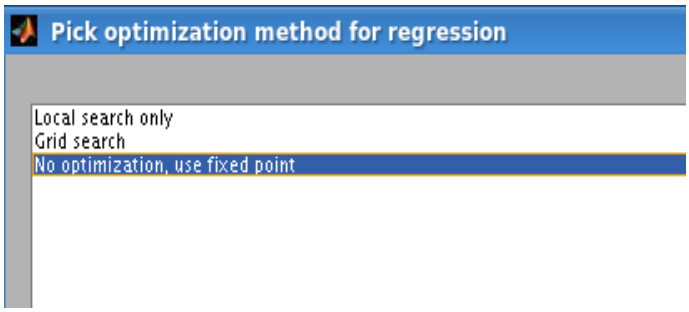


Figure E12: Optimization Methods

When prompted to select regression methods, choose *'lin'* and *'elm'*.

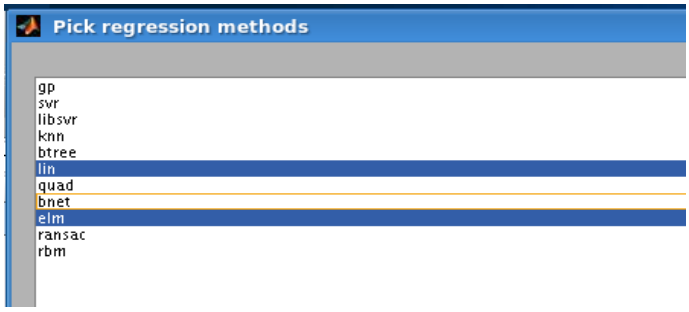


Figure E13: Regression Methods in ACCEPT

When asked to select the type of detection methods to use, choose both 'redline" methods, *e.g.* "Redline-Training" and "Redline-Validtion."

A prompt will appear to allow for entering the optimized values for the selected regression methods; for *'lin'* enter 0.00001 and for *'ELM'* enter 481 (refer to section 4 for further explanation of these hyperparameters). A series of questions based on the detection

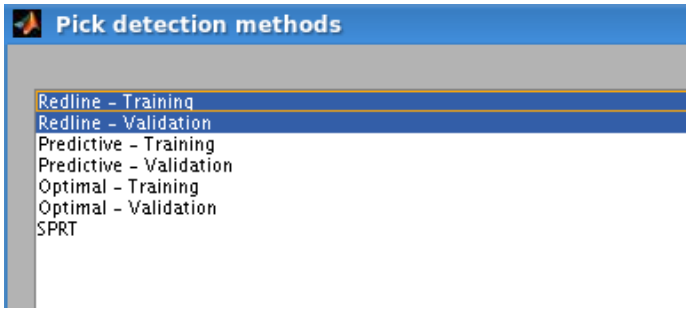


Figure E14: Detection Methods in ACCEPT

methods that were selected will then follow. Use the inputs shown in Figure E15 to answer these questions. See section 5 for details on the context of the first four questions shown in Fig. E15. The remaining two questions can be addressed as follows:

1. Enter resolution (number of points) for Monte Carlo-based integration (smoothness factor): *This option is used for ROC curve construction based upon methods that rely on a model-based approach to generate the relevant ROC curve statistics/performance metrics (labeled* ***X-Training*** *and discussed in Sec. 5). A good default setting for this value has been found to be 3600, however in general smaller values yield a trade-off between less accurate results and less time for the integration to complete. Similarly, higher values yield more accurate results and take more time to complete. The purpose of Monte Carlo integration has been discussed in Sec. 5.*

2. Resolution of ROC curve: *Construction of an ROC curve using "training data only" means that it does not rely on any ground truth labels, and typical computational shortcuts cannot be employed. As such, we must find an alternate way to create the ROC curve (please see Algorithm 1 for a procedural summary to accompany the subsequent narrative). A heuristic approach is taken by splitting the domain of* $P_b \in [0, 1]$ *or* $L_a \in [0, L_{a_{max}}]$ *into two distinct subdivisions. Independent grids are then established for each subdivision, and false alarm/detection probabilities will be computed for each of the points in the grid. Pre-specified tolerance criteria associated with these probabilities will be used to terminate the process before continuing on to create twice as many new subdivisions as the last time. The final result will yield an ROC curve with sufficient resolution meeting the specific tolerance criteria. Refer to Algorithm 1 for the pseudo-algorithmic procedure employed in this process. Clearly, creation of these subdivisions has exponential complexity, so the user should be cau-*

tious about using too high of an integer value. We have found that a value of 10 typically provides sufficient resolution.

Algorithm 1 ROC Curve Construction

function ROC(ϵ, tol)
  $\epsilon \leftarrow Dom_1([\epsilon_0, \epsilon_{max}])$
  $stop \leftarrow N > 3600 \bigvee \max|\Delta(P_{fa)}| < .01 \bigvee \max|\Delta(P_{md)}| < .01$
  while $not(stop)$ **do**
    if $i < tol + 1$ **then**
      $Dom_{i+1} \leftarrow \bigcup_{m=1}^{2^i} Dom_m([\epsilon_{m-1}, \frac{\epsilon_m}{2}] \bigcup [\frac{\epsilon_m}{2}, \epsilon_m]))$
    else
      $Dom_{i+1} \leftarrow \bigcup_{m=1}^{i} Dom_m([\arg\min_\epsilon \neg stop, \arg\max_\epsilon \neg stop])$
    $fa \leftarrow getFalseAlarms(Dom_{i+1})$
    $fd \leftarrow getFalseDetections(Dom_{i+1})$
    $i \leftarrow i + 1$
  $fa \leftarrow cleanFalseAlarms(Dom_{end})$
  $fd \leftarrow cleanFalseDetections(Dom_{end})$
  $fa \leftarrow sortFalseAlarms(Dom_{end})$
  $fd \leftarrow sortFalseDetections(Dom_{end})$

Provide a descriptive name after being prompted to save the configuration. Now ACCEPT will run and produce the desired results.

```
>> ressarch
1 - Use cluster scheduler, 2 - Use local resources : 2
Use regression only ? : 1 - Yes, 2 - No: 2
Global Optimization Toolbox not detected, ACCEPT will use either a grid search for optimizing the regression
Design Alarm System by constraint on 1 - False Alarm Rate, 2 - Missed Detection Rate, 3 - Equal Tradeoff : 3
Use ASOS approximation for LDS learning ? 1 - Yes, 2 - No : 2
Maximum state order = : 2
What is the maximum design (and validation) prediction horizon ? : 30
Enter resolution (number of points) for Monte Carlo-based integration (smoothness factor) : 3600
Resolution of ROC curve (bits) : 10
```

Figure E15: Inputs for ACCEPT

E.3 ACCEPT Results

The following table shows various runtimes for the most computationally intensive elements of ACCEPT. The values are in seconds, and are approximated to the nearest second.

	Method	
	Linear	ELM
Regression	1.8	1.1
LDS Parameters	90	206
Detection	5.8	5.9
Total	**97.6**	**213**

Table E4: Time to Run (seconds)

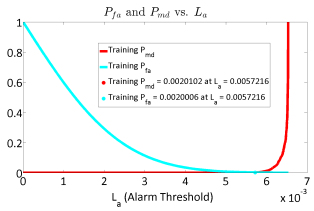
Learning the linear dynamical system is where most of the computation lies for this problem. This is due to the fact that there are $\mathcal{O}(n^2)$ parameters in θ to learn that are associated with the LDS learning process. This stands in distinction to the regression and detection parts of the pipeline which do not suffer from the same level of computational complexity. For more detail on the complexity associated with LDS learning, refer to [28] and [30].

Results		Method	
		Linear	ELM
Regression	Global Optimum	1.1e-05	481
	Optimized Values	0.0000	0.0001
Missed Detection	Redline - Training	0.0000	0.0088
	Redline - Validation	0.0088	0.0088
False Alarm	Redline - Training	0.0476	0.0108
	Redline - Validation	0.0022	0.0108
Detection Time	Redline - Training	2.0000	0.0000
	Redline - Validation	0.0000	0.0000

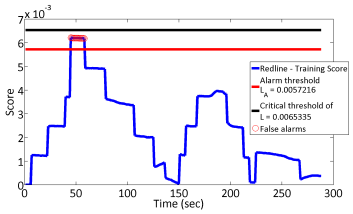
Table E5: Results From ACCEPT

From Table E5, it is clear that the performance of using either linear regression or ELM regression is very similar, and the small differences are negligible with respect to system performance. As such, Figs. E16 and Fig. E17 illustrate only the results for linear regression.

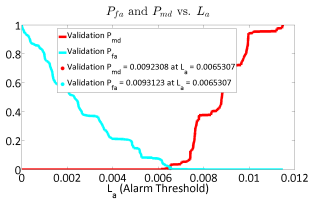
The reproduction of similar results should be possible by following the step-by-step instructions provided for this same exercise. Please contact the authors if dramatically different results were generated, which may



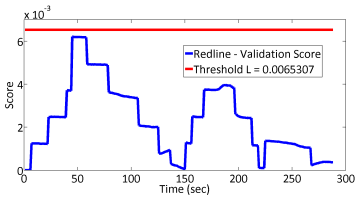
(a) Redline - Training



(b) Redline - Training

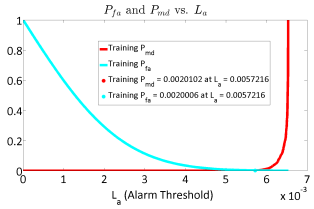


(c) Redline - Validation

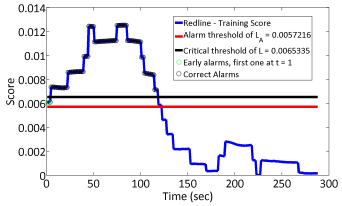


(d) Redline - Validation

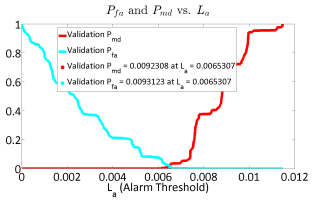
Figure E16: Detection Results for Linear Regression (LR1), Nominal Realizations



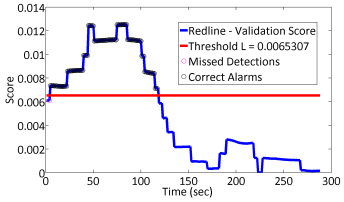
(a) Redline - Training



(b) Redline - Training



(c) Redline - Validation



(d) Redline - Validation

Figure E17: Detection Results for Linear Regression (LR1), Anomalous Realizations

be possibly be due to inconsistencies between the version of the code posted to DaSHlink and the ones used to generated the results shown here. Although highly accurate and robust advance prediction capability was demonstrated here with "toy" scenario that relies on *real* data, it can also be employed for more realistic scenarios relating to safety and even for other types of mission critical systems. Ultimately, ACCEPT will not only enable real-time advance prediction of critical events, but will also allow for an enhanced ability to choose the best combination of algorithms to address the needs of application-specific performance

requirements.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704–0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-11-2015	Technical Memorandum	

4. TITLE AND SUBTITLE
ACCEPT: Introduction of the Adverse Condition and Critical Event Prediction Toolbox

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Rodney A. Martin, Santanu Das, Vijay Manikandan Janakiraman, Stefan Hosein

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Ames Research Center
Moffett Field, CA 94035-0001

8. PERFORMING ORGANIZATION REPORT NUMBER
L–

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)
NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
NASA/TM–2015–218927

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Category
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES
An electronic version can be found at http://ntrs.nasa.gov.

14. ABSTRACT
The prediction of anomalies or adverse events is a challenging task, and there are a variety of methods which can be used to address the problem. In this paper, we introduce a generic framework developed in MATLAB® called ACCEPT (Adverse Condition and Critical Event Prediction Toolbox). ACCEPT is an architectural framework designed to compare and contrast the performance of a variety of machine learning and early warning algorithms, and tests the capability of these algorithms to robustly predict the onset of adverse events in any time-series data generating systems or processes.

15. SUBJECT TERMS
Alarm systems, Kernel regression, Linear regression, Level-crossing prediction, Early detection

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Information Desk (email: help@sti.nasa.gov)
U	U	U	UU	58	19b. TELEPHONE NUMBER *(Include area code)* (757) 864-9658