Detection and Modeling of high-dimensional Thresholds for Fault Detection and Diagnosis

Yuning He
UARC, NASA Ames Research Center
Moffett Field, CA 94035
Email: yuning.he@nasa.gov

***Abstract*—Many Fault Detection and Diagnosis (FDD) systems use discrete models for detection and reasoning. To obtain categorical values like ”oil pressure too high”, analog sensor values need to be discretized using a suitable threshold. This task is usually performed by the “wrapper code” of the FDD system.**

In practice, selecting the right threshold is very difficult, because it heavily influences the quality of diagnosis. In many cases, proper thresholding needs to be performed using non-linear high-dimensional threshold surfaces to accommodate dependencies between system components and different sensors. Often those dependencies are complex and can not handled analytically.

In this paper, we will describe a statistical modeling technique using hierarchical Bayesian methods for the detection of threshold surfaces using a low number of necessary simulation experiments.

I. Introduction

Many Fault Detection and Diagnosis (FDD) systems use discrete models for detection and reasoning. To obtain categorical values like ”oil pressure too high”, analog sensor values need to be discretized using a suitable threshold. Time series of analog and discrete sensor readings are discretized as they come in before processed by the diagnosis engine. This task is usually performed by the “wrapper code” of the FDD system, together with signal preprocessing and filtering.

In practice, selecting the right threshold is very difficult, because it heavily influences the quality of diagnosis. If a threshold causes the alarm trigger even in nominal situations, false alarms will be the consequence. On the other hand, if threshold setting does not trigger in case of an off-nominal condition, important alarms might be missed, potentially causing hazardous situations.

Usually, each sensor is handled individually and different threshold values might exist for different modes of the plant. For example, the threshold for the oil pressure for a cold engine (mode: cold) might be different from that for a hot engine (mode: hot). For complex industrial systems with hundreds of sensors and dozens of modes, a large number of thresholds must be selected and validated.

The use of a threshold for the discretization of a sensor signal, however, ignores any dependencies and correlations between different signals. Therefore, discretization with individual thresholds can only form a coarse approximation. Essentially, the thresholds form a hypercube in the high-dimensional space of sensor signals. This approach can easily lead to over-conservative settings. In those cases, proper thresholding would need non-linear high-dimensional threshold surfaces to accommodate dependencies between system components and different sensors. Often, however, dependencies between system components and different sensors are complicated and often not fully understood. Domain experts might have an idea of the approximate shape of the envelope, but exact values are unknown. Therefore, experiments need to be carried out to determine the threshold curves. Because of high dimensionality and lack of analytical solutions, straight-forward grid-based methods are not applicable in general.

In this paper, describe an advanced statistical method that uses Bayesian dynamic modeling and on-line learning techniques to estimate threshold surfaces in a high-dimensional space. Once a representation of the threshold surface has been obtained, techniques for fitting its shape and estimate shape parameters [1], [2] can be applied. This approach goes way beyond traditional algorithms, which obtain thresholds in the form of hyper surfaces. By selecting the most likely shape of a surface from a domain-specific “library” and estimating it’s parameters, the domain expert can immediately recognize and understand that shape—a very important prerequisite for Verification and Validation (V&V) of FDD systems. This is in stark contrast to other well-known techniques like neural networks, where this information is hidden in a representation that is not suitable for human understanding. Here, however we will focus on statistical modeling and active learning for the detection of threshold surfaces.

The rest of this paper is structured as follows: in the next section, we will briefly describe Fault Detection and Diagnosis architecture and signal discretization. We also will introduce our example, the analysis of the stall speed for an aircraft. Section III is devoted to our statistical modeling approach and architecture and Section IV focuses on active learning. In Section V we present results of experiments. Section VI concludes and discusses future work.

II. Fault Detection and Diagnosis

Typically, Fault Detection and Diagnosis (FDD) systems are used to continuously monitor complex systems, e.g., an aircraft or spacecraft. Observable information obtained by sensors is used to detect any off-nominal situation and to perform root

978-1-4799-1894-2/15/$31.00 © 2015 IEEE

cause analysis. A number of different approaches for FDD or vehicle health management exist, but for this paper we focus on a very generic architecture as shown in Figure 1. The plant is observed using a number of analog sensors (e.g., pressure, temperature, battery voltage). Each signal is discretized by the wrapper code using thresholds θ in order to obtain discrete values comprising the outcome of a test. For example, for pressure p, $(p < \theta_p) \equiv true$ might indicate a dangerously low pressure. Often, one analog signal is discretized into various discrete ranges like "too low", "nominal", and "too high" using thresholds θ_{low} and θ_{high}. The discrete outcomes of the tests are then fed into the diagnosis engine where hypotheses about the most likely set of failure modes (e.g., pump faulty, fuse open) is produced. That information can then be used to initiate mitigation and recovery actions. Diagnostic engines could be, for example, TEAMS/RT,[1] TFPG [3], [4], or a Bayesian Network [5], just to mention a few. In practice, discretization thresholds are, in most cases defined during design time. There might be different thresholds for different modes or configurations of the plant.

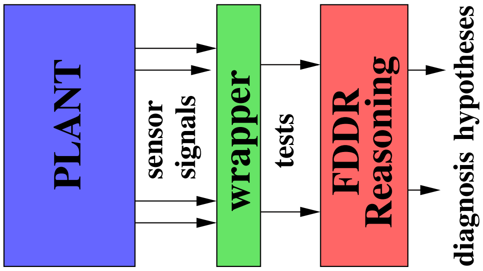

Fig. 1. High-level architecture of an FDD system

A. Example: stall speed

Throughout this paper, we will illustrate our approach with the analysis of the stall speed v_{stall} of an aircraft. For stable flight, the airspeed must always be larger than v_{stall}. Otherwise, the flow over the wing separates and the aircraft will loose altitude at a dramatic rate and is in severe danger to crash. Fault detection and health management systems therefore need to monitor the current speed of the aircraft; discretization might use thresholds θ to determine "nominal", "low-speed", and "very-low-speed" conditions. However, the stall speed depends on numerous factors including altitude, weight, position of flaps, thrust, just to mention a few. For each aircraft, seven stall speeds are given, depending on the mode of the aircraft, typically v_S or v_{S1} for a clean configuration, where flaps and landing gear are retracted, or v_{S0} for an aircraft in landing configuration with fully extended flaps, among others.[2] Usually, the number of modes is kept small.

As the true threshold surface is non-linear, there are considerable possibilities for false alarms as shown in Figure 2. This graph shows airspeed v over a parameter p. The stall speed v_{stall} is non-linear with respect to parameter p and shown as a red curve. Using this curve, a discretization of sensor values into "stable" and "stall" can be made: the aircraft is stalling if its speed is below the red curve. The use of two mode-specific thresholds (green in Figure 2) leaves a large space open for false alarms (shaded): values of v in that range are tagged as "stall" by the mode-specific constant threshold, but v is actually below the true threshold.

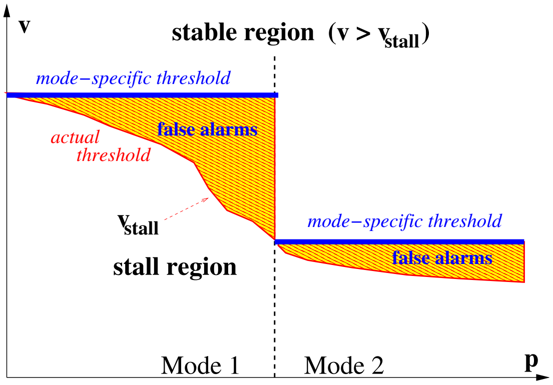

Fig. 2. Mode-specific thresholds (green) for two modes for speed v over a parameter p, curve for actual v_{stall} (red). Areas, where false alarms occur are shaded.

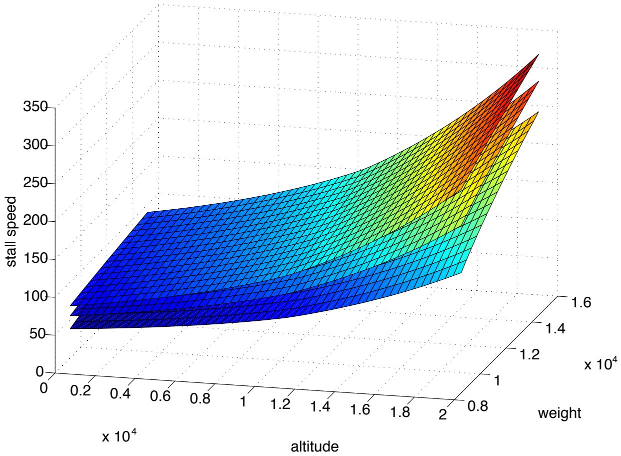

Fig. 3. Threshold surface over airspeed, altitude, and aircraft weight for three different values of thrust.

The stall speed substantially depends on shape and size of wings, the engine nacelle characteristics, as well as fuselage, weight, thrust, and other factors (see Figure 3 for dependency on weight and altitude). In practice, the stall speed for a few different configurations and scenarios, is usually determined using high-fidelity simulators or even test flights. Our goal is therefore to provide statistical methods to estimate the non-linear v_{stall} threshold surface in a high-dimensional space using as few experiments (e.g., simulation runs) as possible.

[1] http://www.teamqsi.com

[2] http://en.wikipedia.org/wiki/Stall_(fluid_mechanics)

III. Statistical Modeling

A. Algorithm Overview

We propose a sequential method for the estimation of parameterized threshold surfaces in high dimensional spaces. We represent our problem as learning the response surface for the function f, where $f(x) = 1 - \epsilon$ for some small $\epsilon > 0$ if the experiment succeeds and $f(x) = 0 + \epsilon$ otherwise. In this representation a threshold surface is determined by points x with $f(x) = 0.5$. This representation allows us to formulate a powerful method to select the next data point, which is explained later.

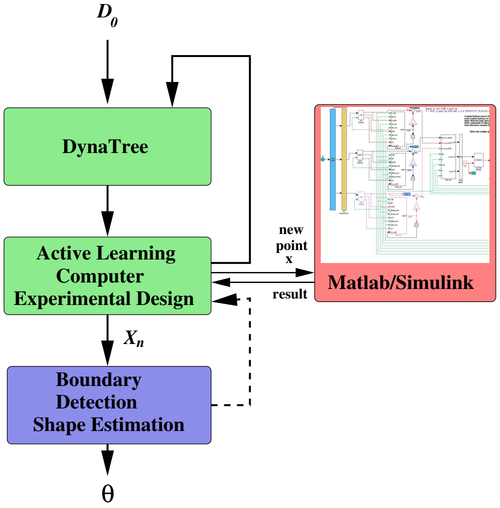

Fig. 4. Overview of active learning architecture

Given an initial set of labeled data D_0, our approach builds a hierarchical Bayesian representation. Using active learning and computer experimental design, the number of required experiments and simulation runs can be kept minimal. The hierarchical representation provides information and confidence intervals for subsequent estimation of shape parameters Θ for the threshold surface. For shape estimation see [2].

The overall process is depicted in Figure 4. The active learning algorithm builds an initial classifier based upon D_0. Then, candidate points (i.e., sets of input parameters) are selected by the algorithm and handed over to the computer experiment, which executes the system under consideration using the given parameters and returns a categorical result (success or failure). Since each run of the simulator can require substantial computational resources, the overall number of new data points should be kept as small as possible. For example, in Figure 4, the experiment is depicted as a Simulink simulation.

Our algorithm is based upon the sequential classification and regression framework as given by DynaTree [6], [7]. It features dynamic regression trees and a sequential tree model. Particle learning for posterior simulation makes Dynatrees a good candidate for applications, where new data points are processed sequentially. At any given point in time, the classifier is represented by a DynaTree. Figure 5 shows the individual steps of our overall algorithm. In the initial phase, a classifier using the data set D_0 is constructed. It provides an initial partitioning of the space and provides the information to estimate posteriors over given sets of data points. The main body is an iterative loop where, by adding new data points, the classifier will be extended and improved with the main goal of identifying and characterizing the threshold surfaces. In the first step, the current classifier is used to estimate a set of data points, which are close to the current prediction of the threshold. These comprise a subset of data points from a regular grid or a Latin hyper square, for which their entropy measure is high (classification representation) or the estimated response value is close to 0.5. The location of these points do not only depend on the actual boundary, but also on the shape of the dynamic tree and the size of the partitions, because points in the same partition have the same values. This set of data points is then used to estimate the best parameters Θ for each of the boundary shapes, together with a confidence interval for each of the parameters.

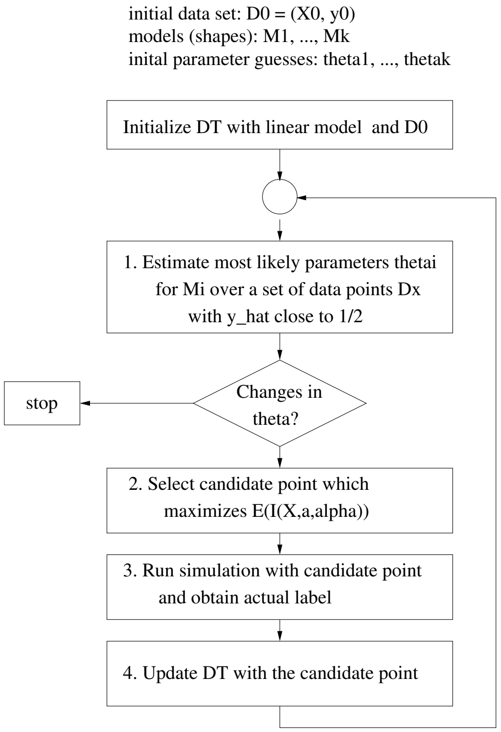
initial data set: D0 = (X0, y0)
models (shapes): M1, ..., Mk
inital parameter guesses: theta1, ..., thetak
Initialize DT with linear model and D0
1. Estimate most likely parameters thetai for Mi over a set of data points Dx with y_hat close to 1/2
Changes in theta?
stop
2. Select candidate point which maximizes E(I(X,a,alpha))
3. Run simulation with candidate point and obtain actual label
4. Update DT with the candidate point

Fig. 5. Overview of active learning procedure

The candidate point selection in this active learning algorithm can use as much information as is available at the current stage, for example, information and entropy of the current data set. It then selects a new point (i.e., set of input parameters), for which the label is obtained by running the system simulator. Next we present the individual steps in detail.

IV. Active learning and Expected Improvement

A. Finding threshold surfaces

Each data point describing one simulation run (experiment) is defined as $\mathbf{x} = \langle P_1, \ldots, P_p \rangle$, where P_i are the input parameter settings and the outcome $o(\mathbf{x}) \in \{pass, fail\}$. Thus these data points define a classification problem with $C = 2$ classes. Informally, a boundary can be found between regions, where all experiments yield passing tests $p(\mathbf{x} = pass) = 1$ and those, where the experiments do not meet the success criterion $p(\mathbf{x} = fail) = 1$. Therefore, we can define a point $\mathbf{x}$ to be on the boundary if $p(\mathbf{x} = pass) = p(\mathbf{x} = fail) = 0.5$. Although this condition can easily be generalized to more than two classes, in this work, we will focus on $C = 2$.

A common metric to characterize points on the boundary is based upon the entropy. The entropy $entr = -\sum_{c \in c_1,..,c_C} p(\mathbf{x} = c) \log p(\mathbf{x} = c)$ becomes maximal at the boundary. In cases of more than two classes, [8] uses a BVSB (Best vs. Second Best) strategy. [9] defines a metric *advantage* as essentially $adv(\mathbf{x}) = |p(\mathbf{x} = pass) - p(\mathbf{x} = fail)|$ and considers points with minimal advantage to be close to the boundary.

In general, there are two basic methods: explicitly from knowledge of the classification function, or by treating the classifier as a black box and finding the boundaries numerically. For some classifiers it is possible to find a simple parametric formula that describes the boundaries between groups, for example, LDA or SVM. Most classification functions can output the posterior probability of an observation belonging to a group. Much of the time we do not look at these, and just classify the point to the group with the highest probability.

Points that are uncertain, i.e., have similar classification probabilities for two or more groups, suggest that the points are near the boundary between the two groups. For example, if a point is in Group 1 with probability 0.45, and in Group 2 with probability 0.55, then that point will be close to the boundary between the two groups. We can use this idea to find the boundaries. If we sample points throughout the design space we can then select only those uncertain points near boundaries. The thickness of the boundary can be controlled by changing the value, which determines whether two probabilities are similar or not. Ideally, we would like this to be as small as possible so that our boundaries are accurate. Some classification functions do not generate posterior probabilities. In this case, we can use a k-nearest neighbors approach. Here we look at each point, and if all its neighbors are of the same class, then the point is not on the boundary and can be discarded. The advantage of this method is that it is completely general and can be applied to any classification function. The disadvantage is that it is slow ($O(n^2)$), because it computes distances between all pairs of points to find the nearest neighbors. In general, finding of the boundaries faces the "curse of dimensionality": as the dimensionality of the design space increases, the number of points required to make a perceivable boundary (for fitting or visualization purposes) increases. This problem can be attacked in two ways, by increasing the number of points used to fill the design space (uniform grid or random sample), or by increasing the thickness of the boundary.

B. Active Learning

Computer simulation of a complex system like those discussed above, is frequently used as a cost-effective means to study complex physical and engineering processes. It typically replaces a traditional mathematical model in cases where such models do not exist or cannot be solved analytically.

Active learning, or sequential design of experiments (DOE), in the context of estimating response surfaces (in our case boundaries), is called adaptive sampling. Adaptive sampling starts with a relatively small space-filling input data, and then proceeds by fitting a model, estimating predictive uncertainty, and then choosing future samples with the aim of minimizing some measure of uncertainty, or trying to maximize information. We perform active learning with new data until the boundary is characterized with sufficient accuracy and confidence, and the whole space has been sufficiently explored to not miss any boundary areas in the space.

Consider an approach which maximizes the information gained about model parameters by selecting the location $\mathbf{x}$ which has the greatest standard deviation in predicted output. This approach has been called ALM for Active Learning-Mackay, and has been shown to approximate maximum expected information designs [10]. An alternative algorithm is to select Σ^2 minimizing the expected reduction in squared error averaged over the input space [11], called ALC for Active Learning-Cohn. Rather than focusing on design points which have large predictive variance, ALC selects configurations that would lead to a global reduction in predictive variance.

The ALM/ALC algorithms are suitable for classification but not primarily for boundary detection [12]. These heuristics are in general not suited for the boundary-finding task because they do not take the specifics of the boundaries into account and they tend to also explore sparsely populated regions far away from current boundaries.

C. Our New Boundary Expected Improvement

Finding a boundary between two classes can be considered as finding a contour with $a = 0.5$ in the response surface of the system response. Inspired by [13] and work on contour finding algorithms, we loosely follow [14], and define our heuristics by using an improvement function. In order to use the available resources as efficiently as possible for our contour/boundary finding task, one would ideally select candidate points which lie directly on the boundary, but that is unknown. Therefore, new trial points x are selected, which belong to an ϵ-environment around the current estimated boundary. This means that $0.5 - \epsilon \leq \hat{y}(x) \leq 0.5 + \epsilon$ for $\epsilon > 0$. $\hat{y}(x)$ is the learned estimate of the response function at x. New data points should maximize the information in the vicinity of the boundary. Following [13] and [14], we define an improvement function for x as

$$I(X) = \epsilon^2(x) - \min\{(y(x) - 0.5)^2, \epsilon^2(x)\} \qquad (1)$$

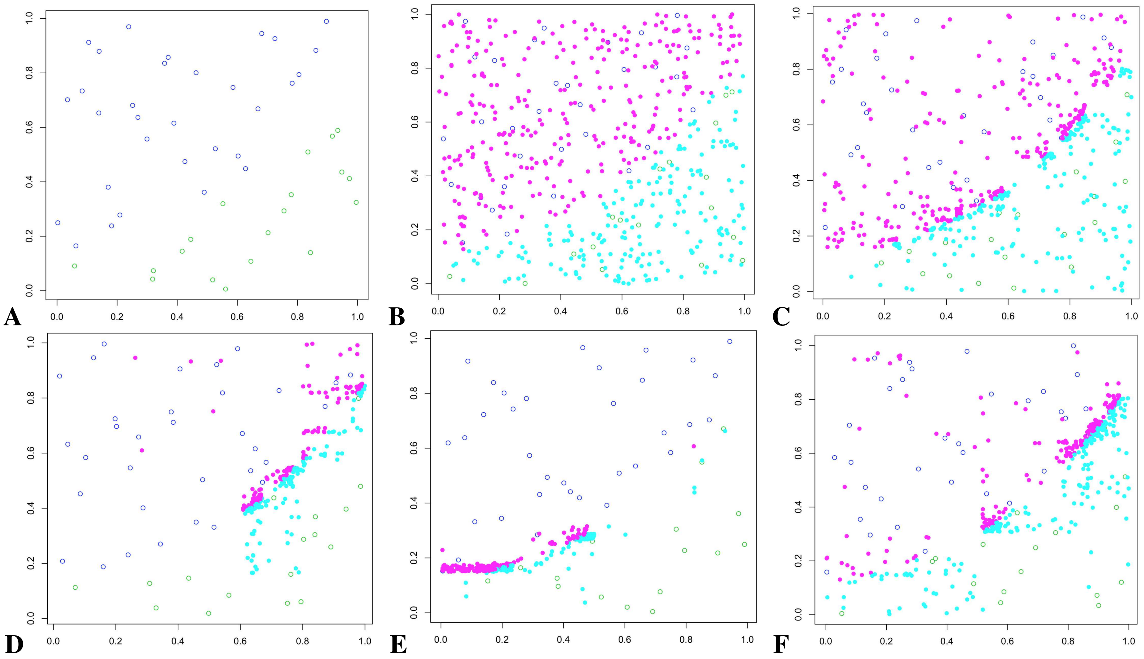

Fig. 6. Selection of candidate points during active learning. Shown is normalized airspeed over normalized altitude, starting from a random initialization (A). Different selection strategies are: random selection (B), ALC (C), ALM (D), EI (E), and our boundary-EI (F)

here, $y(x) \sim N(\hat{y}(x), \sigma^2(x))$, and $\epsilon(x) = \alpha\,\sigma(x)$ for a constant $\alpha \geq 0$. This term defines an ϵ-neighborhood around the boundary as a function of $\sigma(x)$. This formulation makes it possible to have a zero-width neighborhood around existing data points. For boundary sample points, $I(X)$ will be large when the predicted $\sigma(x)$ is largest.

The expected improvement $E[I(x)]$ can be calculated easily following [14] as

$$E[I(x)] = -\int_{0.5-\alpha\sigma(x)}^{0.5+\alpha\sigma(x)} (y-\hat{y}(x))^2 \phi\left(\frac{y-\hat{y}(x)}{\sigma(x)}\right) dy \qquad (2)$$
$$+2(\hat{y}(x)-0.5)\sigma^2(x)\left[\phi(z_+(x)) - \phi(z_-(x))\right]$$
$$+(\alpha^2\sigma^2(x) - (\hat{y}(x)-0.5)^2)\left[\Phi(z_+(x)) - \Phi(z_-(x))\right],$$

where $z_\pm(x) = \frac{0.5-\hat{y}(x)}{\sigma(x)} \pm \alpha$, and ϕ and Φ are the standard normal density and cumulative distribution, respectively. Each of these three terms are instrumental in different areas of the space. The first term summarizes information from regions of high variability within the ϵ-band. The integration is performed over the ϵ-band as $\epsilon(x) = \alpha\,\sigma(x)$. The second term is concerned with areas of high variance farther away from the estimated boundary. Finally, the third term is active close to the estimated boundary. After the expected improvement has been calculated, the candidate point is selected as the point, which maximizes the expected improvement.

V. Experiments and Results

We illustrate the evaluation of our learning architecture using the above stall speed example. For simplicity of experiments, we did not use a full aircraft simulator but rather an analytic approximation [15]. Figure 6 visualizes our approach in two dimensions (airspeed over altitude). We start with a low number of randomly selected points, which are shown as circles in Figure 6A. From there, the active learning procedure selects $N = 500$ new data points. Experiment outcomes are shown in cyan (failure) and magenta (pass). N has been selected this large for visualization purposes. The figure shows, how the different selection algorithms behave in this situation. Our goal is to find many data points near the threshold curve in order to enable accurate representation and to facilitate subsequent shape estimation. On the other hand, the entire area should be considered as well in order not to miss any other boundary curve.

The random Monte-Carlo style selection (Figure 6B) meets both requirements but needs a prohibitively large N for reasonable results. The classical approach ALC ([11], Figure 6C) finds many points near the threshold curve, but still too many data points are away from the curve, demanding large N. Other algorithms are too localized and do not even explore the entire threshold curve (Figure 6D, E). Our approach (F) tries to find a suitable balance between both requirements.

There is a trade-space between closeness of selected points to the threshold surface and a good curve coverage. For example, a greedy algorithm might always select the same point near the threshold (high closeness) but extremely low coverage. Figures 7 and 8 show visualizations of this trade space. For each of the newly added points, we calculate d as the minimal distance of that point to the threshold surface. Obviously, small values should be preferred, as such points close to the threshold help to accurately estimate its shape.

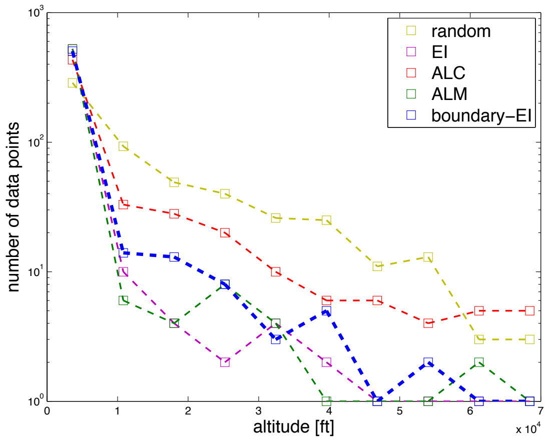

Fig. 7. Histograms for distance d of candidate points from the threshold surface for different update rules (2D). Leftmost bars are cropped for better visibility.

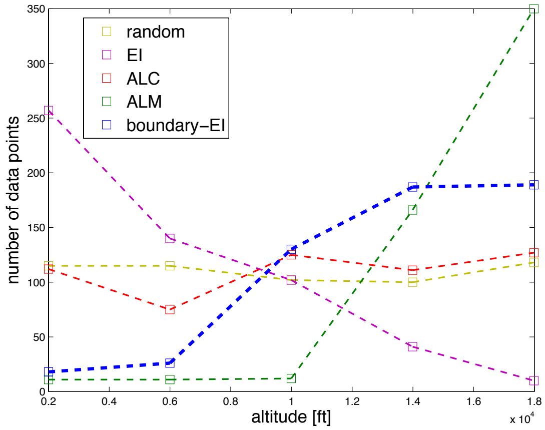

Fig. 8. Histograms of curve coverage (number of candidate points at given parameter value for different update rules (2D).

Figures 7 shows a histogram of distance d for various update rules. Whereas random and ALC have a large number of points that are far away from the threshold surface, ALM seems to perform best for this metric. However, Figure 8 reveals that ALM only covers a very small portion of the threshold surface. Random selection provides the best coverage here. With our analysis goal in mind, our boundary-aware EI metric features a good overall coverage and a high density of points close to actual threshold surface.

Our metric is parameterized by the parameter α, see Equation (2). This parameter influences the width of the "band" around the threshold surface that is considered for the selection of the candidate point. Figure 9 shows runs with several values of α. It seems that values around $\alpha = 0.8$ produce the best results; values of α that are too small or too large tend to lead to a situation, where the new points are located too far from the threshold surface.

The detection and characterization of a (high-dimensional) threshold surface requires a substantially larger overhead during design time and during actual diagnosis than a constant threshold. Therefore, it needs to be considered if the improvement in detection accuracy justifies the additional effort. Detection accuracy is usually defined in terms of false and missed alarms. A false alarm is a situation where the selected threshold is crossed, but the value is still acceptable with respect to the true threshold surface. For analysis of this behavior, Receiver Operating Characteristic (ROC) curves [16] can be used. For a given set of experiment, they show the true positive rate over the false positive rate. For a perfect classifier, the curve immediately rises to 1 and continues horizontally. A random classifier produces the black diagonal line. Figure 10 shows the ROC curves with respect to variability in one parameter, the aircraft weight w. We assume a nominal weight of $w_0 = 12,000$lbs. All other input parameters are kept fixed. If the aircraft weight is known and fixed and we use a

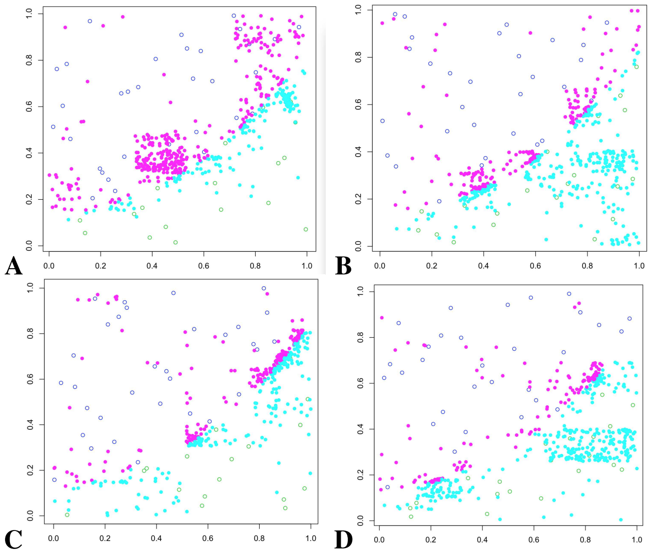

Fig. 9. Boundary-EI with different parameters for $\alpha = 0.2, 0.5, 0.8, 1$

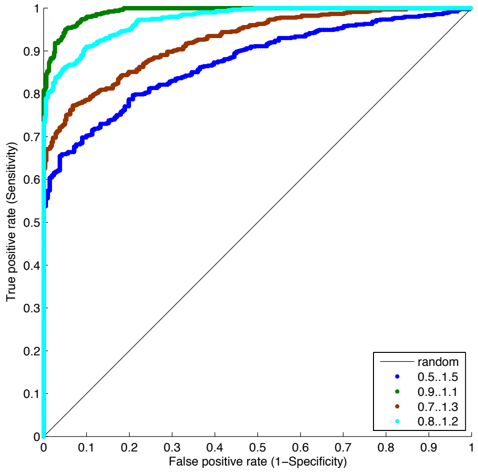

Fig. 10. ROC curves for different weight deviations from $w_0 = 12000lbs$. A simple constant threshold must tolerate more false alarms for larger deviations of the weight.

constant threshold or we use our nonlinear threshold surface, then the classifier is perfect. Possible variations in the aircraft weight lead to a situation as indicated in Figure 2: the actual threshold surface deviates from the constant threshold. Thus, false alarms are possible. The more the actual value can deviate from w_0, more false alarms will occur, reducing the accuracy. In this figure, we let the weight deviate up to (an unrealistic) $\pm 50\%$.

This ROC analysis can be used as an important tool during FDD design to see if constant thresholds are sufficient for the required detection accuracy and to assess the quality of the detected thresholding surfaces.

VI. Conclusion

In this paper, we focused on the discretization of sensor values for discrete Fault Detection and Diagnosis systems. In many cases, threshold surfaces are nonlinear and have complex dependencies between system components and different sensors. Therefore, in practice, selecting the right threshold is very difficult to avoid false or missed alarms.

We described the underlying statistical modeling techniques using hierarchical Bayesian models and an active learning algorithm to obtain data for a potentially high-dimensional threshold surface with a low number of necessary experiments. We developed a candidate selection rule that is aware of the specific threshold surface. Using a small aeronautics example, we illustrated the behavior of the active learning procedure and discussed metrics on the quality of the set of data points for subsequent shape estimation. We have used ROC curves to demonstrate the effect of using constant thresholds (or mode-dependent thresholds) versus an approximated threshold surface with respect to missed and false alarms.

Future work will address the synergistic combination our approach with techniques to extract a compact representation of the threshold surface that can be easily integrated into the wrapper code of real-time diagnostic systems. Of major importance will be the development of effectiveness metrics in the high-dimensional case. Finally, we aim to evaluate our approach with a realistic FDD application.

References

[1] Y. He, "Online Detection and Modeling of Safety Boundaries for Aerospace Applications using Active Learning and Bayesian Statistics," *Proc. IJCNN*, 2015.

[2] Y. He, "Variable-length Functional Output Prediction and Boundary Detection for an Adaptive Flight Control Simulator," Ph.D. dissertation, University of California at Santa Cruz, 2012.

[3] S. Abdelwahed, A. Dubey, G. Karsai, and N. Mahadevan, "Model-based tools and techniques for real-time system and software health management," *Machine Learning and Knowledge Discovery for Engineering Systems Health Management*, p. 285, 2011.

[4] N. Mahadevan and G. Karsai, "FACT Tool Suite," https://fact.isis.vanderbilt.edu/, 2000–2014.

[5] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann, 1988.

[6] M. A. Taddy, R. B. Gramacy, and N. G. Polson, "Dynamic Trees for Learning and Design," *Journal of the American Statistical Association*, vol. 106, no. 493, pp. 109–123, 2011. [Online]. Available: http://EconPapers.repec.org/RePEc:bes:jnlasa:v:106:i:493:y:2011:p:109-123

[7] R. B. Gramacy, "TGP: An R package for Bayesian nonstationary, semiparametric nonlinear regression and design by treed Gaussian process models," *Journal of Statistical Software*, vol. 19, no. 9, pp. 1–46, Jun. 2007, http://www.jstatsoft.org/v19/i09/paper. [Online].

[8] R. Gramacy and N. Polson, "Particle learning of Gaussian process models for sequential design and optimization," *Journal of Computational and Graphical Statistics*, vol. 20, no. 1, pp. 467–478, 2011.

[9] H. Wickham, "Practical tools for exploring data and models," Ph.D. dissertation, Iowa State, 2008.

[10] D. J. C. MacKay, "Information–based objective functions for active data selection," *Neural Computation*, vol. 4, no. 4, pp. 589–603, 1992.

[11] D. A. Cohn, "Neural network exploration using optimal experimental design," *Advances in Neural Information Processing Systems*, vol. 6, no. 9, pp. 679–686, 1996.

[12] R. B. Gramacy, "Bayesian treed Gaussian process models," Ph.D. dissertation, University of California at Santa Cruz, Dec. 2005, http://faculty.chicagobooth.edu/robert.gramacy/papers/gra2005-02.pdf.

[13] D. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black box functions," *Journal of Global Optimization*, vol. 13, pp. 455–492, 1998.

[14] P. Ranjan, D. Bingham, and G. Michailidis, "Sequential experiment design for contour estimation from complex computer codes," *Technometrics*, vol. 50, no. 4, pp. 527–541, 2008.

[15] B. P. Anderson, "Equations for stall speed," 2007. [Online]. Available: http://www.electraforge.com/brooke/flightsims/aces_high/stallSpeedMath/stallSpeedMath.html

[16] T. Fawcett, "An introduction to {ROC} analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861 – 874, 2006, ROC Analysis in Pattern Recognition. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016786550500303X