

Fusing Binary and Continuous Output of Multiple Classifier

Kai F. Goebel

GE Global Research Center
One Research Circle
K1-5C4A
Niskayuna, NY 12309, USA.
goebelk@crd.ge.com

Weizhong Yan

GE Global Research Center
One Research Circle
K1-5B34B
Niskayuna, NY 12309, USA.
yan@crd.ge.com

Abstract - *This paper describes a fusion architecture and implementation for classifiers with binary as well as continuous output. The fusion scheme is represented as a multi-layered architecture which structures the approach into pre-processing, analysis, and post-processing. All components are partitioned into layers where each layer performs logical tasks. In particular, the pre-processing component is partitioned into a temporal layer and a scaling layer. The analysis component is partitioned into strengthening and weakening layers. The post-processing component is structured into suppression and exception handling layers. Manipulations within these layers transform the binary classifier output as well as continuous output into a single continuous domain. Because the fusion architecture is designed in a modular fashion, additional modules can be relatively easily be added or removed. The modular design also allows re-use of the core fusion engine for other domains. We show results of this architecture applied to a system monitoring environment of industrial equipment.*

Keywords: Information fusion, decision fusion, classification, diagnostics, monitoring.

1 Introduction

Remote services operations are information based technology service provider which furnish timely and accurate operational and system knowledge through the evaluation of in-service equipment data. The classifier tool suite that remote service operations use for monitoring performance are often times an amalgam of different individual tools that were put into operation to satisfy particular monitoring and performance needs. However, different monitoring tools generate a large number of alarms daily. Among these alarms, only a very small portion (less than 1%) are actionable alarms, i.e., real fault(s) somewhere in the system which would cause a maintenance action. The rest of them are false alarms, nuisance alarms, and redundant alarms from the individual tools. Manually analyzing the sheer number of alarms everyday and determining which alarms are real and relevant is a great deal of burden for the analysts. As a result, often times the alarm generating tools are switched

off altogether because there is no way to follow up on every alarm. This is clearly not the optimal way to deal with potential problems. Rather, a mechanism is needed that makes decisions based on the outputs from all individual tools and some secondary information so that it alerts the service engineers only when a truly suspicious situation arises and suppresses false and repeated alarms.

Such a decisioning scheme holds the promise to deliver a result that is better than the best result possible by any one classifier used. In part this can be accomplished because redundant information is available that when combined correctly improves the estimate of the better tool and compensates for the shortcomings of the less capable tool. However, applying the data fusion principles of Hall and Garga [1] to decision level fusion, this must not be understood as an invitation generally to use poorly performing classifiers. There is no substitute for a good diagnostic tool and, ordinarily, multiple, marginal-performance tools do not necessarily combine to produce an improved result and in fact may worsen the outcome.

Classification is a field that spans a wide range of disciplines such as pattern recognition, modeling, diagnosis, clustering, etc. Often times, problems not typically associated with classification tasks are re-phrased to allow use of tools from the classification toolbox. One problem that can make classification a hard task is the dynamic nature of many systems. As their signature changes, they escape the carefully partitioned regions. Efforts such as adaptive classification [2] try to cope with this challenge. An added level of complexity comes through the integration and aggregation of information of redundant information which has to be ranked based on their integrity and correlation [3]. In case of dynamic systems, the aggregation scheme has to deal with temporally disjoint information. That is, although relevant, information is produced at different sampling frequencies. Some classification tools may operate at a millisecond sampling period while others give only one estimate at a specific phase of operation of the system. This is a problem if a significant event occurs between the estimate of one classifier and the estimate of the other

classifier. The fusion tool does of course not know whether an event occurs. As far as it is concerned, either one estimate is bad (and it does not know a priori which one that is) or both tools are right and something has happened. But even in the static case, that is, when classifiers give their opinion at the same time, there are a number of issues involved with information fusion. If several tools agree on the class the resulting output should be done with more confidence. However, if tools disagree, one has to decide which tool to believe or to what degree. In addition, information might be expressed in different domains, such as probabilistic information, fuzzy information, binary information, weights, etc. The fusion scheme has to map the different domains into a common one to be able to properly use the encoded data. Of particular interest in this investigation is the integration of information from both continuous output classifiers and binary output classifiers.

We consider classifier problems where a feature vector $x \in \mathfrak{R}^p$ is to be labeled into one or more of c classes. In order to achieve high overall performance of the fault detection function, the performance of each individual classifier has to be optimized prior to using it within any fusion schemes. The optimization is performed with respect to selecting appropriate parameters and – where applicable – structure that govern the performance.

		Classes assigned by Classifier	
		0	1
True Classes	0	N^{00}	N^{01}
	1	N^{10}	N^{11}

Figure 1: Typical 2-class confusion matrix

For each classifier, a confusion matrix C can be generated using the labeled training data [4]. The confusion matrix lists the true classes c versus the estimated classes \hat{c} . Because all classes are enumerated, it is possible to obtain information not only about the correctly classified states (N^{00} and N^{11}), but also about the false positives (N^{01}) and false negatives (N^{10}). The top-left entry of the confusion matrix is dedicated to the normal case N^{00} . The first row – except the first entry – contains the N^{01} . The off-diagonal elements – except the first row – contains the N^{10} . Sometimes a further distinction is made between false negatives and false classifieds where the false classifieds are defined to be the off-diagonal elements of the confusion matrix except the first row and the first column. A typical 2-class confusion matrix C is shown in Figure 1.

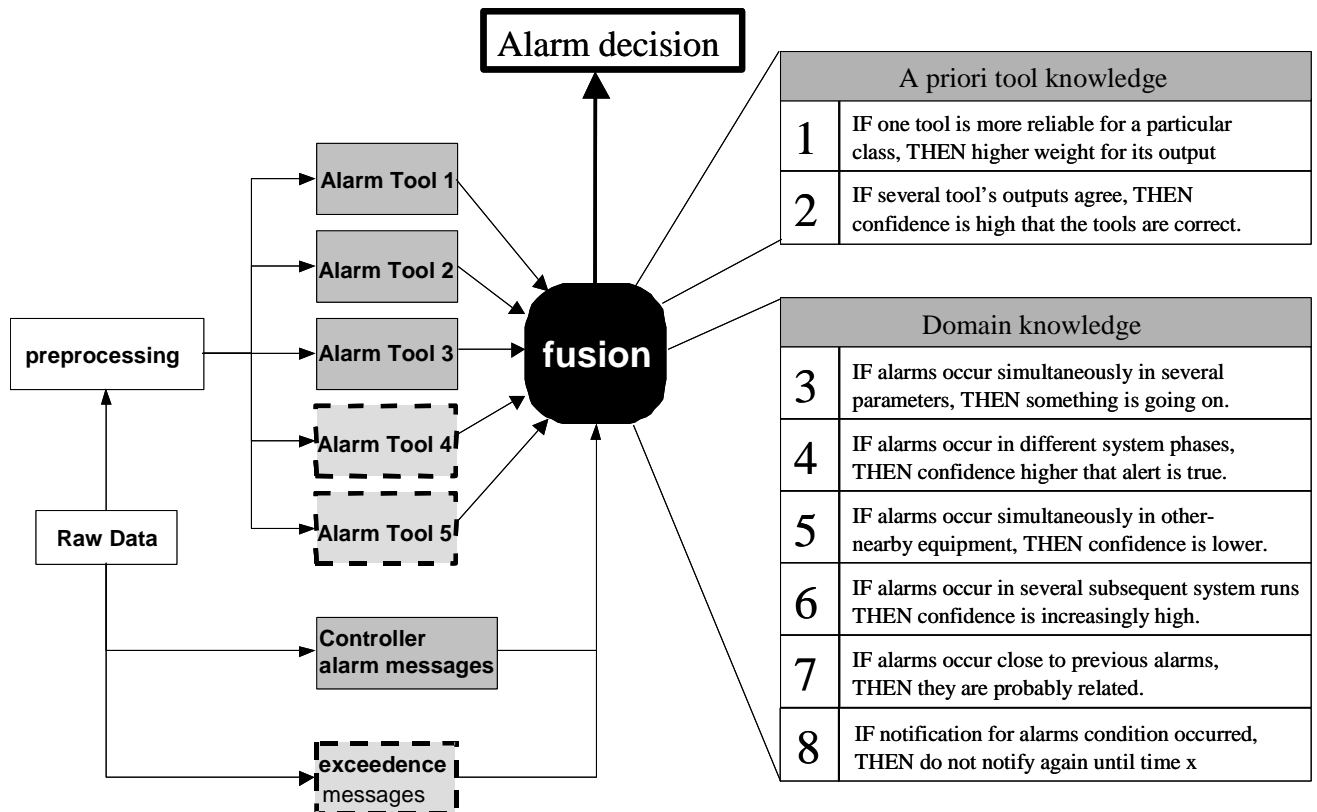


Figure 2 : Fusion Concept

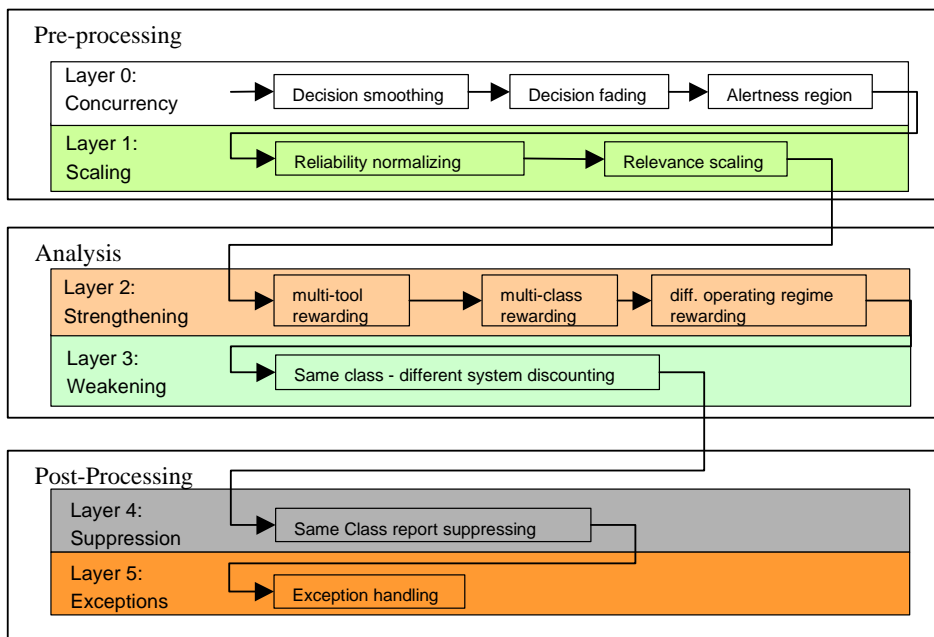


Figure 3 : Fusion Architecture

2 Solution Approach

The primary goal of the fusion scheme is to establish alarm confidence factors, severity and priority while reducing the number of alarms sent to the analysts by integrating the outputs from the different alerting and alarming tools as well as information from other sources (such as fault codes etc.). Secondary domain specific information is also utilized to further improve the alarm reliability. Critical design requirements are: 1) reducing the number of false positive alarms; 2) increasing the alarm reliability and confidence; and 3) not increasing the level of false negatives.

Figure 2 shows the scheme of the alarm fusion concept proposed. It consists of two fundamental integration functions: integrating the outputs of individual alarming tools and integrating domain knowledge. The two integration functions operate independently. While the alerting tool outputs are fed to the fusion box as inputs, domain knowledge is coded inside the fusion algorithms. Such independent integration design provides users the flexibility of adding more classification tools and expanding domain knowledge.

Unlike previous approaches such as Bagging and boosting [4], Dempster-Shafer [5], Model-based approaches [6], fuzzy fusion [7] or statistics based approaches [8], we employ a sequential and parallel multi-layered configurations strategy [9]. In particular, we propose a hierarchical, multi-layer architecture [10] to implement the fusion concept. It manipulates the information from the classification tools (initially) and the fused estimate (later) sequentially until the most likely candidate class has been refined. This process increases and decreases the weight given to the classes according to the strategies

implemented in the respective layers of the fusion process. This implies also that it is possible for some classes to emanate as winners from one layer, only to be overruled in the next layer. The architecture displayed in Figure 3 gives an overview over how the fusion is performed conceptually. In particular, there are a total of 5 layers. They are grouped into 3 major components : 1.) Pre-processing ; 2.) Analysis ; and 3.) Post-Processing. Details of each layer will be given in the following section.

2.1 Major Components: Preprocessing

The Preprocessing component deals with manipulations of the classifier output before the first fusion is performed. The layers of this component also deal with the integration issues of binary and continuous classifier output. This is done primarily in layer 0 and layer 1 that deal with temporal issues such as classifier disagreement when classifiers are subject to different update rates as well as scaling the classifier output according to classifier performance.

2.1.1 Concurrency (Layer 0)

Fundamentally, all output types are converted into the continuous domain. This is accomplished by the decision smoothing and decision fading operations.

2.1.1.1 Decision smoothing

As mentioned before, the challenge in dynamic systems is in providing a different reaction to situations where decisions agree and situations where decisions disagree. When decisions agree, and in the absence of evidence to the contrary, we postulate there is no reason for the fusion agent to change the collective opinion within that time step (there is still a minute chance of joint incorrect

classification). However, if tools disagree, the fusion main module has to decide whether one tool is correct and the other is not (and which) or whether an event has occurred between the two tools. To help in this situation, we try to support the fusion main module by removing outliers and generally by smoothing decisions of individual tools in situations of agreement and by updating decisions quickly when a changed event is indicated. This necessitates the need to provide a system status recognition tool that allows the two different strategies to work side by side.

We implement the concept of decision smoothing via an exponential averaging time series filter with adaptive smoothing parameter [11, 12]. Let $x(k)$ and $x(k-1)$ be the filtered decisions at time k and $k-1$, respectively, α be the smoothing parameter bounded between a lower and upper value to ensure that the filter becomes neither too sluggish nor too responsive, i.e., $\alpha \in [lowerbound, upperbound]$, where *lowerbound* and *upperbound* are tunable parameters), and $y(k)$ is the new incoming decision. Then the filtered decision can be expressed as

$$x(k) = \alpha x(k-1) + (1-\alpha)y(k) \quad (1)$$

Changes of the smoothing parameter will allow weeding out noise and outliers when no fault has occurred but reacting quickly to changes from one event to another. Therefore, if everything is calm, the smoothing parameter is large (remove noise); if something is happening, the strategy is to be more cautious, and to reduce the smoothing parameter value. The reduction is accomplished with the help of an intermittency factor. This factor can be interpreted as a sentinel that monitors system changes. The sentinel checks if the status of a decision changes and keeps track of these changes. First, a change counter establishes whether there is a change of opinion compared to the last one. If the change counter is greater than zero, an intermittency factor is computed dividing the change counter by some user defined sentinel window, i.e.,

$$intermittency = \frac{change_counter}{sentinel_window} \quad (2)$$

The sentinel window is a running window that is small initially to allow operation even when there is only one measurement. This implies that initially, the parameter will be more receptive to changes. When more values become available, the window increases and the decision smoothing becomes more temperate. This also means that in general, more information is better under this paradigm because longer and more refined smoothing can take place. The smoothing parameter α is calculated by

$$\alpha = (1 - intermittency)^{constriction_exponent} \quad (3)$$

where the *constriction_exponent* scales α and is a value > 1

Finally, a forgetting factor is employed that has the effect of reducing the sensitivity of the parameter when no changes have been observed for a while thus allowing the algorithm to settle to its smoothing task. Let the forgetting factor be *forgetting_factor* $\in]0,1[$. Then *change_count* is computed over time as

$$change_count = change_count \cdot forgetting_factor \quad (4)$$

The smoothing operation also contributes to the conversion of binary classifier output to the continuous domain. Binary and continuous output classifiers behave similarly over time. Consider random binary and random white noise. Both mean statistics converge to 0.5. The smoothing operation accelerates this process depending on the value of the smoothing parameter. Figure 4 shows random binary input and random continuous values and their smoothed values initialized at 1 over time. Both filtered values average around 0.5 (in this case the specific means after 100 samples 0.5000 and 0.4837 for the binary and continuous case, respectively). The filtered values after 100 samples are 0.5882 and 0.5098 with $\alpha=0.95$.

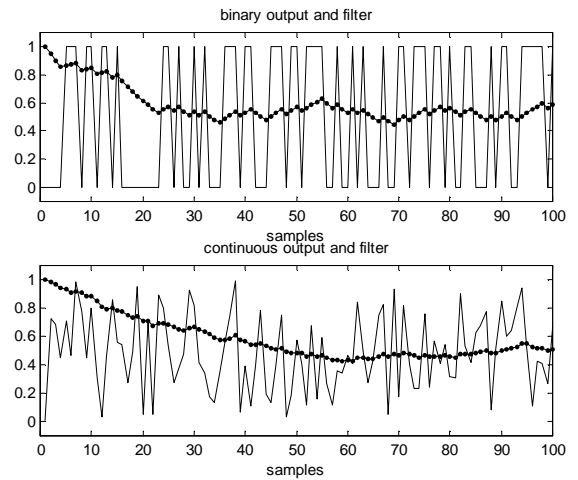


Figure 4 : Filtered values of binary and continuous output

2.1.1.2 Decision fading

So far we have discussed how to reduce variations in decisions during decision agreement and how the smoothing is automatically turned off when the decisions disagree. However, we still need a mechanism to deal with the disagreement where one tool makes a decision d_1 about a system state at time t_1 and another tool comes to a different conclusion d_2 at a later time t_2 . It is then necessary to account for the fact that d_2 may have occurred between t_1 and t_2 . We postulate that the later decision needs to be given more weight in case of decision disagreement to account for the possibility of occurrence

of event d_2 . The question is how much more weight d_2 (or how much less weight d_1) should have. We further propose that the discounting is a function of time passed between d_1 at time t_1 and d_2 at time t_2 . The reason is that there is a higher possibility for event d_2 to occur when the period t_2-t_1 is larger and vice versa. In addition, the tools must have information about their a priori performance. Again, we assume that the decisions are scaled between 0 and 1 [12] and propose to change the forgetting factor as the confidence value increases. The idea of decision fading is to discount information as it ages when tools disagree at different times (and no new update of both tools can be obtained). We force the older information to “fade” as a function of time passed. If the conditions outlined are met, the decision will be subjected to a fading exponent. The fading exponent must be tuned to the system at hand. The fading factor is close to 1 for small tool confidence and rises to 1.1 as tool confidence increases. The particular slope and upper saturation is system dependent. The governing equation is

$$new_decision = old_decision^{fading_exponent} \quad (5)$$

Figure 5 shows how the information of faults at two different confidence levels ($confidence_{upper}(1)=0.8$ and $confidence_{lower}(1)=0.2$) is discounted over time.

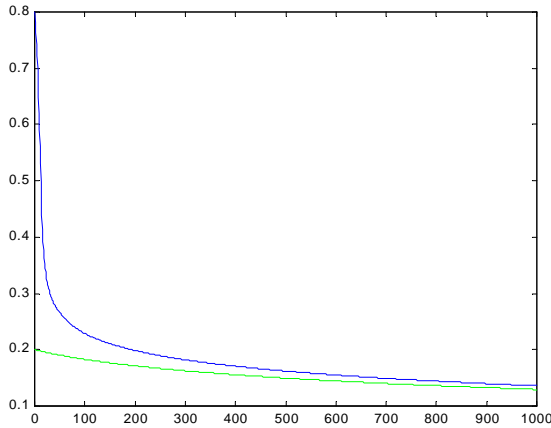


Figure 5 : Decision fading over time

Decision fading is another operation helpful in converting binary and continuous output by further performing operations on the previously binary output in the continuous domain.

2.1.1.3 Alertness region

The underlying idea of the alertness region is to temporarily raise the attentiveness to the possibility that an alarm situation might be currently present. This helps counterbalance the various damping mechanisms that are in place to avoid false positive alarms. Cues to this situation are primarily currently issued alarms and the

alarm ratio. The alarm ratio is a measure of frequency of alarms occurring in the immediate past. This ratio is computed as the number of alarms within a fixed window divided by the window size. If the alarm ratio passes a certain threshold, the alarm is issued. Let $alarm_damper$ be a tunable parameter that quantifies the number of readings that the system should wait before reporting an alarm, $alarm_window$ be a tunable parameter that is used to look at the most recent output, raw_alarm be the alarm class as output by the classification tool, then the $refined_alarm$ is :

$$refined_alarm = \begin{cases} 1 & \text{if } \frac{\sum_i^{alarm_window} raw_alarm > \frac{alarm_damper}{alarm_window}}{alarm_window} \wedge raw_alarm = 1 \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

2.1.2 Scaling (Layer 1)

The scaling layer implements functionality allowing a tool that is more reliable for a particular fault to be weighted more heavily than a tool with a worse track record for that same fault [13]. Clearly, there should be higher trust in a more reliable tool, even when a less reliable tool indicates a strong confidence in its own opinion. However, even the more reliable tool fails occasionally and when the less reliable tool happens to be correct, it would be a mistake to always use the more reliable tool’s classification output exclusively. Then again, it is not known when a particular tool is correct and when it is not. Therefore, we propose to use each tool’s information scaled by the degree of its reliability. This reliability is encoded as a priori information in the confusion matrix. More specifically, it is the diagonal entries of the confusion matrix which we will use to weigh each classification output. In cases where there are considerably less classifiers for a class than for other classes and the reliabilities for this class are small, there is a chance that the output is biased to the other classes. Therefore, we boost the former cases to allow a more balanced comparison. The test for that scenario is whether the sum of the reliabilities for a class (given they are relevant) is less than a threshold multiplied by the relevances. Relevance is defined as the ability of a classifier to provide output for particular class. The boosting operation scales the pertinent reliability values to the power of the sum of the reliabilities for that fault normed by the relevances. Let C be the confusion matrix of a classifier, raw_output the raw output, $relevance$ the binary information granule whether a classifier is able to give output for a particular class (i.e., $relevance=1$ if the classifier can give output, 0 otherwise), then the governing equation for scaling is:

$$scaled_output = raw_output \cdot \left(\prod_{classes} C(i,i) \cdot \frac{trace(C)}{\sum_{classes} relevance} \right)^{scaling_exponent} \quad (7)$$

Scaling is an operation that helps to further modify the previously binary output in the continuous domain.

2.2 Major Components: Analysis

This component performs the actual fusion. The layers contain both generic fusion strategies as well as domain knowledge heuristics.

2.2.1 Strengthening (Layer 2)

There are 3 modules in this layer. They are multiple-tool rewarding, multiple-parameter rewarding, and different operating regime rewarding.

2.2.1.1 Multiple-tool rewarding

Class opinions expressed by different tools which all agree should lead to a more confident assessment of the system state. This is the trivial case where coinciding opinions are rewarded. The rewarding scheme is accomplished by calculating the fused value as the sum of the classifier outputs that are in agreement [14]. Recall that the classification is a value expressing a likelihood for that particular class. That is, it is typically a weighted value. The strengthening equation $s: C' \rightarrow [0,1]$ aggregates the scaled classifier outputs that are greater than threshold t_s ,

$c'_i \geq t_s$, i.e., $c'_i = \begin{cases} c_i & \text{if } c_i \geq t_s \\ c_i & \text{otherwise} \end{cases}$ produces the fused value c_f .

$$c_{f_i} = \sum_{j=1}^t c'_{i_j} \quad (8)$$

2.2.1.2 Multiple-parameter rewarding

From first principles, we know that there exist certain correlations between some system parameters. For example, if the core speed goes up in a gas turbine, the exhaust gas temperature would most likely increase as well. So, the abnormal condition detection class occurs for the two correlated parameters at the same time. Therefore, the confidence for accepting the alarm should be high. Let *strengthening_exponent* be a tunable parameter > 1 . Then the governing equation is:

$$fused_value = fused_value^{strengthening_exponent} \quad (9)$$

2.2.1.3 Different operating regime rewarding

If an alarm class occurs during steady state phase and the same alarm occurred at a transient state before (such as start-up), the likelihood of a true alarm condition should be increased. The amount of confidence increase is directly related to the ratio of number of start-up alarm classes to the total number of system start-ups. The governing equation for each class and for two successive system runs is:

$$fused_value = fused_value \cdot \frac{\sum alarm}{\sum start_ups} \quad (10)$$

2.2.2 Weakening (Layer 3)

The weakening layer integrates the alarm information from similar equipment operating in tandem with the equipment under investigation. The hypothesis is that it is most unlikely that the same fault occurs on both systems at the same time. So, if the two systems give the same alarm class at the same time, most likely, the alarm is not fault related. Rather, it is due to a common environmental condition. Hence, the alarm should be discounted. The governing equation is:

$$fused_value = fused_value \cdot \left(1 - \frac{fused_value_{other_engine}}{fused_value} \right) \quad (11)$$

2.3 Major Components: Post Processing

Layers in this component deal with suppression and exception handling. Suppression only changes the number of alarms that are reported while exception handling lifts the restrictions imposed during suppression when certain conditions have changed that merit a new alarm to be issued.

2.3.1 Suppression (Layer 4)

One issue with alarms is that a large number of persistent alarms are expected which report on the same alarm condition – potentially many hundred times. This is in part due to the lag induced by the time it takes a maintenance team to address reported alarms. It is in part also due to a desired waiting period for low priority alarms which have a propensity for self-correction either due to alarm thresholds set too low, regularly scheduled maintenance or operation with different operating settings. It is clearly not a desired to repeatedly alarm on low priority and low impact alarms. An alarm handling mechanism is proposed which reports on “fresh” alarms, i.e., alarms that appear to be new, and it also reports on repetition alarms, i.e., alarms that have been observed for a while and that are persisting. These repetition alarms would be issued only at user specified intervals and would serve as a reminder that the alarming condition still exists. At the same time, the “fresh” alarm would no longer be issued. When the alarming condition ceases, the capability to issue fresh alarms is automatically re-activated. In addition, a damping is imposed on the fresh alarms which requires a set number of repeated alarms to occur persistently within a given time interval before an initial alarm report is issued.

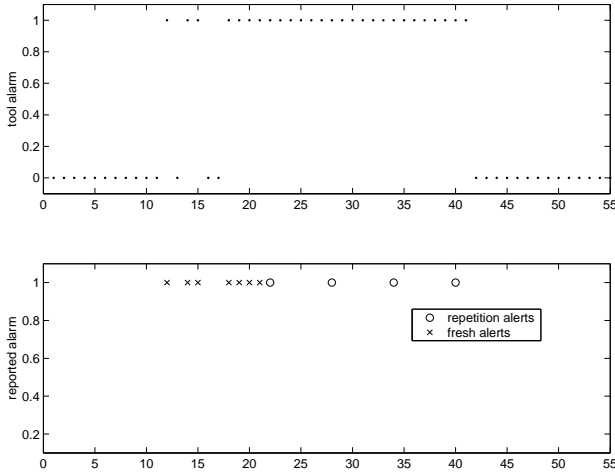


Figure 6: (a) unprocessed alarms, (b) reported fresh and repetition alarms

To illustrate, consider the situation as depicted in Figure 6. Figure 6(a) shows a number of alarms which apparently re-occur many times. Figure 6(b) shows both the fresh and repetition alarms in “x” and “o”, respectively. As mentioned before, both fresh reported alarm rate as well as period between reported repetition alarms are user adjustable.

2.3.2 Exception Handling (Layer 5)

The exception handling deals with situations where the restrictions imposed during the suppression are lifted because the alarm conditions themselves have changed. In particular, if the magnitude of the alarm causing condition changes, a new alarm may be issued to alert about this fact. This is necessary to avoid that worsening conditions go unnoticed.

3 Results

Components and layers of the fusion algorithm were tested via extensive Monte Carlo simulation and against real cases obtained from a variety of different equipment types. Performance was compared to a baseline approach that achieved a performance of 92.65% as measured by the index

$$performance_index = 0.6 \cdot (1 - FP) + 0.3 \cdot (1 - FN) + 0.1 \cdot (1 - FC) \quad (12)$$

The overall fusion performance index was $99.71\% \pm 0.03\%$ with 95% confidence. A performance gain was established as the percent improvement from the baseline performance. The overall fused system performance gain was $96.14\% \pm 0.36\%$ with 95% confidence. Contributions of some of the individual layers are compiled in Table 1.

Table 1 : Individual layer performance gain

Layer	Performance gain
Concurrency	10%
Scaling	6%
Strengthening	39%
Weakening	39%

One system under investigation was an aircraft equipment abnormal condition monitoring environment which includes both continuous as well as binary output. The latter were obtained from a small commercial airline operating Boeing 737 and Airbus 320 aircraft between 1996 and 1999. Among the observed 12 fault test cases are 4 bleed valve faults, 7 vibration faults, and 1 oil fault. Typical test result plots are shown in Figure 7 where alert tools 1 and 2 give continuous output and alert tool 3 gives binary output. The first subplot shows the true alarm status. The second through fifth subplots show the alarm status for the individual parameters – delta exhaust gas temperature (dEGT), fuel flow (WF), core speed (N2), oil temperature (OIL), and fan vibration (VIB) – as well as the fused and suppressed output. The second subplot also shows the result of the suppression. The results indicate the desired response: rejection of false positives, suppression of repetition alarms and responsiveness to the true alarm condition.

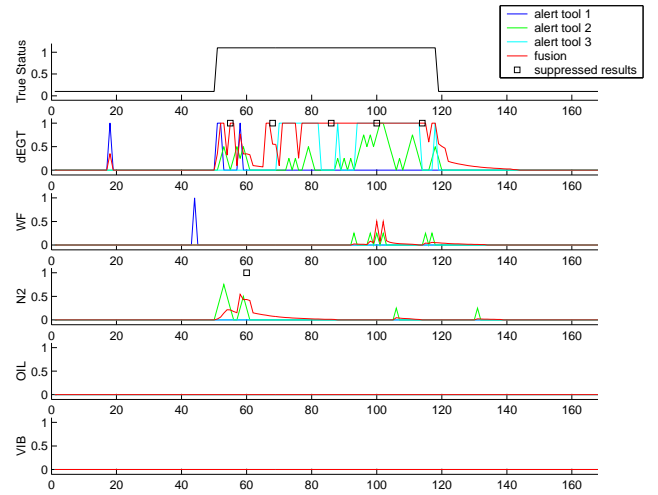


Figure 7: Fusion Results

4 Conclusions and Summary

This paper describes a fusion architecture and implementation for classifiers with binary as well as continuous output. The fusion scheme is represented as a multi-layered architecture which structures the approach into pre-processing, analysis, and post-processing. All components are partitioned into layers where each layer performs logical tasks and contributes to the performance improvement. Within the temporal layers we exploit

information about recent classifier output through a series of smoothing, fading, and alertness operations. The scaling layer capitalizes from the knowledge about classifier performance by scaling the classifier output accordingly. The strengthening layer performs the initial fusion step and employs several rewarding schemes based on similar classifier output found across tools and parameters observed as well as classes observed in different system operation regions. Conversely, the weakening layer discounts the fused output if similar output (other than “normal operation”) is found across different systems. The post-processing layers deal with suppression of repeated class notification as well as exception to that rule when significant changes have been observed within the same class. Operations in the preprocessing layers allow binary classifier output as well as continuous output to be merged into a single continuous domain without loss of information. It is both necessary and desirable to perform the fusion operations in the continuous domain because many concepts introduced here rely on the ability to express gradations of rewarding and penalizing. In addition, the operators used also need to be used in the continuous domain.

Because the fusion architecture is designed in a modular fashion, additional modules can be relatively easily be added or removed. It also allows re-use of the core fusion engine for other domains. In fact, various concepts and components of this algorithm were used for different applications such as on-board gas path diagnostics for aircraft engines [10], medical equipment parts ordering decision support, among others. We show results of this architecture applied to a system monitoring environment of industrial equipment.

References

- [1] D. Hall, A. Garga. *Pitfalls in Data Fusion (and How to Avoid Them)*. Proc. Second Int. Conf. Information Fusion (Fusion '99), pp. 429-436, 1999.
- [2] P. Bonissone, Y. Chen, K. Goebel, and P. Khedkar, *Hybrid Soft Computing Systems: Industrial and Commercial Applications*, Proceedings of the IEEE, Special Issue on Computational Intelligence, Vol. 87, No. 9, pp. 1641-1667, 1999.
- [3] K. Goebel, W. Yan, and W. Cheetham, *A Method to Calculate Classifier Correlation for Decision Fusion*, Proceedings of IDC 2002, Adelaide, Australia. pp. 135-140, 2002.
- [4] Y. Freund and R. Schapire, “A Short Introduction to Boosting”, *J. Japanese Society Artificial Intelligence*, **Vol. 14**, No.5, pp. 771-780. 1999.
- [5] P. Smets, “What is Dempster-Shafer’s model?” *Advances in the Dempster-Shafer Theory of Evidence*, Yager, R., Fedrizzi, M., and Kacprzyk, J., (Eds.), John Wiley & Sons, New York, pp. 5-34, 1994.
- [6] M. Nelson and K. Mason, “A Model-Based Approach to Information Fusion”. *Proc. Information, Decision, and Control*, pp. 395-400, 1999.
- [7] A. Loskiewicz-Buczak, R. Uhrig, “Decision Fusion by Fuzzy Set Operations”, *Proc. third IEEE Conf. Fuzzy Systems*, Vol. 2, pp.1412-1417, 1994.
- [8] N. S. V. Rao, Finite sample performance guarantees of fusers for function estimators, *Information Fusion*, **Vol. 1**, no. 1, pp. 35-44, 2000.
- [9] A. Rahman and M. Fairhurst, “Towards a Theoretical Framework for Multi-Layer Decision Fusion”, *Proc. IEE Third Europ. Workshop on Handwriting Analysis and Recognition*, pp. 7/1-7/7, 1998.
- [10] K. Goebel, *Architecture and Design of a Diagnostic Information Fusion Tool*, AIEDAM: special edition AI in Equipment Service, vol. 15 no. 4, pp. 335-348, 2001.
- [11] Khedkar, P., and Keshav, S., “Fuzzy Prediction of Time Series”, *Proceedings of the IEEE International Conference on Fuzzy Systems*, San Diego, 1992.
- [12] K. Goebel, “Decision Forgetting and Decision Smoothing for Diagnostic Information Fusion in Systems with Redundant Information”, *Proc. SPIE: Sensor Fusion: Architectures, Algorithms, and Applications IV*, Vol. 4051, pp. 438-445, 2000.
- [13] K. Goebel and S. Mysore, “Taking Advantage of Misclassifications to Boost Classification Rate in Decision Fusion”, *Proc. SPIE: Sensor Fusion: Architectures, Algorithms, and Applications V*, pp. 11-20, 2001.
- [14] K. Goebel, “Conflict Resolution using Strengthening and Weakening Operations in Decision Fusion”, *Proc. 4th Annual Conf. Information Fusion, Fusion 2001*, pp. ThA1-19 - ThA1-25, 2001.