A Boosting Method for Process Fault Detection with Detection Delay Reduction and Label Denoising

Mihajlo Grbovic, Slobodan Vucetic
Department of Computer and Information Sciences, Temple University
Philadelphia, PA 19122, USA
{mihajlo.grbovic, slobodan.vucetic}@temple.edu

Weichang Li, Peng Xu, Adam K. Usadi
Corporate Strategic Research, ExxonMobil Research and Engineering Company,
Annandale, NJ 08801, US
{weichang.li, peng.xu, adam.k.usadi}@exxonmobil.com

ABSTRACT

In this paper we propose a novel fault detection algorithm for process control and maintenance that builds an ensemble of classifiers based on the modified AdaBoost technique. While seeking for the best fault detection accuracy, our algorithm also concentrates on reducing detection delay, which ensures safety and timely equipment service. In addition, the new algorithm can simultaneously detect and remove class-label noise in process data. Training is performed via iteratively optimizing an exponential cost function. The cost function also adaptively changes at each iteration, such that (1) the importance of the fault transition periods is increased to reduce the detection delay and (2) noisy samples are removed from training data. The algorithm was tested on a well known benchmark problem, the Tennessee Eastman Process (TEP), and compared to the baseline AdaBoost ensemble fault detector that does not pay specific attention to minimization of the detection delay and noise removal.

Categories and Subject Descriptors

H.2.8 [Database applications]: Database Applications Data Mining

General Terms

Algorithms, Performance, Experimentation

Keywords

Fault Detection; Ensemble Learning; Boosting; Optimization

1. INTRODUCTION

Control and maintenance of complex manufacturing systems is an essential task in order to support quality control and to ensure safety. Timely detection of abnormal events and service requirement symptoms is critical to effective and safe operation of plant equipment. In general, fault detection algorithms can be categorized into unsupervised and supervised ones. In unsupervised algorithms such as principal component analysis [1] and independent component analysis [4], normal operation is modeled and faults are detected as deviations from the normal behavior. In supervised algorithms such as support vector machines and neural networks, a classifier is trained on historical data containing both normal and faulty conditions. In this paper we will concentrate on supervised methods that treat fault detection as a binary classification problem.

The desirable characteristics of a fault detection algorithm include low detection delay time, low false positive rate, and high detection probability. Early detection provides invaluable warning about emerging problems to avoid catastrophic consequences. Low false positive rate ensures the usability of the detection system. High detection accuracy is an essential requirement for successful detection and tracking of fault events. These performance metrics define an overall quality of any fault detection system, although tradeoff between detection speed and accuracy highly depends on specific applications.

Supervised classification algorithms in general minimize the overall classification error only, without explicitly considering detection delay. However, error minimization does not necessarily ensure small detection delay. As the transition periods between normal and faulty conditions are typically covered by a relatively small fraction of training data, their error contribution could be neglected at the expense of achieving high accuracy during the steady-state periods of normal or faulty behavior. Simultaneous minimization of detection error and detection delay is an important open problem to be addressed in this paper.

Another issue common to supervised fault detection is related to the quality of historical data annotation and the detectability of various types of faults. In reality, labeling of process data can often be inaccurate. In addition, certain types of faults might be hard to detect or even completely unobservable by the installed sensors. Similar could be said of the transition period between normal and faulty condition, which could be undetectable at first, due to slow fault propagation through the system. Labeling undetectable faults and such transition periods as faults is similar to mislabeling and can have an adverse effect on the accuracy.

In this paper, we propose a learning algorithm based on AdaBoost which simultaneously minimizes detection delay, maximizes accuracy, and is robust to inaccurate annotations. The idea is to pose the fault detection problem as minimizing an adaptive loss function which penalizes misclassification such that the contribution from samples occurring during fault transition periods is emphasized. It also removes samples estimated to be noisy or undetectable. The model obtained as a result of minimizing of such a loss function favors low detection delay and is also robust to noisy labeling. Our experimental results show the propose algorithm reduces detection delay while retaining high accuracy.

KDD4Service'11, San Diego, California, USA.

2. PRELIMINARIES

2.1 Problem Setup

We consider a plant monitored by a network of K sensors providing measurements synchronously in regular time intervals. At time i, the measurement at the k-th sensor is represented by a row vector variable $\mathbf{x}_i^k$ of length d_k. Combining all K sensors forms a single row vector $\mathbf{x}_i = [\mathbf{x}_i^1, \mathbf{x}_i^2, \ldots, \mathbf{x}_i^K]$ whose length is $d = \sum_{k=1}^{K} d_k$. The true plant state at time i is denoted as $y_i \in \{-1, +1\}$. Here –1 represents normal process condition; +1 presents a faulty state. We assume N samples of historical data are collected in a data set $D = \{(\mathbf{x}_i, y_i),\ i = 1 \ldots N\}$ for developing the fault detection model.

The classification of the system state at time i is denoted by $\hat{y}_i$. The goal is to build a single classification function $h: \mathbf{x}_i \rightarrow y_i$ from the data set D that accurately and timely detects faults in real time.

2.2 Fault Detection Performance Measures

Several metrics are typically used to evaluate the performance of fault detection algorithms. Assume that a labeled sequence of observations D_{test} is available for performance evaluation which consists of both normal and faulty conditions and is disjoint from the training data set. Assume D_{test} contains J fault occurrences.

The *true positive rate* (TPR) is defined as

$$TPR = (n_1 / N_1) \cdot 100, \quad (1)$$

where n_1 is the number of correctly detected faulty samples and N_1 is the total number of faulty samples in D_{test}.

The *false positive rate* (FPR) is given by,

$$FPR = (n_0 / N_0) \cdot 100, \quad (2)$$

where n_0 is the number of misclassified normal condition samples and N_0 is the total number of normal condition samples in D_{test}.

The *detection delay* DD_j for the j-th faulty occurrence from D_{test} is defined as the delay between t_j^0, the introduction time of the fault and its detection time t_j^1, i.e.,

$$DD_j = t_j^1 - t_j^0. \quad (3)$$

The average detection delay DD for all fault occurrences in D_{test} is

$$DD = \sum_{j=1}^{J} DD_j / J. \quad (4)$$

In Figure 1 a subset of the training data set D_{test} is shown, where the flat line is the true label and the dotted line is outcome from the fault detection model. It illustrates the detection delay region as well as examples of false positive and false negative predictions.

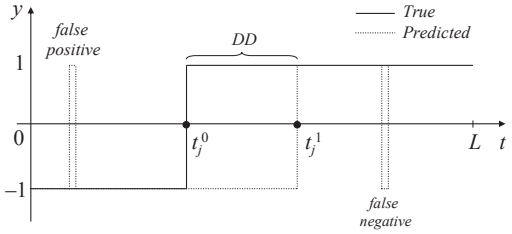

Figure 1. Illustration of detector performance during a fault

2.3 The AdaBoost Algorithm

The AdaBoost algorithm is formulated in [6] as an ensemble of base classifiers trained in a sequence using weighted versions of the training data set. At each iteration, it increases the weights of examples which were misclassified by the previously trained base classifiers. Final classifier is defined as a linear combination of all base classifiers. While AdaBoost has been developed using arguments from the statistical learning theory, it has been shown [7] that it can be interpreted as fitting an additive model through an iterative optimization of an exponential cost function.

Given: Data set $D = \{(x_i, y_i),\ i = 1 \ldots N\}$, initial data weight coefficients $w_i^0 = 1/N$, number of iterations M

FOR $m = 0$ **TO** M–1

(a) Fit a classifier $f_{m+1}(x)$ to training data by minimizing

$$J_{m+1} = \sum_{i=1}^{N} w_i^m I(y_i \neq f_{m+1}(x_i)) \quad (5)$$

(b) Evaluate the quantities:

$$\varepsilon_{m+1} = \sum_{i=1}^{N} w_i^m I(y_i \neq f_{m+1}(x_i)) / \sum_{i=1}^{N} w_i^m \quad (6)$$

and then use these to evaluate

$$\alpha_{m+1} = \ln\left(\frac{1-\varepsilon_{m+1}}{\varepsilon_{m+1}}\right) \quad (7)$$

(c) Update the example weights

$$w_i^{m+1} = w_i^m e^{\alpha_{m+1} I(y_i \neq F_{m+1}(x_i))} \quad (8)$$

END

Make predictions for new point x_{test} using:

$$\hat{y} = sign\left(\sum_{m=1}^{M} \alpha_m f_m(x_{test})\right) \quad (9)$$

Figure 2. AdaBoost algorithm

For a two-class classification setup, let us consider the exponential cost function defined as

$$E_m = \sum_{i=1}^{N} e^{-y_i \cdot F_m(x_i)}, \quad (10)$$

where $F_m(x)$ is the current additive model defined as a linear combination of m base classifiers produced so far,

$$F_m(x) = \sum_{j=1}^{m} \alpha_j f_j(x), \quad (11)$$

the base classifier $f_j(x)$ can be any classification model with output values +1 or –1 and α_j are constant multipliers called *the confidence parameters*.

Given the additive model $F_m(x)$ at iteration $m-1$ the objective is to find an improved one, $F_{m+1}(x) = F_m(x) + \alpha_{m+1} f_{m+1}(x)$, at iteration m. The cost function can be expressed as

$$E_{m+1} = \sum_{i=1}^{N} e^{-y_i (F_m(x_i) + \alpha_{m+1} f_{m+1}(x_i))} = \sum_{i=1}^{N} w_i^m e^{-y_i \alpha_{m+1} f_{m+1}(x_i)}, \quad (12)$$

where

$$w_i^m = e^{-y_i \cdot F_m(x_i)}, \quad (13)$$

are called *the example weights*. By rearranging E_{m+1}, we can obtain an expression that leads to the familiar AdaBoost algorithm,

$$E_{m+1} = (e^{\alpha_{m+1}} - e^{-\alpha_{m+1}}) \sum_{i=1}^{N} w_i^m I(y_i \neq f_{m+1}(x_i)) + e^{-\alpha_{m+1}} \sum_{i=1}^{N} w_i^m. \quad (14)$$

where $I(y_i \neq f_{m+1}(x_i))$ is an indicator function which equals 1 if i-th example is misclassified by f_{m+1} and 0 otherwise. For fixed α_{m+1}, classifier $f_{m+1}(x)$ can be trained by minimizing (14). Since α_{m+1} is fixed, the second term is constant and the multiplication factor in front of the sum in the first term does not affect the location of minimum, the base classifier can be found as $f_{m+1}(x) = \arg\min_{f(x)} J_{m+1}$, where J_{m+1} is defined as the weighted error function (5). Once the training of the new base classifier $f_{m+1}(x)$ is finished, α_{m+1} can be determined by minimizing (14)

assuming $f_{m+1}(x)$ is fixed. By setting $\partial E_{m+1}/\partial\alpha_{m+1}=0$ the closed form solution can be derived as (7), where ε_{m+1} is defined as in (6). After we obtain $f_{m+1}(x)$ and α_{m+1}, before continuing to round $m+1$ of the boosting procedure and training of f_{m+2}, the example weights w_i^m have to be updated. By making use of (13), weights for the next iteration can be calculated as (8). Thus, weight w_i^{m+1} depends on the performance of all previous base classifiers on i-th example. The procedure of training an additive model by stage-wise optimization of the exponential function is executed in iterations, each time adding a new base classifier. The resulting learning algorithm is identical to the familiar AdaBoost algorithm summarized in Figure 2.

In the next section, we introduce a new algorithm for joint detection delay and classification error minimization, which is also able to detect class-label noise. The algorithm naturally extends AdaBoost by changing the cost function at each iteration based on detection delay performance during the previous iteration

3. DEM-DEN BOOST

(De)lay (m)inimization and (De)-(n)oising Boosting algorithm is derived by introducing an additional multiplier into the original AdaBoost cost function,

$$E_m = \sum_{i\in D} g_i e^{-y_i\cdot F_m(x_i)}. \quad (15)$$

Function g_i allows us to increase the penalty of wrong predictions for each sample separately. A default value of $g_i=1$, for all $i=1\ldots N$ corresponds to the original AdaBoost cost function. Minimizing the detection delay is achieved by increasing the penalty of wrong predictions (achieved by assigning high g_i value) for examples within the fault transition region. Our algorithm modifies g_i values after each iteration, which creates a need to optimize slightly different cost function E_m after each iteration. AdaBoost is very appropriate for this setup, because we can utilize the existing ensemble when constructing a new fault detection classifier on the modified cost function.

Let us assume that up to iteration m an AdaBoost committee with m base classifiers $F_m(x)$ has been trained by minimizing

$$E_m^{old} = \sum_{i\in D} g_i^{old} e^{-y_i\cdot F_m(x_i)}. \quad (16)$$

Upon the change of g in iteration m, the cost function changes to

$$E_m^{new} = \sum_{i\in D} g_i^{new} e^{-y_i\cdot F_m(x_i)}. \quad (17)$$

Before proceeding to finding a new base classifier f_{m+1} and its confidence parameter α_{m+1} there are two changes one should make to adapt to the cost function change:

1. Confidence Parameter Update. The first task involves updating confidence parameters α_k, $k=1\ldots m$, in such way that they now minimize (17) for fixed $f_k, k=1\ldots m$. This can be achieved by updating the existing α_k, $k=1\ldots m$, using the gradient descent algorithm $\alpha_k^{new}=\alpha_k^{old}-\eta\cdot\partial E_m^{new}/\partial\alpha_k^{old}$, where η is the learning rate. Following this, the confidence parameters for all m base classifiers can be updated as

$$\alpha_k^{new} = \alpha_k^{old} + \eta\sum_{i\in D} g_i^{new} y_i f_k(x_i) e^{-y_i\sum_{j=1}^{m}\alpha_j^{old} f_j(x_i)}. \quad (18)$$

2. Example Weight Update. The second task involves calculating the example weights for training of f_{m+1}. With addition of the new base classifier, $F_{m+1}(x)=F_m(x)+\alpha_{m+1}\cdot f_{m+1}(x)$, the resulting ensemble's cost function can be expressed as

$$E_{m+1} = \sum_{i=1}^{N} g_i^{new} e^{-y_i(F_m(x_i)+\alpha_{m+1}f_{m+1}(x_i))} = \sum_{i=1}^{N} w_i^m e^{-y_i\alpha_{m+1}f_{m+1}(x_i)}, \quad (19)$$

where

$$w_i^m = g_i^{new} e^{-y_i\cdot F_m(x_i)} \quad (20)$$

represent the *example weights* for training f_{m+1}. Training a new base classifier f_{m+1} is conducted in the same manner as in the regular AdaBoost by minimizing the new cost function in a way that optimally utilizes the existing boosting ensemble. After recalculating the example weights using (20), minimizing (18) with respect to f_{m+1} and α_{m+1} leads to familiar equations (5) and (7)

Adapting $g(\cdot)$. Minimization of detection delay is carried out using the multiplier function g. As explained in 2.2, data D consists of J fault occurrences. In each sequence $\{(x_t, y_t), t=1\ldots L\}_j$, $j=1\ldots J$, the fault is introduced at sample t_j^0 and is removed at sample L (Figure 1). We will define function g_i, $i=1\ldots N$ for each sequence separately.

Let us consider the j-th fault occurrence sequence from D and assume the ensemble F_m built up to m-th iteration detects the fault at sample t_j^1. We define $g(t, t_j^1)$ at iteration m for sequence j as

$$g^{new}(t,t_j^1) = \begin{cases} 1+\sigma\cdot e^{\frac{t_j^1-t}{\sigma}}, & t_j^0 < t < t_j^1, \\ 1, & otherwise \end{cases} \quad (21)$$

where σ is the detection delay punishment level parameter to be appropriately selected. Figure 3 illustrates how function $g(t, t_j^1)$ is formed for a j-th faulty sequence based on F_m predictions.

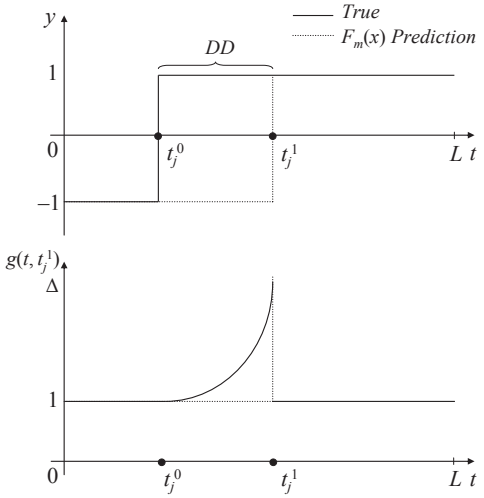

Figure 3. Function $g(t, t_j^1)$ for a j-th fault occurrence sequence

As we can observe function $g(t, t_j^1)$ additionally increases the weights of examples falling within the F_m detection delay region (high g_i value). Specifically, the examples right before the fault detection time t_j^1 by the current model F_m will receive the highest additional weight increase. The examples outside the region, including t_j^1, will not receive any additional weight increase (default g_i value). These actions ensure that the algorithm will attempt to further reduce the detection delay in the next iteration.

Removal of Class-Label Noise. To deal with mislabeled fault sequences, we propose a de-noising procedure similar to the method proposed in [5] where the authors identify outliers during AdaBoost realization as points whose weights become larger than a certain threshold T. In [5], the weights of the detected outliers are set to zero and the boosting procedure continues to the next iteration. However, the weights of the remaining examples are not

updated to reflect this change, which could lead to a deteriorated performance in further boosting iterations. Also, the cost function E_m changes with removal of examples. Such change requires updating of the confidence parameters, which was not considered.

In our implementation, the examples are removed when their weights exceed the threshold T, which is followed by confidence parameter update (18) and example weight recalculation (20). To include class-label noise removal procedure, we modified function g in such way that it accounts for examples whose weights extend over the threshold value T,

$$g^{new}(t,t_j^1,w)=\begin{cases}1+\sigma\cdot e^{\frac{t_j^1-t}{\sigma}}, t_j^0<t<t_j^1\\ 0,\ w/\,g^{old}(t,t_j^1,w)>T.\\ 1,\ otherwise\end{cases} \quad (22)$$

Setting the i-th example cost function multiplier g_i to zero has the same effect as setting its weight w_i to zero (20).

The summary of *Dem-Den Boost* algorithm is shown in Figure 4.

Given: Data set $D=\{(x_i,y_i), i=1\ldots N\}$, consisting of J fault occurrence sequences, initial data weight coefficients $w_i^0=1/N$,, number of iterations M, threshold T, penalty parameter σ

FOR $m=0$ **TO** M–1
(a) Fit a classifier $f_{m+1}(x)$ to training data by minimizing (5)
(b) Evaluate the quantities ε_{m+1} (6) and α_{m+1} (7)
(c) Find t_j^1 for each fault sequence using F_{m+1} fault detector
(d) Determine $g_i^{new}(t, t_j^1, w_i)$, $i=1\ldots N$ for each fault sequence (22)
(e) Update α_k , $k=1\ldots m$ using (18) (possibly until convergence)
(f) Calculate example weights for the next iteration using

$$w_i^{m+1}=g_i^{new}e^{-y_i\cdot F_{m+1}(x_i)},\ i\in D$$

END
Make Predictions using (9)

Figure 4. Dem-Den Boost Algorithm

4. EXPERIMENTS

4.1 Tennessee Eastman Process (TEP)

The proposed boosting method was evaluated on a well-known benchmark problem, the Tennessee Eastman Industrial Challenge Problem [2], which was created to provide a realistic simulator of an industrial process in order to evaluate process control methods.

A large number of fault detection approaches [1, 4] were tested on TEP. The process has 53 variables including 22 process, 19 analyzer and 12 manipulated variables. In our research, all 19 analyzer measurements are excluded as well as the manipulated variable representing agitator speed, which is constant in all simulation runs. For details about variables see [1].

TEP has 20 identified faults. They range from faults that are relatively easy to detect without any delay (e.g., fault 1 and 4) due to significant and clear deviation from normal conditions with fast transition, to faults that are relatively easy to detect but require a certain amount of time to propagate through the plant before they become detectable (e.g., fault 17 and 18), to faults that are extremely hard to detect due to their subtleness (faults 3, 9, 15).

4.2 Experimental Setup

The training data set $D_{tr}=\{(\mathbf{x}_i,y_i),\ i=1\ldots N,\ y_i\in\{-1,+1\}\}$ consists of sequences in which normal and faulty data interchange. Specifically, four separate data sequences, each of length $L=1{,}000$ samples, were generated for each of the 20 TEP faults, where the fault was introduced at time $t^0=501$. In this way, a total of $N=80{,}000$ samples were generated for the training data. For the purposes of evaluation, we generated test data set D_{test} consisting of ten sequences of length $L=2{,}000$ for each of the 20 faults, where the fault was introduced at the 1001st sample point.

The TEP data annotation simulates the human annotation by providing imperfect, noisy data labeling. The following situations in TEP data sequences can cause problems for supervised training.

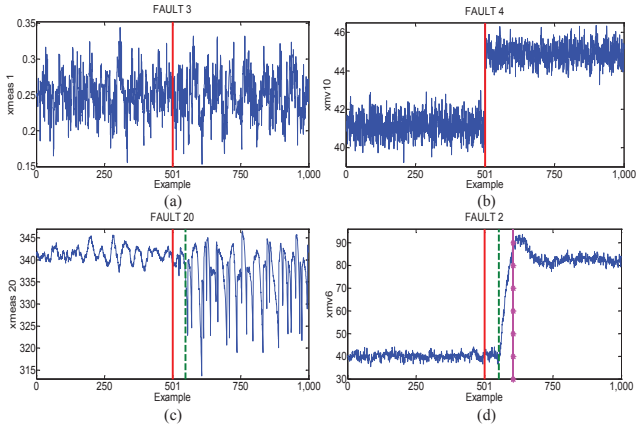

Figure 5. Faults introduced by TEP at sample 501 (flat line)

1. Subtle faults are faults of such small intensity that it is very hard, or impossible, to distinguish them from normal operating conditions. Labeling these conditions as faults and using them for training corresponds to introducing class label noise and will reflect in reduced overall accuracy. Figure 5-a shows TEP sensor *xmeas*1 sequence of length 1,000 in which fault 3 is introduced at 501th sample. As it can be observed, deviation from normal conditions is absent. This behavior is consistent in all sensors.

2. Propagation periods can occur when there is a delay between introduction of a fault by the TEP simulator and its visible effects in the process. Some faults have especially long propagation periods. Labeling these long propagation periods as faults can have similar adverse effects as the undetectable faults. In Figure 5, fault introduction and annotation by the simulator is marked with the flat line, while the actual occurrence of the fault is marked with the dotted line. The period between them is referred to as the propagation period. For faults 2 and 20 it can be observed. However, this is not the case with fault 4.

3. Transition periods are defined as a periods between the occurrence of fault and its settling to the faulty steady state. This period is usually not long and some faults can even have transition period of length 0 (Figure 5-b). Accuracy during the transition period is critical for achieving low detection delay. Figure 5-d shows fault 2 as observed by sensor *xmv*6. The period between the occurrence of the fault (dotted line) and the steady fault state start (marker) is referred to as the transition period.

Dem-Den Boosting parameters were selected as follows. Value of threshold T for sample removal was chosen using the same procedure used in [5]. We ran the algorithm for several values for T, and the best threshold value was chosen to be the one that gives the lowest classification error on the hold-out validation set (30% of D_{tr}). Choice of parameter σ is an important yet nontrivial task. Different fault dynamics may require different optimal σ values hence the choice of σ ideally should be data dependent. One solution is to develop adaptive method within a robustness constraint, such that the adaptive procedure does not over

emphasize short delays by sacrificing too much accuracy and also the maximum delay is also upper bounded by the minimum latency requirements. Developing such an adaptive scheme for choosing and updating σ is a part of our ongoing work. In this paper we use the following procedure for choosing σ. Several values of parameter σ were tested using the validation set such that the sum of $g(t, t_j^1)$ between t_j^0 and t_j^1 equals 5, 10, 50 or 100.

The CART decision trees were used as base classifiers. A total of $M = 20$ boosting iterations were performed, where the maximum number of decision tree splits was set to 30. For easier evaluation we compared the methods on a fixed FPR. To set the FPR value, the committee prediction was regularized with a threshold π

$$\hat{y} = sign(\sum_{m=1}^{M} \alpha_m f_m(x_{test}) - \pi) \quad (23)$$

4.3 Experimental Results

In Table 1 the *Dem-Den Boost* was compared to the baseline *AdaBoost* algorithm, as well as to a reduced version called *Den Boost* which only removes label noise without delay minimization. The comparison was carried out based on a test set D_{test} consisting of 400,000 points. The multi-criterion performance was evaluated for each fault at FPR = 1%. The TPR was calculated using 10,000 points for each fault, corresponding to a t-score of $t_{0.05}$=1.645. Symbol ∞ is used if a fault cannot be detected. Average values do not include fault 3, 9 and 15 results. Bottom column shows statistical significance of *Den* and *Dem-Den Boost* TPR and DD improvement over *AdaBoost* in terms of paired t-test *p*-values.

Table 1. Performance Comparison at FPR = 1%

	TPR (%)			DD		
fault	**Ada**	**Den**	**Dem-Den**	**Ada**	**Den**	**Dem-Den**
1	99.19	99.77	99.8	2.3	1.9	1.5
2	97.1	98.35	98.41	16.2	14.5	13
3	1.51	1.13	1.29	∞	∞	∞
4	97.69	99.91	99.82	0.2	0	0
5	87.07	99.27	98.98	2.4	0.2	0
6	96.97	98.14	98.99	3.4	1.5	0
7	97.92	99.75	99.46	0.3	0.2	0
8	94.02	96.32	96.96	20.2	21	17
9	1.83	1.35	1.3	∞	∞	∞
10	61.48	73.6	74.89	26.4	21	18.2
11	73.36	81.78	71.16	7.1	7.4	6
12	97.16	98.55	98.59	7.8	6.1	5.1
13	92.55	93.34	93.63	56.9	54.7	50.2
14	96.91	99.55	99.47	0.9	0.6	0
15	1.73	1.06	1.22	∞	∞	∞
16	63.75	77.02	77.38	18.5	15	13.6
17	91.3	93.51	93.64	23.9	23.1	19.3
18	81.24	82.63	83.45	54.9	51.4	47.3
19	43.66	70.36	71.21	10.8	3.9	2
20	60.26	73.68	74.21	45.7	44.2	39.2
avr	**84.2**	**90.32**	**90.00**	**17.52**	**15.76**	**13.7**
p-value	**-**	**0.0043**	**0.0082**	**-**	**0.002**	**0.0001**

We can conclude that the *Den Boost* outperforms the original *AdaBoost* in both TPR and DD by locating and removing the noisy examples. The biggest TPR improvement was achieved on faults 5, 10, 11, 16, 19 and 20. These results confirm that the class-label noise removal component of our algorithm is beneficial. The results also confirm that by emphasizing the importance of early detection with cost function modification, *Dem-Den Boost* improves detection delay by approximately 4 samples on average. The biggest improvement can be seen on faults 6, 10, 13, 19, 20. Notice that *Dem-Den Boost* also achieves better TPR than AdaBoost because of successful noise removal procedure that runs simultaneously with the detection delay minimization. Also, by conventional criteria (p-values), all the differences are considered to be very statistically significant.

To present the results at different FPR levels, the fault detection performance metrics were utilized to form two curves – the Activity Monitoring Operating Characteristic (AMOC) [3] and the Receiver Operating Characteristic (ROC) curve. By sliding the threshold π from higher to lower values, the model increases its FPR. Figure 6 shows the performance comparison averaged over 17 faults (faults 3, 9 and 15 are excluded), within the FPR range of 1-5%, which is the range of typical practical interest (high FPR would drastically reduce the usability of the system)

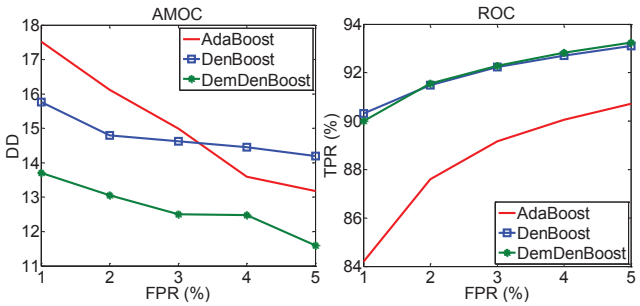

Figure 6. Performance Comparison

The results suggest that the *Dem-Den* algorithm has the overall best performance in the observed range. Additional numerical experiments comparing *Dem-Den Boost* with SVM is on-going. Preliminary results show that at the same FPR level our algorithm achieves better DD, and with denoising, improves TPR on most of the faults as well.

5. CONCLUSION

This paper presented a novel supervised fault detection algorithm capable of improving fault detection accuracy and reducing detection time over the baseline *AdaBoost* algorithm. The results based on the benchmark Tennessee Eastman data demonstrate that the new approach achieves a significant overall improvement in detection delay and accuracy over the baseline approach.

6. REFERENCES

[1] Chiang L. H., Russell E., and Braatz R. D. 2001. *Fault detection and diagnosis in industrial systems*, *Springer*

[2] Downs D.D., Vogel E.F. 1993. A plant-wide industrial process control problem, *Computers & Chemical Engineering*, 17, 3 (March. 1993) 245-255.

[3] Fawcett T., Provost F. 1999. Activity monitoring: noticing interesting changes in behavior, *ACM SIGKDD* 53-62.

[4] Leea J. M, Yoob C. K. and Leea I-B. 2004. Statistical monitoring of dynamic processes based on dynamic independent component analysis, *Chemical Engineering Science*, 59, 14 (July 2004) 2995-3006.

[5] Karmaker A., Kwek S. 2006. A boosting approach to remove class label noise, *International Journal of Hybrid Intelligent Systems*, 3, 3 (August 2006) 169-177.

[6] Freund Y., Schapire R. E. 1996, Experiments with a new boosting algorithm, *International Conference on Machine Learning*, (July 1996) 148-156.

[7] Friedman J., Hastie T., Tibshirani R. 2000, Additive logistic regression: a statistical view of boosting, *The Annals of Statistics*, 28, 2 (April 2000) 337-407.