Prognosis of Faults in Gas Turbine Engines[1][2]

Tom Brotherton
Intelligent Automation Corporation
10299 Scripps Trail, PMB 231
San Diego, CA 92131
858-586-6628
tbrotherton@bigfoot.com

Gary Jahns, Jerry Jacobs, and Dariusz Wroblewski
ORINCON Corporation
9363 Towne Centre Drive
San Diego, CA 92121

Abstract— A problem of interest to aircraft engine maintainers is the automatic detection, classification, and prediction (or prognosis) of potential critical component failures in gas turbine engines. Automatic monitoring offers the promise of substantially reducing the cost of repair and replacement of defective parts, and may even result in saving lives. Current processing for prognostic health monitoring (PHM) uses relatively simple metrics or features and rules to measure and characterize changes in sensor data. An alternative solution is to use neural nets coupled with appropriate feature extractors. Neural nets have been shown to give superior statistical performance when compared to more traditional techniques, and some net architectures are capable of "novelty detection" (identifying an event as dissimilar to any previously observed). On the negative side, neural nets used to model and predict engine status necessarily will be complicated and will contain many "black box" components that do not offer insight into the "why" the net came up with the results that it did. There is currently a wide variety of data mining (or rule extraction) tools commercially available. All of these tools come up with readily-understood rules by performing an exhaustive search on training data. However, in most cases the simple rule derived does not have the statistical performance realized with the neural models. Rule extraction tools are also incapable of novelty detection. We have developed techniques that couple neural nets with automated rule extractors to form systems that have:

- Good statistical performance
- Easy system explanation and validation
- Potential new data insights and new rule discovery
- Novelty detection
- Real-time performance

We apply these techniques to data sets data collected from operating engines. Prognostic examples using the integrated system will be shown and compared with current PHM system performance. Rules for performing the prognostics will be developed and the rule performance compared as well.

TABLE OF CONTENTS

1. INTRODUCTION

Real-time engine diagnostic monitoring systems deal with fault and off-nominal conditions as they are occurring. Prognostics is the ability to assess the current health of a part and predict into the future the health of a part for a fixed time horizon or predict the time to failure. Being able to perform reliable prognostics is the key to condition based maintenance (CBM). Prognostics are critical to the aircraft for improving safety, planning missions, scheduling maintenance, and reducing maintenance costs and down time.

Table 1. List of Acronyms

ACRONYM	MEANING
CBM	Condition-Based Maintenance
HNN	Hierarchical Neural Network
MLP	Multi-Layer Perceptron (neural network)
NN	Neural Network
PHM	Prognostic Health Monitoring
RBF	Radial Basis Function (neural network)
SNR	Signal-to-Noise Ratio

[1] 0-7803-5846-5/00/$10.00 © 2000 IEEE
[2] Updated Jan3, 2000

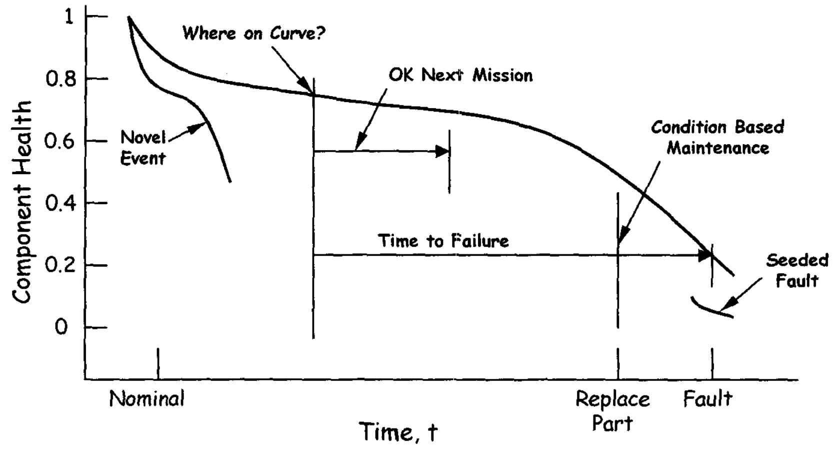


Figure 1 Machine Health Versus Time. Note that "seeded fault" data, while being different from "normal," may not be realistic, as it does not fall on a normal machine health trajectory.

There currently exist simple prognostics for component parts in the form of component *life monitors*. Life monitors are usually based on statistics gathered over a large population of components. Sometimes they do include physical models. However even these models are a measure of an average component's health and do not account for the history of the specific component being monitored. Component life monitors are coarse and conservative. And all are essentially based on measuring "time" in some fashion (for example "cycles"). For CBM problems, our goal is to develop algorithms that make decisions from current measurements of a component to perform prognostics. Lifing modles maybe a good starting point to describe the average behavior of a component. Advanced techniques are then applied to determine the differences from that average.

Figure 1 shows the different component health monitoring problems that need to be addressed. The figure shows the trajectory of a machine component's health as a function of time. When the component is new, its health is considered 100 percent. As time goes on and the component begins to wear out, its health, defined here somewhat arbitrarily, drops. Implied in the figure is that the component is following a known fault-life degradation path.

In the discussion following, an *anomaly* is any off-nominal operational condition. Anomalies come in two types. The first is a *fault*. A fault is a known off-nominal condition. It is assumed that fault-specific processing has been developed to detect a fault. The second anomaly is a *novel* event. A novel event is an unknown off-nominal condition. That is, the novel event is neither nominal nor any of the known fault conditions. It's something completely new. Prognostics necessarily will deal with faults (and not novel events). This is because an important part of the prognostics is the modeling for prediction of the component health trajectory shown in Figure 1. In order to develop that model, some knowledge about the trajectory of a component from nominal to a known fault condition is required.

Referring to Figure 1 the following aspects of the problem may be considered.

- Component health monitoring determines where the component is on the curve shown in Figure 1. Is the part "nominal", does some "anomaly" condition exist, or is it somewhere between those two extremes? Note that a normal component health curve may encompass a variety of behaviors and thus this curve represents a single region or a single fault trajectory rather than a series of strictly defined points. Determining where we are on the component health curve is the first step in prognostics.
- Fault detection/diagnostic reasoning as discussed above, determines if a component has moved away (degraded) from 100% along a known path as indicated in Figure 1, to a point where component performance may be compromised.
- Novelty detection determines if the component has moved away from what is considered acceptable nominal operations (an anomaly detection as defined above) and all known fault health (diagnostics as defined above) propagation paths.
- Prognosis is the assessment of the component's current health and a prediction of the component's future health. There are two variations of the prediction problem. The first prediction type may have just a short horizon time—is the component good to fly the next mission? The second type is to predict how much time we have before a particular fault will occur and, by extension, how much time we have before we should replace it. Or it may be longer term—tell me when to schedule removal of an engine for overhaul. As mentioned above, accurate prognosis is a requirement for implementing CBM.

2. THE PROGNOSTIC PROBLEM

Prognostics is a challenging problem. There are several areas that need be addressed in order to develop a prognostic reasoner that achieves a worthwhile level of statistical performance.

The first step in prognosis is determining "where" on the overall health curve the component resides. Along with "where" is "what" fault curve we are on. This is similar to the "fault detection" problem as already discussed above. However the equivalent signal-to-noise ration (SNR) of the signatures that we are looking for to determine component health will be much lower than for the fully developed fault. This will have two effects.

First, because the health component signatures' SNRs are low, we are always operating in the "gray" area between nominal and a fully developed fault. Because we are in the gray area, even knowing what fault trajectory we are operating on is a challenge. Likely several different fault hypotheses will need to be carried along by the system until a clear-cut condition becomes apparent. Likely a large number of the hypotheses are false so that ultimately no maintenance operation will be required.

Second, when we are on the "flat" part of the overall health curve of the component as shown in Figure 2, it is hard to resolve in time where we are on the curve. Again the problem can be attributed to operating in the gray region between nominal and a fully developed fault. Suppose that the best we can do in resolving the "health" of a component is to determine that it is in a range of 60-80% of perfect. The component is still quite acceptable. However as indicated in by the band in Figure 2, we can not resolve where we are on the curve. Predictions for short time horizons will be reliable (i.e., in determining "good-to-go" for the next mission decisions), but determining remaining life is not possible. The conservative approach would be to assume the worst; that we are at the end of the green part of the curve. Or to couple the prognostics with life usage models: The life usage model (assuming one exists) will form the basic estimate of the component health, and the prognostics prediction is just used to refine that basic result.

Once we determine what the current health of the component is, we need to predict what the health of the component will be sometime in the future. As discussed, this prediction can be for a short time horizon or an estimate of the time until the part needs to be replaced or a failure will occur. There are a variety of issues that need be considered.

The model will need to accurately predict into the future. Those predictions will be required to be unbiased and to have a small variance in order to be useful. Figure 3 illustrates these problems. In that figure the red line is the prediction of the health of the component from the current state. It does not follow the actual trajectory very well so that it is a biased estimate of the actual trajectory. However, the model does accurately predict the health/time to replace the component. Is this sufficient? The green lines represent the error bars for the prediction. The true value of the component health curve should fall inside of these error bars as is does. Thus the model is sufficient since is always includes "truth". How useful is the model at the given level of prediction variance? The spreading of the error bars define the time horizon and resolution that can be achieved with this model for performing prognostics. If the error bars spread rapidly then only the predictions are reliable for only a short time horizon. If they are narrow and follow the true trajectory accurately, then the information from the predictions is useful for longer time horizons.

3. APPROACHES TO PROGNOSTICS

There are essentially three different approaches for the development of prognostics reasoner techniques. The first are physical models. These are models that have been developed by experts in the component field and validated on large sets of data to show that they are indeed accurate. The second are systems that embody rules of thumb that have been developed and refined by human maintenance

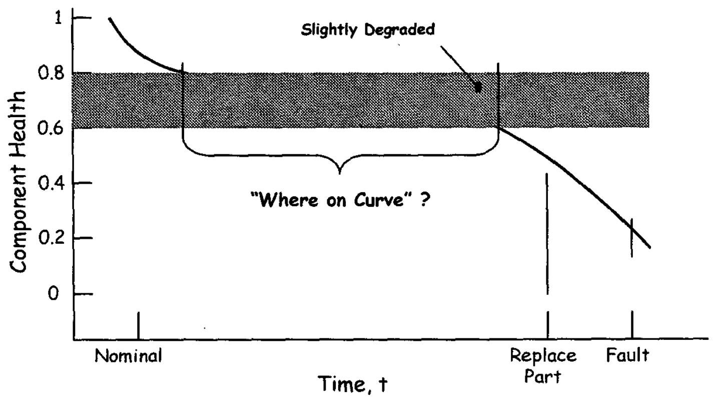


Figure 2 Prognostics: Step 1 - Determining the Current Health of the Component.

experts. Examples of these systems are rule-based expert systems and fuzzy logic systems. The third are statistical models that 'learn' from examination of real data that contain nominal and known fault conditions. Examples of these are neural networks and data mining systems.

Physical models and rule based systems have the good feature of containing information for anticipated fault events that have yet to occur on the component that is being monitoring. This is in contrast to 'learning' systems. Learning systems are only as good as the data from which they have been trained.

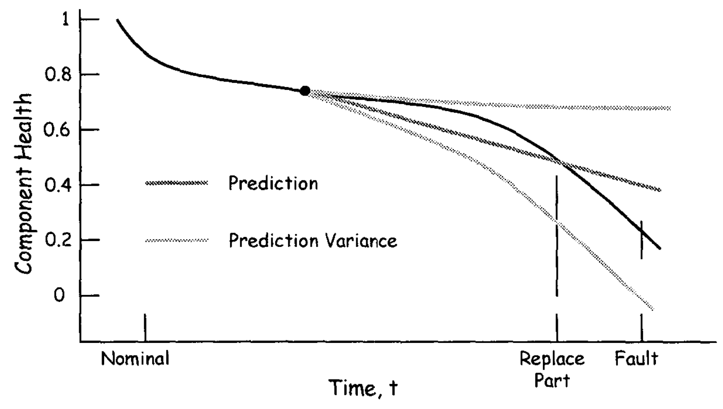


Figure 3 Predictions for Prognostics

On the other hand 'learning' systems have the good feature that they can process a wide variety of data types and potentially have performance superior to rule-based systems because they exploit the nuances in the data that are not covered by general rules. This is particularly true for new sources of data for which expert analysis, physical models, and rules have not been developed.

Artificial Intelligence/Rule-based expert systems/Inference Engines

By definition, *Artificial Intelligence* (AI) programs are intended to give the appearance of human behavior. AI programming languages supplied tools for programs to be built that closely resemble human logic and expertise in their implementation. These programs are called *expert systems.*

Rule-based expert systems are one of the most commonly used approaches for developing expert systems [1]. In this approach, rules are used to represent heuristics, or "rules of thumb," that specify a set of actions to be performed for a given situation. We usually think of the rule-base expert system as a list of "if" and "then" statements. The "if" portion of a rule is a series of patterns which specify the facts (or data) which cause the rule to be activated. The expert system tool provides a mechanism, called the *inference engine*, which automatically matches facts against patterns and determines which rules are applicable. The "if" portion of a rule can actually be thought of as the "whenever" portion of a rule since pattern matching always occurs whenever changes are made to facts. The "then" portion of a rule is the set of actions to be executed when the rule is applicable. The inference engine selects a rule and then the actions of the selected rule are executed (which may affect the list of applicable rules by adding or removing facts). The inference engine then selects another rule and executes its actions. This process continues until no applicable rules remain.

Expert systems have a major advantage over all the 'learning' systems in that real data is not required in order to develop the system. However, a knowledgeable expert is required. Also the rules tend to follow the structure of the rule development environment that is being used. As such possible non-linear and correlated relationships maybe lost.

Fuzzy logic

Fuzzy sets were introduced by Zadeh [2] in 1965. Fuzzy sets were first invented as a means of generalizing conventional set theory to model the realities of everyday life [3], including applications to industrial problems [4]. We've all heard the examples; the fuzzy set of operating temperatures of a machine can be {hot, cold} while the actual measurement is a 'crisp' number i.e., T=516.7 degrees. Based on T is the machine 'hot' or 'cold'? Fuzzy set theory would assign a membership to both 'hot' and 'cold'.

Fuzzy set theory has been extended to include fuzzy logic for prognostic reasoner applications. The critical step is the development of ways to aggregate fuzzy measurements to form fuzzy rules. Standard software packages are available for the development of fuzzy systems. The fuzzy membership functions can be developed 'by hand' using the developers intuition / expertise as the shape of the functions. Or the functions can be estimated from training data. There the developer specifies the function parametric shape (i.e. trapezoid or Gaussian) and the parameters are estimated from the data.

The advantage of fuzzy rules over those developed with an expert system is that the degree of membership is carried through out the computation and a 'hard' decision is only generated at the very last step.

Model-based Approaches

Model-based approaches utilize an explicit mathematical model for the machine being monitored [5]. The models can be physical models or statistical models [6,7]. Physical models are useful in accounting for all operating conditions.

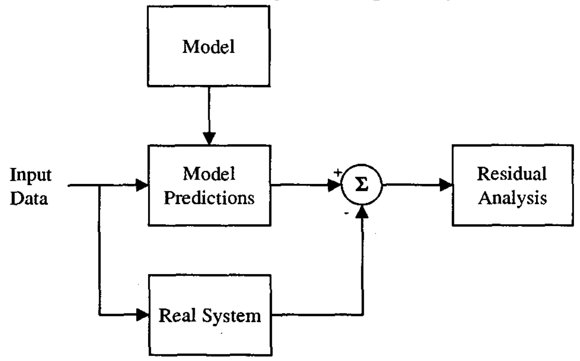


Figure 4 Model-Based Fault Detection/Analysis Example

Statistical models are developed from collected input/output data and as such may not account for conditions that have not been recorded and thus not included into the model. The statistical model may be linear/non-linear, stationary/time varying, single/multi-channel, simple/complex.

Component life models are included here. As mentioned above, these may be based on physics of the components being monitored. However for most applications these are based on statistics collected over a large set of samples and are used to define an average component.

Figure 4 shows an example of a simple model-based system for decision support. The 'model' in the figure has been developed off-line and is either physical or statistical as described above. Here the model is used to predict to output of the system being monitored. Computing the difference (the residuals) compares the model predictions and real system outputs. For nominal operation, the residuals are ideally a small variance, white-noise data stream. If that is not the case, then the residuals are an indication of an impending fault condition.

Model-based approaches are only as good as the models developed. For very simple systems, for example with a single input/output pair that are linearly related, likely the model developed is good. As inputs and complexity of the real system grows, the chances of the model being 'exact' diminish. For statistical models, data from all modes of operation and fault conditions must be collected. This is often not possible. Physical models potentially cover these holes. However, to validate a complicated physical model, data from all modes of operation and fault conditions must also be collected.

Neural networks

Artificial neural networks or 'neural nets' (NN) are an attempt to model the brain by the dense interconnection of a large set of simple processing elements. Details of neural net processing and development can be found in several books [8,9]. There are two easy-to-read papers that give good introductions to the two most popular neural nets, the multi-layer perceptron (MLP) and radial basis function (RBF) neural network [10,11]. Neural nets have proven useful in a variety of areas: detection, classification, multidimensional function approximation, and predictive modeling of data. They are ideal for developing non-linear transformations to map input data to outputs. Thus they can be used for classification as well as prediction.

Neural nets are "trained" by presenting examples of input/output pairs of data. For most applications, the output data has been "labeled" as to the correct class or function response. During training the parameters in the neural net are adjusted until the neural net classification performance reaches an acceptable level.

Two types of neural nets that are used extensively are the multi-layer perceptron (MLP) and the radial basis function (RBF) neural net. MLP NNs have been shown to be excellent for non-linear function approximation and for solving classification problems. They have been shown to be Bayesian classifiers. However, MLPs are not capable of performing novelty detection: an MLP can not detect when data for which it has not been trained is present at the inputs. In addition as the neural net grows in size, training can

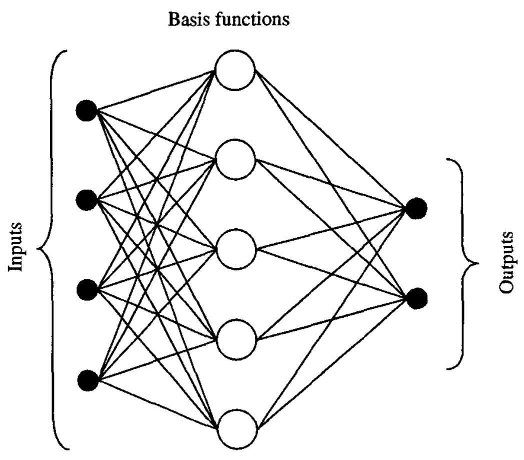


Figure 5 Radial Basis Function (RBF) Neural Net

become a complicated issue. For example, how many hidden layers should be included, what is the number of processing elements (nodes) that should be used for each of the layers, and which features should be retained. In addition the user does not know "what" the features were in the input data that led to the net's performance.

The RBF neural net has also been demonstrated to be an excellent classifier and function approximator. The architecture for the RBF NN is shown in Figure 5. For the standard implementation of the RBF, all processing elements in the middle layer apply a multi-dimensional Gaussian function to the input data. The output layer is the weighted sum of the basis function outputs.

The RBF can be thought of as a nearest neighbor classifier. As such it can perform novelty detection. This is important for many applications, as we would like the system to let us know when inputs do not match anything seen before. It also has the ability to let the user know "why" the neural net performed as it did.

Both the MLP and RBF NNs require real data that represent known fault classes to be trained. As such, when first used for a prognostics problem with limited training data, they likely will not operate very well. However as more data are collected and examples of fault classes of interest are examined, the neural net can be updated using the new data. Over time the neural net's performance should improve significantly.

Data mining/automated rule extraction

Data mining and automated rule extraction are synonymous. Data mining/rule extraction has been in development in the financial community for a number of years. In that field, managers are interested in processing data to target customers for marketing. There are a variety of commercially available software packages to perform data mining.

Those tools can be applied to development of the prognostic reasoner as well. In most rule extraction processing, rules are extracted from input data by brute force examination of the data. Input data must consist of fault class labeled samples.

Rule extraction has several advantages over neural nets:

- Comprehensibility: Something that humans can understand
- Explanation: Lets the user know "why" the system did what it did.
- Validation: The user can explore all possible set of inputs to ensure the system operates as expected under all conditions.
- Discovery: Find something "new" in the features of data not known before.
- As mentioned rule extraction is performed by brute force examination of the input data. Typical rules are found as a binary tree.

Data mining is a powerful new method with which to develop classifiers. The advantages are as described above. However as with the other "trained" classifiers, real data that is representative of all the conditions expected to be encountered need to be collected.

4. THE INTEGRATED PROGNOSTIC SYSTEM

Our prognostic system integrates enhanced neural networks with rule extraction. The system provides fault classification, novelty detection, fault prognosis and rule extraction capabilities.
In order to construct a model of the mechanical system under consideration we employ Dynamically Linked Ellipsoidal Basis Function (DL-EBF) neural networks [12]. This modification of the traditional Radial Basis Function architecture permits full analysis of the geometric relationships between the basis units and provides an additional insight into the system dynamics. The prognostic capability is obtained by quantifying the geometric relationships of different fault and normal classes represented by the basis units or, clusters of basis units. In particular, the fault prediction (time to fault) for a given state of the system may be obtained from an interpolation between normal and fault classes. Also, the time history of the system can be followed in real time and extrapolated (trended) in terms of the known fault development paths to provide a prediction based on the current dynamics of the system. It is expected that a combination of a progressively refined model that incorporates information about 'partial failure' (or, stages of fault development) and real-time trending will provide the most robust prediction. Similar to the traditional neural network paradigm, the DL-EBF network is capable of incorporating pre-existing knowledge (in the form of parametric models) while preserving the adaptive (learning) abilities. Thus, the fault development model may be progressively refined to describe very complicated paths between normal and fault states.

The approach addresses a number of problems encountered in machine health prognosis, such as the small amount of data that is usually available to characterize fault states, and the lack of comprehensive data describing the development of a fault. Approximate predictions may be obtained even if the database of diagnostic measurements is small, i.e., in the initial stages of diagnostic system development. Initially, the fault development does not need to be documented in detail and only measurements for the normal and fault condition are needed for prognostics. This is a major advantage over other neural network prediction systems, which require large databases of time histories. Also, with this approach, fault prediction and health prognosis are incorporated into one processing module (the DL-EBF network).

The TREPAN algorithm, developed by Mark Craven, [13] is used for rule extraction. The basic idea is to data-mine the neural net, which is regarded as having captured all the rules underlying the behavior of the data. A high-level flow diagram of the TREPAN algorithm is shown in Figure 5. TREPAN views the task of extracting a comprehensible rule description from the trained neural net as an inductive learning problem. In the learning task, the target concept is the function represented by the network, and the concept description produced by the TREPAN algorithm is a decision tree that approximates the neural net.

The neural net model is the embodiment of the real process of interest and forms an *oracle* that is available to answer queries during the learning process. Since the target function is simply the concept represented by the network, the rule generator uses the oracle/neural network to answer queries. The advantage of learning with queries, as opposed to the ordinary training examples, is that they can be used to gather information precisely where it is needed; the neural net oracle fills in "holes" that exist with finite samples of training data. This method exploits the interpolation ability of the neural net model. The interpolation accuracy of the neural net model will be poor for a net trained with limited data, but will improve as training is updated with more examples. At all times, the statistical accuracy of the network model can be quantified.

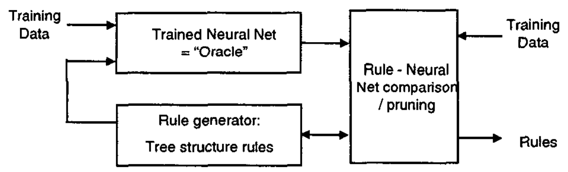

Training Data
Trained Neural Net = "Oracle"
Rule generator: Tree structure rules
Rule - Neural Net comparison / pruning
Training Data
Rules

Figure 6 TREPAN High-Level Flow Diagram

The rule generator of TREPAN is similar to conventional decision tree algorithms such as CART [14] and C4.5 [15] that learn directly from a training set. However, TREPAN does have some differences from these algorithms that ultimately lead to simpler, yet statistically better, rule sets. TREPAN develops a rule set in the form of a decision tree. The role of the nodes of a decision tree is to partition the input space in order to increase the separation of instances into different classes. For example, in C4.5, these splits are based on a single feature. TREPAN does form nodes based on single features. For those nodes, it automatically determines the decision thresholds required. In addition, the TREPAN algorithm can form node decisions that use m-of-n expressions for the splits. An m-of-n expression is satisfied when at least m of its n conditions are satisfied. For example, suppose we have three Boolean features, a, b, and c; then the m-of-n expression 2-of-$\{a, \neg b, c\}$ is equivalent to the rule "if (a and $\neg b$) or ($\neg b$ and c) or (a and c) is true, then the output of the rule is true."

A limitation of conventional tree induction algorithms is that the amount of training data available to select splits (a rule) at a node decreases with the depth of the tree. The rules developed near the bottom of a tree are often poorly chosen because these decisions are based on so few training data samples. In contrast, because TREPAN has the oracle available, in theory the oracle can generate as many training examples as desired to determine the split.

TREPAN uses two separate criteria to decide when to stop growing an extracted tree. First, a given node becomes a leaf in the tree if, with high probability, the node covers only instances of a single class. Second, TREPAN also accepts a parameter that specifies a limit on the number of internal nodes in an extracted tree. This parameter can be used to control the comprehensibility of extracted trees, since in some domains, it may require very large tress to describe the neural network to a high level of fidelity. There is a trade-off between statistical accuracy and comprehensibility (complexity) of the rule set generated.

5. Results

We have applied the integrated prognostic system described above to data that was recorded during seeded-fault test stand operations of F100 engines. Those results will be available and presented at the conference.

6. Summary and Conclusions

Prognostic methods historically have been either based on physical models, a priori rules, or statistical models. Complex systems such as turbine engines are difficult to model physically or with an expert rule set. Out processing has focused on using the advantages of statistical models for engine prognosis while integrating multiple approaches to address the traditional shortcomings of individual statistical models. Our integrated prognosis system uses a Dynamically Linked Ellipsoidal Basis Function neural network to classify engine sensor data and thereby statistically model the state of the engine with respect to known faults. As is true of all basis-function neural networks, the DL-EBF provides novelty detection capability. The dynamic-link feature allows assessment of trending from the nominal class to each of the identified fault classes. This means that quantitative prognostics are built into the network functionality. To add comprehensibility to the system, the TREPAN approach is used to develop a tree-structured rule set that closely approximates the classification of the neural network.

The authors gratefully acknowledge the assistance of Steve Budrow and Dale Southwick of Pratt & Whitney, a United Technologies Company, in providing engine sensor data.

REFERENCES

[1] J.C. Giarratano and G.D. Reley, Expert Systems: Principles and Programming, Third Edition, PWS Publishing, 1998.

[2] L. Zadeh, "Fuzzy set," Inform. Control., vol. 8, pp 338-353, 1965. Reprinted in Fuzzy Models for Pattern Recognition, J.C. Bezdek and S.K. Pal, eds., IEEE Press, 1992.

[3] J.C. Bezdek and S.K. Pal, eds, Fuzzy Models for Pattern Recognition: Methods that Search for Structure in Data, IEEE Press, 1992

[4] J. Yen, R. Langari, L.A. Zadeh, Indusrial Applicatons of Fuzzy Logic and Intelligent Systems, IEEE Press, 1995

[5] J.J. Gertler, Fault Detection and Diagnosis in Engineering Systems, Marcel Dekker, Inc., 1998.

[6] G.E.P. Box and G.M. Jenkins, Time Series Analysis: Forecasting and Control, Holden-Day, 1976.

[7] S.L. Marple, Jr., Digital Spectrum Analysis with Applications, Prentice-Hall, 1987.

[8] R. Hecht-Nielsen, Neurocomputing, Addison-Wesley Publishing Co., 1990.

[9] S. Haykin, Neural Networks–A comprehensive foundation, Macmillan College Publishing Company, Inc., 1994.

[10] R.P. Lippmann, "An introduction to computing with neural nets", IEEE ASSP Mag., pp 4-22, April 1987

[11] D.R. Hush and B.G. Horne, "Progress in supervised neural networks: What's new since Lippmann?", IEEE Signal Processing Mag., Jan 1993.

[12] D. Wroblewski, "Machine Health Prediction for Condition-Based Maintenance", unpublished.

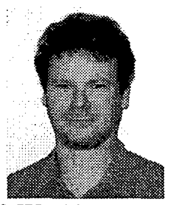
[13] M. W. Craven and J. W. Shavlik, "Extracting tree-structured representations of trained networks," *Advances in Neural Information Processing Systems*, Vol. 8, MIT Press.

[14] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, CART: Classification and Regression Trees, Chapman and Hall/CRC 1993 (CRC reprint 1998).

[15] J. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann.

Tom Brotherton received his B.S. degree from Cornell University (1974), an M.A.Sc. from the University of Toronto (1976) and the Ph.D. from the University of Hawaii (1982) all in electrical engineering. He was an assistant professor in the Information and Computer Sciences Department at the University of Hawaii in 1983. From 1982-1999 he was with the Orincon Corporation in San Diego, CA. He is now with the Intelligent Automation Corporation. There he is the chief scientist for the company. His current interests are in the development of adaptive signal and image processing techniques using fuzzy logic and neural nets for machine condition and fault monitoring as well as medical systems applications.

Gary Jahns received his B.S. degree in physics from Harvey Mudd College (1966), and an M.A. (1968) and Ph.D. (1973), also in physics, from the University of California, Irvine. He worked in the area of controlled fusion research at Oak Ridge National Laboratory, TN, and at General Atomics in San Diego, CA. Since 1992, he has been a Senior Principal Engineer with ORINCON Corporation in San Diego, CA, where he has applied neural networks to the analysis of fusion energy systems, medical imaging, underwater acoustics, and semiconductor manufacturing. His current interests are in the development of neural nets for machine condition and fault monitoring and prognostics.



Jerry Jacobs received B.A. degrees from the University of California, San Diego in Biology (1980) and Chemistry (1981), a M.S. degree in Marine Biology from the Scripps Institution of Oceanography (1984), and a Ph.D. degree in Zoology from the University of Washington (1996). After leaving the University of Washington, he joined the Orincon Corporation in San Diego, CA. There he is an engineer for the Machine Diagnostic Systems Division. His current interests are in the development of advanced diagnostic and prognostic systems for aircraft, as well as the application of neural networks and advanced signal processing to the automation of medical systems.

Dariusz Wroblewski is a Senior Principal Engineer at Orincon Corporation, San Diego. He received his Ph.D. in Electrical Engineering from the University of Wisconsin-Madison. His recent research interests are in application of artificial intelligence methods to data modeling and analysis,

including problems in machine health monitoring, magnetic confinement of high-temperature plasma, medical devices automation, and bio-informatics.