Received December 29, 2014, accepted January 23, 2015, date of publication April 20, 2015, date of current version May 7, 2015.

Digital Object Identifier 10.1109/ACCESS.2015.2422833

Incremental Classifiers for Data-Driven Fault Diagnosis Applied to Automotive Systems

CHAITANYA SANKAVARAM[1,2], (Member, IEEE), ANURADHA KODALI[1,3], (Member, IEEE), KRISHNA R. PATTIPATI[1], (Fellow, IEEE), AND SATNAM SINGH[4,5], (Member, IEEE)

[1]Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT 06269, USA
[2]General Motors Global Research and Development, Warren, MI 48090, USA
[3]NASA Ames Research Center, Moffett Field, CA 94035, USA
[4]General Motors India Science Lab., Bangalore 560066, India
[5]CA Technologies, Bangalore 560017, India

Corresponding author: C. Sankavaram (chaitanya.sankavaram@gmail.com)

This work was supported by the General Motors India Science Laboratory, Bangalore, India.

ABSTRACT One of the common ways to perform data-driven fault diagnosis is to employ statistical models, which can classify the data into nominal (healthy) and a fault class or distinguish among different fault classes. The former is termed fault (anomaly) detection, and the latter is termed fault isolation (classification, diagnosis). Traditionally, statistical classifiers are trained using data from faulty and nominal behaviors in a batch mode. However, it is difficult to anticipate, *a priori*, all the possible ways in which failures can occur, especially when a new vehicle model is introduced. Therefore, it is imperative that diagnostic algorithms adapt to new cases on an ongoing basis. In this paper, a unified methodology to incrementally learn new information from evolving databases is presented. The performance of adaptive (or incremental learning) classification techniques is discussed when: 1) the new data has the same fault classes and same features and 2) the new data has new fault classes, but with the same set of observed features. The proposed methodology is demonstrated on data sets derived from an automotive electronic throttle control subsystem.

INDEX TERMS Adaptive learning, automotive systems, ensemble systems, fault diagnosis, incremental classifiers.

I. INTRODUCTION

The complexity of automotive systems is increasing at a rapid rate to improve vehicle performance, and to provide the necessary convenience and safety features with increasing levels of automation and control. Automobiles consist of huge amounts of software embedded on micro-processors, also termed electronic control units (ECUs), which continuously interact with mechanical and electrical components through sensors and actuators for vehicle operation and control. There is also an increased dependence on ECUs for monitoring the health condition of vehicular components. This is because operational problems associated with degraded components, failed sensors, and improperly implemented controls affect the efficiency, safety and reliability of vehicles, which, in turn, lead to potentially large warranty costs. Hence, it is crucial to quickly detect and isolate (classify/diagnose) faults in order to improve vehicle availability and customer satisfaction.

Methods for fault diagnosis can be categorized into the following three approaches: data-driven, model-based, and knowledge-based [1]. The *data-driven approach* is preferred when the system monitoring data for nominal and degraded conditions is available. One of the common ways to perform data-driven fault diagnosis is to employ neural network and statistical machine learning techniques to classify data into nominal (healthy) and a fault class or into different fault classes [2], [3]. The former is termed fault (anomaly) detection and the latter fault isolation (classification, diagnosis). The *model-based approach* employs consistency checks between the sensed measurements and the outputs of a mathematical model. The expectation is that inconsistencies are large in the presence of malfunctions and small in the presence of normal disturbances, noise and modeling errors. Two main methods of generating the consistency checks are based on observers [4], [5] (e.g., Kalman filters, reduced-order unknown input observers, interacting multiple models, particle filters) and parity relations [6] (dynamic consistency checks among measured variables stemming from hardware or information redundancy relations). The *knowledge-based approach* uses graphical models such as dependency graphs (digraphs), Petri nets, multi-signal (multi-functional) flow graphs, and Bayesian networks for diagnostic

knowledge representation and inference [7]–[9]. This paper presents a data-driven incremental learning approach for fault diagnosis in automotive systems based on fleet warranty data.

Automotive companies collect a variety of vehicle health data from test fleet and production vehicles via telematics and dealer diagnostics services. These data sources acquire different types of vehicle data at different sampling rates. For example, dealer diagnostic data is collected when a vehicle comes for repair at a dealer shop; this warranty data, collected infrequently, includes the diagnostic trouble codes (or fault codes), freeze frame data (engineering variables or so-called parameter identifiers (PIDs) collected from various sensors, such as engine speed, temperature and battery voltage) at the onset of a fault code, repairs/replacement actions specified in terms of repair codes, and structured/unstructured text in the form of customer verbatim. The fleet data is collected at a much higher sampling frequency (e.g., every few ignition cycles) for inferring the overall health of vehicle subsystems, such as the engine and/or transmission system, emission system, airbag system, anti-lock brake system, tire pressure; this data is gathered even when the vehicle is functioning normally. Thus, the test fleet data evolves over a period of time with new fault classes added as they occur in the field. As a result, a diagnostic classifier (model) trained on an earlier data set may become inconsistent or even obsolete with new data. In addition, with new software/hardware content being constantly integrated into the vehicles, a diagnostic model built for one vehicle type might not always be appropriate across all vehicle ranges, because new software/hardware leads to new fault codes/failure modes. Therefore, periodic updating of the classifier structure and its parameters is necessary.

Traditional data-driven techniques operate in a batch mode and need to retrain the diagnostic model structure and parameters from scratch by combining the old and new instances. If classifiers are trained infrequently, they may miss new behavioral trends (the so called ''concept drift'' [10]). Batch training also suffers from increasingly larger computational complexity for training as the data is accumulated. In order to overcome this limitation, it is essential to incrementally adapt the classifiers to evolving data.

The problem of learning from evolving databases can be divided into four major categories depending on how the data evolves over time. They are: (*i*) The new data has the *same fault classes* and *same features*; *(ii)* The new data has the *same fault classes*, but with *new features*, *(iii)* The new data has *new fault classes*, but with the *same features*, and *(iv)* The new data has *new fault classes* and *new features*. In this paper, we consider categories (*i*) and (*iii*), which are most common in automotive systems because sensor suite is typically fixed during vehicle design.

The contributions of this paper are as follows:

1) An age-based incremental learning methodology for adaptive learning of classifier models with evolving databases;
2) An ensemble of classifiers in an incremental learning framework for accurate fault classification when new data and new fault classes are present;
3) An application to datasets obtained from GM's electronic throttle control (ETC) subsystem; the datasets represent vehicle health data and the corresponding fault codes collected from model year (MY) 2008 and MY 2009 vehicles; and
4) A systematic validation analysis of the incremental learning classifier on a number of scenarios to demonstrate diagnostic accuracy improvement as the data evolved from one vehicle model year to the next.

A major advantage of the incremental learning approach in an automotive application is that it minimizes or avoids retraining of classifiers when a new model year vehicle is released into the market. The methodologies and scenarios presented in this paper provide insights for automotive diagnostic engineers to determine if an existing classification/diagnosis algorithm performs effectively across all model years and all vehicle types; if not, how often should these algorithms be adapted/learned? In addition, evolving databases is a common concern in many areas, for instance, facial recognition where incremental learning of various facial expressions, lighting conditions, orientation etc., is of vital importance; therefore, the work presented in this paper has a wide range of real world applications, including aerospace vehicles, medical equipment, cyber security, power networks, semiconductor fabrication facilities, and social networks, to name a few.

The paper is organized as follows. An overview of the existing incremental learning techniques is provided in Section II. The proposed data-driven adaptive learning framework is described in Section III. The framework is evaluated using the data derived from an automotive ETC subsystem and the associated experimental results are presented in Section IV. Finally, the paper concludes with a summary in Section V.

II. BACKGROUND

The incremental learning techniques for evolving data employ one of the following three approaches:

(i) Re-train with new instances and compressed representation of old instances,
(ii) Remember the functional mapping between the old features and fault classes and modify the map in light of new data, or
(iii) Update the weighted metrics of each class with new data.

Yamauchi et al. [11] proposed two methods, viz., incremental learning with retrieving interfered patterns (ILRI)-R, and ILRI-G, for learning new patterns incrementally when the data has the *same classes and same features*. In the first method, the incorrectly classified patterns from the old database of instances are selected for subsequent training with new data (reminiscent of AdaBoost [12]). In the second method, the past patterns are reconstructed from the

previously trained classifier knowledge (for example, support vectors in the case of a Support Vector Machine (SVM)). This avoids remembering the entire database of instances. However, both methods suffer from either requiring substantial memory or severe sub-optimality. Clustering techniques also can be employed to compress the past database of instances [13]. However, this is not widely used because of the behavioral differences of new data from the original data.

Incremental SVMs have been developed to adapt classifiers to evolving data. In [14], the algorithm stores only the support vectors instead of the entire database, thereby reducing the storage space. A sequential (one instance at a time) SVM learning algorithm is proposed in [15], where the Kuhn-Tucker conditions for the old data are enforced with the addition of every new instance. This is extended in [16] by using an optimization technique called multi-parametric programming to update the trained model with multiple data points in a single step rather than a series of optimization problems for each data point. Incremental fusion techniques are also proposed using SVM as the base classifier in [17], which employs Bayesian averaging. In [18], ensemble SVM classifiers are developed using the boosting strategy. An incremental version of probabilistic neural network (PNN) is also proposed to classify new classes [19]–[21]. Skocaj et al. [22] proposed an incremental learning method by combining discriminant analysis and principal component analysis to incorporate both new features and new classes. A pattern recognition-based method is proposed in [23] for detecting drifts in non-stationary processes. This method tracks the changes observed in classes' conditional probability distributions after the classification of each new pattern and when the accumulated changes reach a suitable predefined threshold, the classifier parameters are adapted online.

Techniques based on decision trees are widely used to construct incremental classification strategies. A series of algorithms, ID4, ID5, and ID5R have evolved with successive improvements made to the basic decision tree method, ID3 [24], [25]. A disadvantage of these models is the need to update the branches of the nodes with the inclusion of new training instances. This sometimes equals the effort required in constructing a new tree from scratch.

In [26], a heuristic hill-climbing search method is employed to incrementally learn the structure of a Bayesian network by adding/deleting links at each step. The need to update the network structure is decided based on a score at each step. In the same vein, tree-augmented Naïve Bayes (TAN) method incrementally adapts the network structure when the new instances result in invalidated branches [27]. An averaged tree- augmented Naïve Bayes method is discussed in [28] to improve the overall performance of class probability estimation by averaging the class membership probabilities from a set of TAN classifiers.

Fusion techniques combine the decisions from more than one classifier for better performance. Learn++ [29], inspired by AdaBoost, generates a new ensemble of ''weak classifiers'' based on a weighted distribution of errors on new data. This method is superior to compression because of the inclusion of previously trained classifiers in the decision process. An ensemble-based incremental-learning algorithm for fault classification is discussed in [30] to learn new relationships corresponding to new operational conditions while keeping the previous classifiers to retain the existing knowledge. Another variant of Learn++, called Learn++.NC is proposed in [31] with a new inference rule to avoid the bias for old classes. Normally, there will be more classifiers categorizing the old classes, thereby increasing their corresponding weights in the inference rule. This can be avoided by increasing the number of classifiers to categorize new classes.

Although there is considerable literature on incremental learning and ensemble classifiers, this paper presents an automotive application of the age-based incremental learning methodology. The scenarios presented in this paper are to guide automotive engineers to determine whether or not a classifier model needs adaptation and, if needed, to adaptively learn the new classifier model. The subsystem considered for the demonstration of the methodologies is the ETC subsystem and the datasets pertain to vehicle health data and the corresponding fault codes collected from MY 2008 and MY 2009 vehicles.

III. METHODOLOGY

Our approach to developing adaptive classification strategies is guided by the following criteria given in [29]:

a) Learn additional information from new data,
b) Not requiring access to the original data used to train the existing classifier,
c) Preserve the previously acquired knowledge (avoid ''catastrophic forgetting''), and
d) Accommodate new fault classes as they occur with new data.

The proposed incremental data-driven classification framework accommodates all the four evolving data types as shown in Fig. 1. Firstly, when the features and fault classes are the same, the existing diagnostic model is evaluated on the new data. If this results in significant degradation in diagnostic accuracy, the diagnostic model is updated via incremental learning algorithms. Secondly, when new features are observed, it is important to utilize the information from these features and update the diagnostic model. This is crucial because newly observed feature sets may become salient for classification with higher predictive power than the current ones. Incremental feature selection techniques that can constantly update the set of features used for classification are needed. Thirdly, when the observed features correspond to a new fault class, then the new instance is classified as an ''unknown'' class and the model is updated with the new class information. Lastly, when both new features and new classes are observed, the considerations in categories 2 and 3 are used to adapt the diagnostic model over time [31]. In this paper, the discussion on incremental

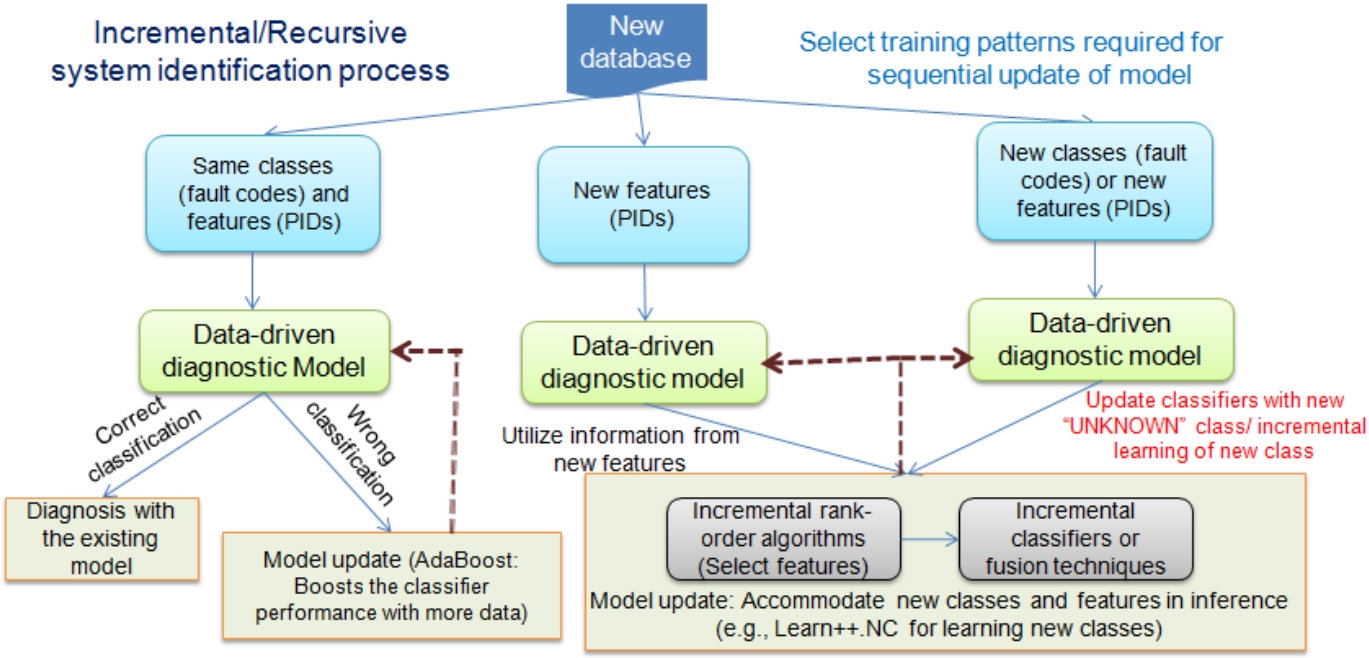

FIGURE 1. Adaptive (or Incremental) Data-Driven Reasoners.

learning of classifiers will be limited to the following two categories because our application context is limited to these two categories:

(i) New data with the same classes and the same features, and
(ii) New data with new classes, but with the same features.

The diagnostic model comprises of ensemble of classifiers including AdaBoost, bagging, and Learn++.NC classifiers.

A. ENSEMBLE OF CLASSIFIERS FOR INCREMENTAL LEARNING

Ensemble classifiers generally seek to improve the classification accuracy by generating a set of classifiers, each trained with samples from training data using a different probability distribution. The outputs of these classifiers are then combined to obtain the final classification rule. Some examples of ensemble classifiers include the AdaBoost [12], bagging [32], [33], Learn++ [29], Learn++.NC [31], and so on. Here, we will briefly discuss the different ensemble classifier techniques and the major differences/drawbacks among them.

1) ADABOOST

AdaBoost is a technique widely known to improve the performance of moderately accurate (also termed weak) classifiers. The training data $\mathcal{D}\{(x_i, y_i), i = 1, 2, \ldots, N\}$ is sampled with replacement to create an ensemble of classifiers $\{t = 1, 2, \ldots, T\}$. The initial distribution to sample the training patterns is assumed to be uniform over the training data as in (1), i.e., the samples are chosen with equal probability from the training data.

$$D_1(i) = \frac{1}{N}, \quad \forall i, N = \text{total number of patterns} \tag{1}$$

The weight vector for each training pattern is initialized as,

$$w^1_{i,y} = \frac{D_1(i)}{C-1}, \quad \forall i, y \neq y_i \tag{2}$$

where $y_i \in \Omega = \{1, 2, \ldots, C\}$ and y is the predicted class. The training samples are supplied to the classifier, and based on the classifier performance, the distribution and weight vectors are updated such that the patterns which are most difficult to classify (or incorrectly classified) are likely to be included in the next training set. The distribution update rule, at any iteration, is given by,

$$D_t(i) = \frac{W^t_i}{\sum_{i=1}^{N} W^t_i}; \quad t = 1, \ldots, T \tag{3}$$

and $W^t_i = \sum_{y \neq y_i} w^t_{i,y}$. The weight vectors for each training pattern are updated as in (4),

$$w^{t+1}_{i,y} = \begin{cases} w^t_{i,y}\beta_t & \text{if } y = y_i \\ w^t_{i,y} & \text{if } y \neq y_i. \end{cases} \tag{4}$$

It can be seen from (4) that the weight of pattern i is multiplied by the normalized error given by,

$$\beta_t = \frac{\varepsilon_t}{1 - \varepsilon_t} \tag{5}$$

when the pattern is correctly classified and left unchanged when incorrectly classified implying that more weight is

given to the incorrectly classified instance and less weight for the correctly classified instances. The error (pseudo-loss, ε) for the classifier 't' can be computed as,

$$\varepsilon_t = \sum_{\substack{i=1 \\ y \neq y_i}}^{N} D_t(i). \tag{6}$$

When the training error is greater than 50% (or) when the training accuracy is 100%, the training of the classifier ensemble is stopped. Subsequently, when a test pattern is given to the classifier ensemble, the final class label L_f is decided based on the weighted voting rule given by,

$$L_f = \arg\max_{y \in \Omega} \sum_{t=1}^{T} \log \frac{1}{\beta_t} h_t(x, y) \tag{7}$$

where $h_t(x, y)$ represent the output class prediction from the ensemble classifier t. More details on the AdaBoost algorithm can be found in [12].

2) BAGGING

*B*ootstrap *aggregat*ing (Bagging) is another widely used ensemble technique to improve the classifier performance. The algorithm generates multiple training subsets from the original training set and classifiers are trained with each of the training subsets. Given a training dataset $\mathcal{D}$ of size N, the algorithm generates M bootstrap training subsets of size N; here, the training dataset is sampled with replacement. Therefore, some of the original training examples will not be selected for inclusion in the bootstrap training set and others will be chosen one or more times. The M individual classifiers thus generated are combined together to form the classifier ensemble. When a test pattern is presented to the ensemble, the final class label is decided using the majority voting rule, viz., the class that is predicted *the most* by the trained classifiers is chosen as the class label for the test pattern.

The primary difference between bagging, and boosting is that, unlike AdaBoost, there is no distribution update or pattern weight update rule based on classifier performance; instead, bagging chooses training patterns at random with replacement based on a uniform distribution. More details can be found in [33].

3) LEARN++

The Learn++ algorithm builds on the AdaBoost idea; it generates a set of classifiers and the final classification decision is obtained by combining the decisions of individual classifiers in a majority voting scheme that is reminiscent of the AdaBoost algorithm.

Given a dataset $\mathcal{D}_k\{k = 1, 2 \ldots K\}$, the pattern weights and the pattern distribution are initialized to be,

$$w_1(i) = D_1(i) = \frac{1}{N_k}, \quad \forall i, i = 1, 2, \ldots, N_k. \tag{8}$$

The distribution at any iteration is obtained by normalizing the weights as,

$$D_t = \frac{w_t}{\sum_{i=1}^{N_k} w_t(i)}; \quad t = 1, 2, \ldots T_k. \tag{9}$$

Subsequently, a training subset is drawn according to D_t and the classifier is trained with this training subset. The classifier error is computed as the sum of distribution weights of misclassified instances as in (10).

$$\varepsilon_t^k = \sum_{i=1}^{N_k} D_t(i) h_t^k(x_i, y) \tag{10}$$

where $h_t(x_i, y)$ is equal to 1 for incorrectly classified instances (that is, $h_t(x_i, y) \neq y_i$) and 0 otherwise. If this error is less than 0.5, the classifier is considered for the ensemble, else it is discarded. The classifier decisions obtained thus far are combined using a weighted majority voting rule to compute the ensemble decision as,

$$H_t^k = \arg\max_{y \in \Omega} \sum_{z: h_z^k(x) = y} \log(1/\beta_z^k), \quad z = \{1, 2, \ldots, t\}. \tag{11}$$

Similar to the AdaBoost algorithm, the normalized error of an individual classifier error can be computed from (5) using (10). Given the individual classifier decisions, the ensemble error is calculated to be,

$$E_t^k = \sum_{i: H_t^k(x_i) \neq y_i} D_t(i). \tag{12}$$

Subsequently, the normalized ensemble error is obtained as,

$$B_t^k = \frac{E_t^k}{1 - E_t^k} \quad 0 < B_t^k < 1. \tag{13}$$

This normalized error is used in updating the weight of training patterns (see (14)). Here, similar to AdaBoost, the weight associated with the correctly classified instance is reduced and for the incorrectly classified instance, it is left unchanged, meaning, it is more likely that the incorrectly classified instance is chosen for training in the next iteration.

$$w_{t+1}(i) = w_t(i) \times \begin{cases} B_t^k, & \text{if } H_t^k(x_i) = y_i \\ 1 & \text{otherwise} \end{cases} \tag{14}$$

After generating T_k classifiers for each database $\mathcal{D}_k$, the final classifier decision of a test pattern is computed via a majority voting rule – the class with the highest vote among all the classifiers, and is given by,

$$L_f = \arg\max_{y \in \Omega} \sum_{k=1}^{K} \sum_{t: h_t(x) = y} \log(1/\beta_t^k). \tag{15}$$

The primary difference between Learn++ and AdaBoost is that in Learn++, at each iteration, the pattern distribution is updated based on the ensemble classifier performance H_t^k(i.e., classifiers learned so far (see (14)),

whereas in AdaBoost, the distribution is updated based on a single classifier performance at the previous iteration (see (4)). In other words, Learn++'s pattern distribution rule is fine-tuned to accommodate the new data, whereas in AdaBoost, the pattern distribution is adjusted to improve the classifier performance. Accordingly, when a new dataset is presented, Learn++ generates additional classifiers to accommodate new data in the form of new instances or new classes.

Suppose a new dataset with new classes is presented. The algorithm generates a number of classifiers until the decisions of previously trained classifier ensembles are out-voted with the decisions of newly trained classifiers. This is a major drawback of Learn++ algorithm because, since all the old classifiers are guaranteed to misclassify the new class, the algorithm has to generate a significant number of new classifiers so as to out-vote the decisions from the old classifiers. In order to address this issue of classifier proliferation, Muhlbaier et al. [31] developed Learn++.NC algorithm in which each classifier is assigned a voting weight based on their performance on the training data and this voting weight is dynamically updated depending on whether or not a classifier is trained on a particular class. The description of the Learn++.NC is presented in the subsequent section.

4) LEARN++.NC

The main idea of Learn++.NC is that the algorithm stores the list of class labels on which each classifier is trained, and depending on whether or not a classifier is trained on a particular class, the voting weight of the classifier is adjusted. This is called dynamically weighted consult and vote (DWCAV) scheme.

Given a dataset $\mathcal{D}_k\{k = 1, 2, \ldots K\}$, the initial distribution is selected as in (9) and the dataset thus obtained is used to train the base classifier. The classifier performance is computed using equations similar to (10) and (5) in Learn++ algorithm and the list of class labels (CL_t^k) for which each classifier is trained is stored. Subsequently, the DWCAV subroutine is called to compute the ensemble classifier decision. The ensemble error is calculated from the incorrectly classified instances and, consequently, the normalized ensemble error is used to update the weights of classifier instances. These steps are similar to Learn++ algorithm except that Learn++.NC keeps track of the class labels for each trained classifier and accordingly updates the voting weight of the classifier.

Initially, each classifier, h_t is assigned a regular voting weight, W_t^k computed via normalized classification error based on the classifier performance on the training dataset as in (16).

$$W_t^k = \log\left(1/\beta_t^k\right) \tag{16}$$

This classifier weight is dynamically adjusted based on the classifiers' decision on current instance x_i. The dynamically updated voting weight of classifier t for instance x_i is given by,

$$W_t^k(x_i) = \begin{cases} W_t^k.\left(1 - P_c(x_i)\right), & t : \omega_c \notin CL_t \\ W_t^k, & t : \omega_c \in CL_t. \end{cases} \tag{17}$$

Here $P_c(x_i)$ is the class-specific confidence that represents the confidence of classifiers trained on class ω_c in choosing class ω_c for an instance x_i. If a classifier t is not trained on a particular class label ω_c, the voting weight of that classifier is reduced. Formally,

$$\text{Class-specific confidence } P_c(x_i) = \frac{\sum_k \sum_{t:h_t^k(x_i)=\omega_c} W_t^k}{Z_c} \tag{18}$$

and $Z_c = \sum_k \sum_{t:\omega_c \in CL_t^k} W_t^k$. The final ensemble classifier decision for an instance x_i is the class with the largest sum of adjusted voting weights (see (19)).

$$L_f = \arg\max_{\omega_c \in \Omega} \sum_{k=1}^{K} \sum_{t:h_t^k(x_i)=\omega_c} W_t^k(i) \tag{19}$$

More details about the algorithms can be found in [31].

Several statistical and pattern classification techniques, such as support vector machines, k-nearest neighbor, partial least squares, probabilistic neural network and relevance vector machine, can be employed as the base classifiers in an ensemble learning framework.

B. BASE CLASSIFIERS FOR INCREMENTAL LEARNING

1) SUPPORT VECTOR MACHINE (SVM)

Support vector machine is one of the most widely used supervised learning algorithms for classification. Given a set of training samples belonging to different classes, support vector machines find an optimal decision boundary (also called hyperplane) that maximizes the margin between the classes [34]. Therefore, SVM is also known as the maximum-margin classifier. The vectors that define the optimal separating hyperplane are known as the support vectors. These support vectors lie closest to the decision surface and can be used to estimate the fault class of a test feature vector. In the case of non-linear classification, where a linear boundary is not appropriate, a kernel function is used for mapping the data onto a higher dimensional feature space and the optimal hyper-plane is then constructed. Some of the kernel functions that are commonly used are polynomial, Gaussian, radial basis functions, and hyperbolic tangent [3].

2) k - NEAREST NEIGHBOR (KNN)

The *KNN* classifier is a simple non-parametric method that classifies test vectors based on the samples from the training data [2]. The classifier finds the k-nearest points to a test vector from the training data, and the class with the *maximum a posteriori* probability within those k points is declared as the most-likely class. Normally, k is chosen as an odd number

to avoid ties. Mathematically, the *posterior* class probabilities $P(c_i|\underline{x}_{new})$ are given by,

$$P\left(c_i|\underline{x}_{new}\right) = \frac{k_i}{k} P\left(c_i\right) \tag{20}$$

where k_i is the number of vectors belonging to class c_i within the $k-$nearest points. $P(c_i)$ is the prior probability of class c_i. A new test sample $\underline{x}_{new}$ is assigned to the class c_i with the highest *a posteriori* class probability $P(c_i|\underline{x}_{new})$.

3) PARTIAL LEAST SQUARES (PLS)

PLS is a conjugate gradient-based regression technique used to find a set of components that explain the covariance between the independent training data matrix X (of size $N \times J$, N – number of patterns and J – number of measurement variables or features), and the dependent matrix Y (of size $N \times M$, M- number of fault classes). Both matrices X and Y are decomposed into a number of components, which is known as the model reduction order plus residuals. Each component captures certain amount of variation in the data. This reduction order is determined by cross-validation [35].

The decompositions are given by,

$$\begin{aligned} X &= TP^T + E \quad T \in R^{N\times L}, \quad P \in R^{J\times L}, \quad E \in R^{N\times J} \\ Y &= UQ^T + F \quad U \in R^{N\times L}, \quad Q \in R^{M\times L}, \quad F \in R^{N\times M} \end{aligned} \tag{21}$$

where L is the model reduction order, T and U are score matrices, and P and Q are loading matrices, while E and F are residuals. The goal of the PLS algorithm is to determine highly correlated latent variables (or the score vectors) that not only capture the variations in the input data X, but also the variations that are most predictive of the output data Y. Once the latent variables are extracted, a least squares regression is performed to estimate the fault class. The score vectors are determined using nonlinear iterative partial least squares (NIPALS) algorithm [36].

4) PROBABILISTIC NEURAL NETWORK (PNN)

PNN is a supervised method that computes the likelihood of an input vector belonging to a specific class based on the learned probability distributions of each class [3]. The learned patterns can also be weighted with *a priori* probability (relative frequency) of each category and misclassification costs to determine the most likely class for a given input vector. If the relative frequency of the categories is unknown, then all the categories can be assumed to be equally likely and the determination of category is solely based on the closeness of the input feature vector to the class distribution function.

5) RELEVANCE VECTOR MACHINE (RVM)

RVM is a supervised machine learning algorithm that has the same functional form as SVM, but employs a Bayesian approach to learning. The algorithm provides probabilistic outputs for the class membership, given an input $\underline{x}_i$ based on the training data. The idea of RVM is to learn a dependency model of the targets Y and input data X, and subsequently make accurate predictions of class labels for any unseen or new pattern. Similar to SVM, RVM employs kernel functions to describe the input-output relationship and the output (or class) predictions are based on the functional mapping $F(X)$ learned from the training data.

The targets are assumed to be of the form,

$$y_n = F\left(x_n; w\right) + \varepsilon_n \tag{22}$$

where, ε is assumed to be Gaussian process noise with zero mean and variance σ^2, and $w = (w_0, w_1, w_2, \ldots, w_m)^{\mathrm{T}}$ is a weight vector. Assuming the conditional independence of y_n, the likelihood of the training data is given by,

$$p(y|w, \sigma^2) = (2\pi\sigma^2)^{-N/2} \exp\left\{-\frac{1}{2\sigma^2}\|y - \Phi w\|^2\right\} \tag{23}$$

where prior on w is assumed to be Gaussian and is given by,

$$p(w|\alpha) = \prod_{m=0}^{N} N(w_m|0, \alpha_m^{-1}). \tag{24}$$

Here $\alpha = \{\alpha_m\}$ is a vector of hyper-parameters, with one hyper-parameter α_m assigned to each model parameter w_m and $\Phi = [\phi(x_1), \phi(x_2), \ldots, \phi(x_N)]^{\mathrm{T}}$ in which $\phi(x) = [1, K(x_n, x_1), K(x_n, x_2), \ldots., K(x_n, x_N)]^T$ and $K(x_n, x_i)$ is a kernel function.

With the prior density (24) and the likelihood function (23), the posterior density over the weights, $p(w|y, \alpha, \sigma^2)$ can be computed using Bayes rule [37]. Then, given a test point x_s, the output class prediction in the form of probability density function is given by,

$$p(y_s|y) = \int p(y_s|w, \alpha, \sigma^2)\, p(w, \alpha, \sigma^2|y)\, dw\, d\alpha\, d\sigma^2. \tag{25}$$

More information on RVM can be found in [37].

IV. APPLICATION OF INCREMENTAL LEARNING METHODOLOGY TO AN AUTOMOTIVE ELECTRONIC THROTTLE CONTROL SYSTEM

The methodology for adaptive learning is validated using a dataset derived from an automotive electronic throttle control (ETC) subsystem. The subsystem details and the experimental results are discussed in the following subsections.

A. ELECTRONIC THROTTLE CONTROL SYSTEM DESCRIPTION

The electronic throttle control system mainly comprises of throttle pedal assembly, electronic throttle body, throttle position sensor, and vehicle electronic control module or powertrain control module (ECM/PCM). Fig. 2 shows the block diagram of electronic throttle control subsystem. The ETC subsystem determines the necessary throttle opening using various input sensors (for e.g., accelerator pedal position, engine speed, vehicle speed, etc.) and is responsible for controlling the servomotor (actuator) to achieve the required throttle position via a closed-loop control algorithm in the ECM. The ECM is also responsible for monitoring the

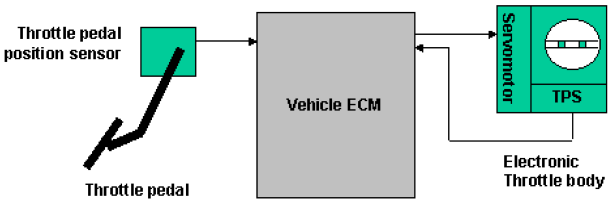

FIGURE 2. Electronic Throttle Control Subsystem [38].

health of the subsystem by processing parameter identifier data (PIDs) collected from various sensors. The parameter identifier data contains valuable information about vehicle operating condition — e.g., engine speed, engine coolant temperature and so on.

ETC System Data Description: The ETC subsystem data for model year (MY) 2008 and MY 2009 vehicles is obtained from the dealer diagnostic datasets. The data consists of fault codes, parameter identifiers or status parameters (PIDs) collected from various sensors and the age of the vehicle when the parameters are recorded. The total number of cases (or records) collected for MY 2008 and MY 2009 were 1932 and 655, respectively, with 27 distinct fault codes and 638 PIDs. The age of the vehicles extend up to 1084 days (i.e., approximately 3 years and 1 month in service). We also observed that MY 2008 vehicles had 26 fault codes; of which 4 fault codes were not present in the MY 2009 vehicle data. Similarly, MY 2009 cases had 23 fault codes with 1 new fault code that was not present in the data collected from MY 2008 vehicles. The list of fault codes is included in the Appendix.

B. FEATURE SELECTION FOR FAULT DIAGNOSIS

To decrease the implementation complexity of classification approaches, information gain (IG) algorithm [39] is employed to rank order the PIDs and the optimal number of PIDs are then selected for classification/diagnosis. The idea of IG algorithm is to evaluate the amount of information contributed by each feature to a particular class; and the subset of features with high information content is used for classification. In the current application, IG algorithm is employed on 638 status parameters and the top 16 PIDs are chosen based on the diagnostic accuracy on test data. Fig. 3 shows the value of mutual information for each of the status parameters and the top features are circled in red.

C. RESULTS AND DISCUSSION

After selecting the top-ranked PIDs, the MY 2008 and MY 2009 cases are arranged in an age-ordered sequence for incremental learning analysis. The age-ordered datasets are divided into training and testing subsets for experimentation with the age-based incremental learning scenarios discussed below. Throughout our experimentation process, the base classifiers discussed earlier are employed in a Learn++.NC ensemble framework. The performance of incremental ensemble classifier is illustrated using four

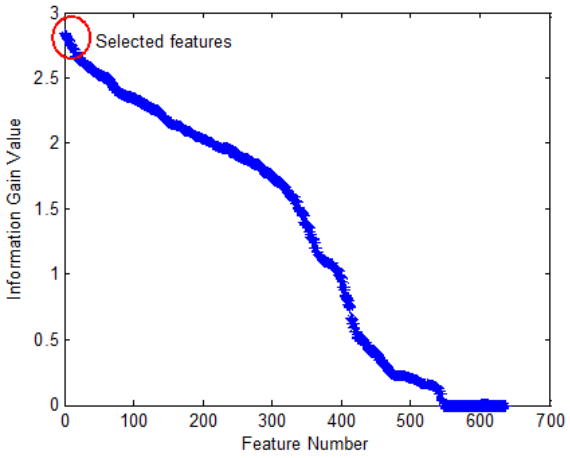

FIGURE 3. Plot of Mutual Information for 638 Status Parameters.

age-based scenarios. The following subsections briefly discuss the scenarios and experimental results:

Scenario 1 - Window Method: MY 2008 and MY 2009 data are arranged in sequential order w.r.t. age, but the data is partitioned according to a certain window size.

In the *Window method*, each model year's data is arranged in the increasing order of age, and the data is divided into four subsets based on the age of the vehicle. So, MY 2008 data is divided into four datasets (D_1–D_4) and MY 2009 is partitioned into another four datasets (D_5–D_8). Each dataset within a model year has approximately the same incremental age (e.g., 271 days for MY 2008). Refer to Fig. 4 for the data partition and window sizes for each window of a model year. Specifically, the data items collected from MY 2008 vehicles with ages between 0 to 271 days are placed in dataset D_1, and similarly the data items with age between 272 to 542 days are placed in dataset D_2 and so on. Therefore, for MY 2008, the patterns/cases in dataset D_1 have least age, whereas the cases corresponding to highest age belong to D_4. In the same way, for MY 2009 cases, dataset D_5 has the least age whereas D_8 has the highest age. Another important aspect is that no two datasets have all the fault codes the same. MY 2008 cases have 26 distinct fault classes and 4 of these fault classes

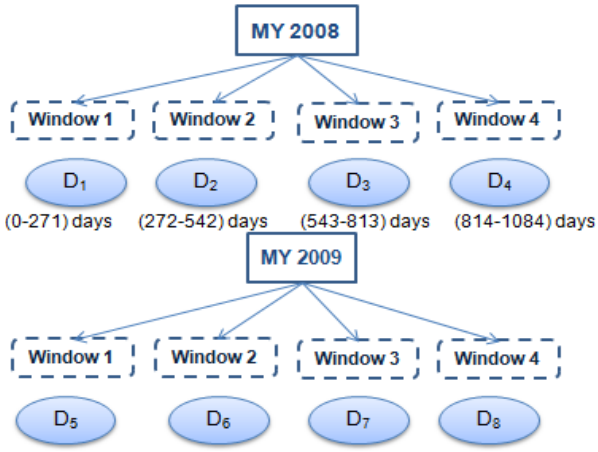

FIGURE 4. Data Partition for Window Method.

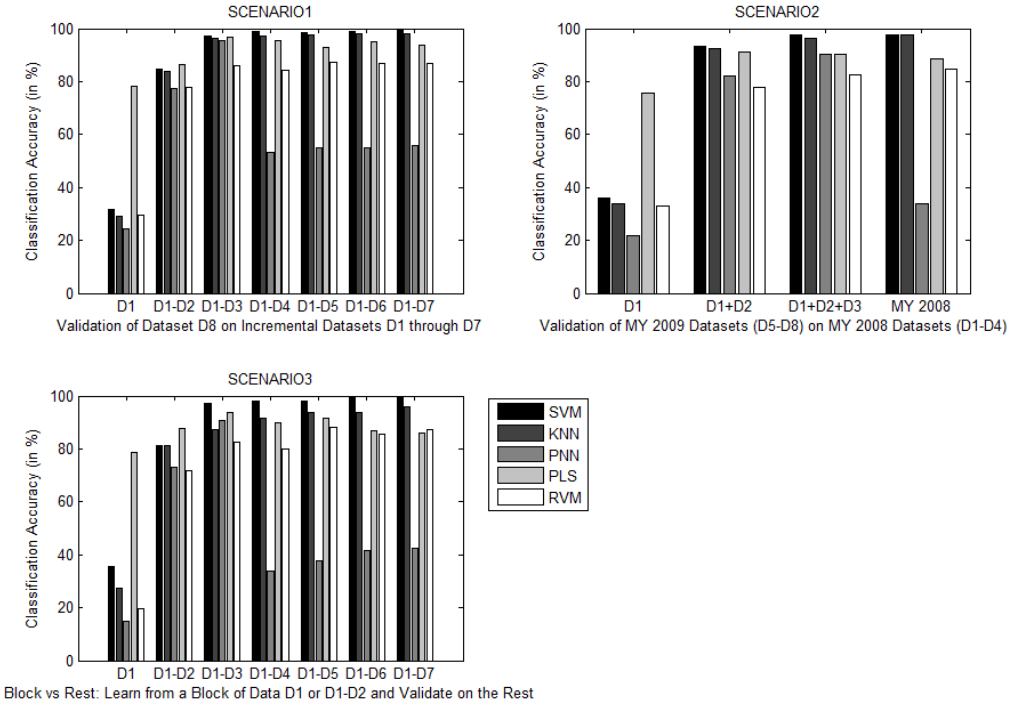

FIGURE 5. Ensemble Classifier Performance for Scenarios 1–3.

are different from MY 2009 cases; similarly, MY 2009 has 23 fault classes with 1 fault class different from that of MY 2008 cases. Table 1 shows the data distribution (Refer to Appendix for fault class description).

TABLE 1. Data distribution.

Model Year	Dataset	Number of Patterns	Number of Classes
2008	D_1	98	14
	D_2	310	22
	D_3	915	24
	D_4	609	21
2009	D_5	97	14
	D_6	221	14
	D_7	229	20
	D_8	108	14

Once the datasets D_1-D_8 are available, the adaptive methodology is validated using three experimental runs. The experimental setup consisted of different classifiers in a Learn++.NC framework. First run is performed by training the ensemble classifier using MY 2008 datasets alone. Here classifiers are incrementally trained from datasets D_1, D_2, D_3; and the classifier performance is validated using dataset D_4. Similarly, a second run is made by training the classifiers with MY 2009 datasets D_5, D_6 and D_7 incrementally and the classifier performance is tested using dataset D_8. Finally, in the third run, the classifiers are trained incrementally with datasets D_1 through D_7 and tested on D_8.

The three experimental runs are summarized below:

(i) Train on $\{D_1, D_2, D_3\}$ incrementally and test on D_4 for MY 2008

(ii) Train on $\{D_5, D_6, D_7\}$ incrementally and test on D_8 for MY 2009

(iii) Train on $\{D_1, D_2, \ldots, D_7\}$ incrementally and test on D_8 for combined MY 2008 and MY 2009 datasets.

The top left corner of Fig. 5 shows the performance of different classifiers on testing dataset D_8 by learning incrementally from datasets $D1$ through D_7 (i.e., run (*iii*)). Here, classifiers SVM, KNN and PNN achieved diagnostic accuracy of approximately 85% when they are trained with at least two datasets. Except for PLS, all other classifiers performed poorly when D_8 is validated on D_1. However, SVM and other classifiers (KNN, PLS, RVM) performed better when D_8 is validated on other datasets after incremental learning. The classification results of SVM for different experimental runs are provided in Table 2 (Note that SVM consistently performed well in Scenarios 1 to 4 and hence only SVM results will be provided in detail from now on for different experimental runs. More details on classification techniques and their performance on other real world applications and synthetic datasets can be found in some of our previous work [40]–[42]). It can be seen that the classification performance improved after incrementally learning with each of the datasets when compared to learning from D_1 alone. Initially, the evaluation of testing dataset D_8 (MY 2009 cases with highest age) on D_1 (MY 2008 cases with least age) was only 31.6% (run (*iii*) in Table 2), but with incremental learning of new information with new datasets,

TABLE 2. Classification performance of incremental ensemble of SVM classifiers – scenario1.

Experimental Run	Classification Accuracy (CA in %)						
(i) MY 2008 alone	D_4 *on* D_1 54.7			D_4 *on* D_1 *and* D_2 84.6		D_4 *on* D_1, D_2 *and* D_3 97.6	
(ii) MY 2009 alone	D_8 *on* D_5 88.9			D_8 *on* D_5 *and* D_6 95.3		D_8 *on* D_5, D_6 & D_7 98.6	
(iii) Combined (MY 2008 and MY 2009)	D_8 *on* D_1 31.6	D_8 *on* D_1 -D_2 84.5	D_8 *on* D_1 - D_3 97.1	D_8 *on* D_1 - D_4 98.9	D_8 *on* D_1 - D_5 98.7	D_8 *on* D_1 - D_6 98.9	D_8 *on* D_1 -D_7 99.2

the classification accuracy increased from 31.6% to 99.2%. In the case of MY 2008 data alone, the classification performance on D_4 increased from 54.7% to 97.6% with classifiers incrementally learning new information with new datasets (run (*i*) in Table 2). Similarly, for MY 2009 data alone, the classification accuracy of D_8 increased from 88.9% to 98.6% (run (*ii*) in Table 2).

The results also showed that the classifiers built for each model year when trained with at least two datasets from the same model year could achieve classification accuracy of approximately 84%. For instance, when the classifier model is trained with datasets D_1 and D_2 from MY 2008, the validation on D_4, produced a classification accuracy of about 85% (see experimental run (*i*) in Table 2). Similarly, validating D_8 on D_5 for MY 2009 alone resulted in a classification accuracy of about 89% (see experimental run (*ii*) in Table 2). Also, at least two datasets from MY 2008 (D_1 and D_2) are required to estimate the fault classes in the last dataset of MY 2009 (D_8) with an accuracy of 84% (experimental run (*iii*) in Table 2). In other words, classifiers learned with at least two datasets from a previous model year (i.e., cases corresponding to an age of at least 542 days) could be used for fault diagnosis in subsequent model year vehicles.

Scenario 2 – Block Method: MY 2008 and MY 2009 data with the same features is arranged in sequential order w.r.t. age. The experimentation is carried out by training the classification algorithms with blocks of data from MY 2008, and testing the algorithms with the corresponding blocks of data from MY 2009 (see Fig. 6). In this scenario, the data distribution and the data partition is similar to the previous case (see Table 1 for data distribution). Here, the classification model is built using MY 2008 datasets i.e., $\{\{D_1, \ldots, D_i\}: i = 1, 2, 3, 4\}$, and validated using datasets from MY 2009 $\{\{D_5, D_6, \ldots, D_j\}: j = 5, 6, 7, 8\}$ as shown in Fig. 6. In this scenario, similar to the previous scenario, the performance of classifiers, except for PLS classifier, is poor when trained with dataset D_1 alone, and the diagnostic accuracy improved significantly when trained with subsequent datasets (see top right figure of Fig. 5). Table 3 shows the SVM classifier performance in the *Block method*. The classification performance increased from 36.1% to 97.7% with increase in the number of datasets. This again shows that in order to use

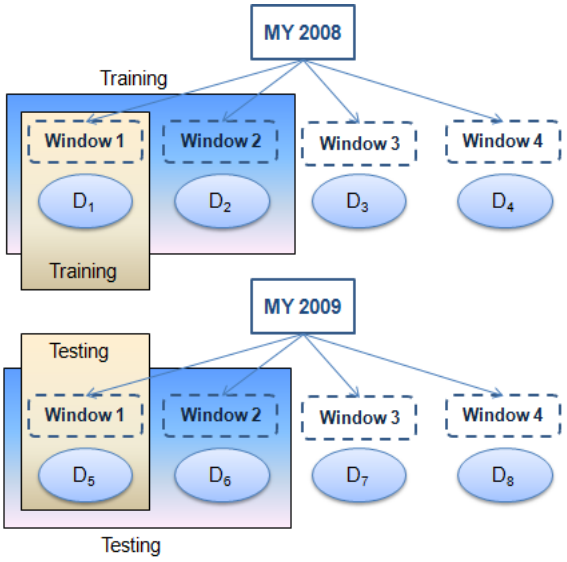

FIGURE 6. Block Method.

the classifiers built on previous MY data, the classification techniques should be trained with at least two datasets (approx. 542 days for MY 2008) to achieve a classification accuracy of 93.4% in the subsequent year (i.e., MY 2009).

TABLE 3. Classification performance of incremental ensemble of SVM classifier – scenario2.

Classification Accuracy (in %)	
D_5 *on* D_1	36.08
D_5+D_6 *on* D_1+D_2	93.4
D_5+D_6+D_7 *on* D_1+D_2+D_3	97.42
MY 2009 on MY 2008	97.69

Scenario 3 – Block versus Rest: MY 2008 and MY 2009 data with the same features is arranged in sequential order w.r.t. age. We call this the '*Block versus Rest*' because we are training on a block of data and testing on the rest of the data. In the *Block vs Rest* method, the partitioning of

the data is similar to Scenario 1 (see Fig. 4) except that the training of the classification algorithms is done using data $\{\{D_1, D_2, \ldots, D_i\}: i = 1, 2, \ldots, 7\}$ and tested using the remaining datasets $\{D_{i+1}, D_{i+2}, \ldots, D_8\}$. Here, when the classifiers are learned using D_1 alone and tested on datasets from D_2 to D_8, the diagnostic accuracy was very poor, but it gradually improved to 97.1% when trained with datasets D_1, D_2, and D_3, and tested on the rest of the datasets ($\{D_4, D_5, \ldots, D_8\}$). This again demonstrates that, in order to use the classification model built with MY 2008 cases for the next model year, the classification techniques need to be trained with at least two datasets in the increasing order of age to achieve a diagnostic accuracy of 81% (see Table 4). The performance of different classifiers is provided in Fig. 5 (bottom left corner). Once again, SVM performed better than the other classifiers compared here.

TABLE 4. Classification performance of incremental ensemble of SVM classifiers – scenario3.

Experimental Runs	Classification Accuracy (CA) in %
CA of datasets $\{D_2, D_3, \ldots, D_8\}$ on model from D_1	35.47
CA of datasets $\{D_3, D_4, \ldots, D_8\}$ on model on $\{D_1, D_2\}$	81.26
CA of datasets $\{D_4, D_5, \ldots, D_8\}$ on model from $\{D_1, D_2, D_3\}$	97.08
CA of datasets $\{D_5, D_6, \ldots, D_8\}$ on model from $\{D_1, D_2, \ldots, D_4\}$	97.91
CA of datasets $\{D_6, \ldots, D_8\}$ on model from $\{D_1, D_2, \ldots D_5\}$	98.02
CA of datasets $\{D_7, D_8\}$ on model from $\{D_1, D_2, \ldots, D_6\}$	99.41
CA of dataset $\{D_8\}$ on model from $\{D_1, D_2, \ldots, D_7\}$	99.19

Scenario 4 – Block versus One: MY 2008 and MY 2009 data with the same features is arranged in sequential order w.r.t. age. We call this the 'Block versus One' because, here, we are training on a block of data from MY 2008 and/or 2009 and testing on each of the remaining datasets. In this scenario, the data partitioning is similar to Scenario 1 (Fig. 4) but here the classifiers are trained using data $\{\{D_1, D_2, \ldots, D_i\}: i = 1, 2, \ldots, 7\}$ and tested on $\{\{D_j\}: j = i+1, i+2, \ldots, 8\}$ individually. Here, the classifiers need to be trained with at least two datasets to achieve a diagnostic accuracy of approximately 80% on the subsequent datasets. When the classifiers are learned with only one dataset, i.e., D_1 alone, the classification performance on D_2 was 72% whereas the performance on all the other subsequent datasets is very poor (approx. 35%). This is in conformance with the assertion on other scenarios that at least two windows of datasets are required for the training of classifiers in order to achieve good diagnostic accuracy on subsequent datasets. Table 5 summarizes the classification performance.

Thus, the different age-based scenarios presented here for incremental learning of classifiers consistently showed

TABLE 5. Classification performance of incremental ensemble of SVM classifiers – scenario4.

Training Dataset	Testing Dataset						
D_1	D_2 71.8	D_3 60.7	D_4 53.5	D_5 45.9	D_6 34.76	D_7 34.3	D_8 33.6
D_1, D_2	D_3 85.9	D_4 83.2	D_5 83.4	D_6 84.7	D_7 85.4	D_8 85.7	
D_1, D_2, D_3	D_4 98.7	D_5 98.5	D_6 98.5	D_7 98.4	D_8 98.3		
D_1, D_2, D_3, D_4	D_5 99.4	D_6 99.5	D_7 99.2	D_8 99.2			
D_1, D_2, D_3, D_4, D_5	D_6 99.9	D_7 99.4	D_8 99.3				
$D_1, D_2, D_3, D_4, D_5, D_6$	D_7 99.6	D_8 99.6					
$D_1, D_2, D_3, D_4, D_5, D_6, D_7$	D_8 98.7						

improved performance with new datasets. Also, it is evident from the presented scenarios that, partitioning the data based on window size (as in Scenarios 1 through 4) and training the diagnosis algorithm using the approach described in either Scenario1 or Scenario 4 enables us to use least amount of data and yet achieve good diagnostic accuracy (Scenario4: >83%; and Scenario 1: 84%) on the subsequent datasets.

V. CONCLUSION

In this paper, we briefly discussed the incremental learning techniques and validated the performance of an ensemble of classifiers on an automotive system via four incremental learning scenarios. The different scenarios presented here enable us to train the classifiers incrementally with one model year (e.g., MY 2008) data and use the resulting classifiers to diagnose faults in the subsequent model year vehicles (e.g., MY 2009). In all of the scenarios, a variety of statistical and pattern classification techniques, such as support vector machines, k-nearest neighbor, partial least squares, probabilistic neural network and relevance vector machine, are employed as the base classifiers in an ensemble learning framework. The performance of the classifiers improved when they are incrementally trained with new datasets. The increase in accuracy shows that the classifiers are able to learn the behavioral trends captured in the new data. However, among all the classifiers, SVM consistently performed well in all the scenarios 1-4. Another empirical finding is that if we need to achieve a diagnostic accuracy of approximately 90% on data from a subsequent model year, the classifier must be trained *with at least two of the datasets* (each with a window size of approx. 271 days) from the prior model year. The scenarios presented also provide insights to automotive diagnostic engineers to determine whether an existing diagnosis algorithm is efficient across all

MY vehicles; and if needed, allows incremental learning of the classifier ensemble. The approach is suitable for a wide-range of real-world applications where a large amount of data is anticipated to evolve with time and accordingly continuous learning is needed. In the future, we plan to investigate the incremental learning approach with respect to the evolving data with new features.

APPENDIX
LIST OF FAULT CODES (OR) FAULT CLASSES

Fault Code	Fault Code Description
P0068	MAP/MAF - Throttle Position Correlation (F1)
P0101	Mass or Volume Air Flow Circuit Range/Performance (F2)
P0102	Mass or Volume Air Flow Sensor Circuit Low Input (F3)
P0121	Throttle Position Sensor A Circuit Performance (F4)
P0122	Throttle/Pedal Position Sensor/Switch A Circuit Low (F5)
P0123	Throttle/Pedal Position Sensor/Switch A Circuit High (F6)
P0222	Throttle/Pedal Position Sensor/Switch B Circuit Low (F7)
P0223	Throttle/Pedal Position Sensor/Switch B Circuit High (F8)
P0562	System Voltage Low (F9)
P0563	System Voltage High (F10)
P0601	Internal Control Module Memory Checksum Error (F11)
P0604	Internal Control Module RAM Error (F12)
P0606	Control Module Processor (F13)
P062F	Internal Control Module EEPROM Error (F14)
P1516	Throttle Control Module Actuator Performance (F15)
P1631	Theft Deterrent Fuel Enable Signal Not Correct (F16)
P1682	Ignition 1 Switch Circuit 2 (F17)
P2101	Throttle Control Motor Circuit Performance (F18)
P2119	Throttle Control Throttle Body Performance (F19)
P2122	Throttle/Pedal Position Sensor D Circuit Low (F20)
P2123	Throttle/Pedal Position Sensor D Circuit High (F21)
P2127	Throttle/Pedal Position Sensor E Circuit Low (F22)
P2128	Throttle/Pedal Position Sensor E Circuit High (F23)
P2135	Throttle Position Sensor A/B Voltage Correlation (F24)
P2138	Throttle/ Position Sensor D/E Voltage Correlation (F25)
P2176	Throttle Control Sys.Idle Position Not Learned (F26)
P2610	ECM/PCM Engine Off Timer Performance (F27)

ACKNOWLEDGMENT

Any opinions expressed in this paper are solely those of the authors and do not represent those of the sponsors.

REFERENCES

[1] K. Pattipati *et al.*, "An integrated diagnostic process for automotive systems," in *Computational Intelligence in Automotive Applications* (Studies in Computational Intelligence), vol. 132. Berlin, Germany: Springer-Verlag, 2008, pp. 191–218.
[2] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. New York, NY, USA: Wiley, 2001.
[3] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer-Verlag, 2006.
[4] G. Ciccarella, M. D. Mora, and A. Germani, "A Luenberger-like observer for nonlinear systems," *Int. J. Control*, vol. 57, no. 3, pp. 537–556, 1993.
[5] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation With Applications to Tracking and Navigation*. New York, NY, USA: Wiley, 2001.
[6] J. Gertler, "Fault detection and isolation using parity relations," *Control Eng. Pract.*, vol. 5, no. 5, pp. 653–661, 1997.
[7] J. Luo, H. Tu, K. Pattipati, L. Qiao, and S. Chigusa, "Diagnosis knowledge representation and inference," *IEEE Instrum. Meas. Mag.*, vol. 9, no. 4, pp. 45–52, Aug. 2006.
[8] C. Sankavaram, A. Kodali, D. F. M. Ayala, K. Pattipati, S. Singh, and P. Bandyopadhyay, "Event-driven data mining techniques for automotive fault diagnosis," in *Proc. 21st Int. Workshop Principles Diag.*, Portland, OR, USA, Oct. 2010, pp. 1–8.
[9] C. Sankavaram *et al.*, "Model-based and data-driven prognosis of automotive and electronic systems," in *Proc. 5th Annu. IEEE Conf. Autom. Sci. Eng.*, Bangalore, India, Aug. 2009, pp. 96–101.
[10] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Mach. Learn.*, vol. 23, no. 1, pp. 69–101, 1996.
[11] K. Yamauchi, N. Yamaguchi, and N. Ishii, "Incremental learning methods with retrieving of interfered patterns," *IEEE Trans. Neural Netw.*, vol. 10, no. 6, pp. 1351–1365, Nov. 1999.
[12] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.
[13] T. Yoneda, M. Yamanaka, and Y. Kakazu, "Study on optimization of grinding conditions using neural networks—A method of additional learning," *J. Jpn. Soc. Precis. Eng.*, vol. 58, no. 10, pp. 1707–1712, Oct. 1992.
[14] N. A. Syed, S. Huan, L. Kah, and K. Sung, "Incremental learning with support vector machines," in *Proc. IJCAI*, San Diego, CA, USA, 1999.
[15] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," in *Advances in Neural Information Processing Systems*, vol. 13. Cambridge, MA, USA: MIT Press, 2001.
[16] M. Karasuyama and I. Takeuchi, "Multiple incremental decremental learning of support vector machines," *IEEE Trans. Neural Netw.*, vol. 21, no. 7, pp. 1048–1059, Jul. 2010.
[17] Y.-M. Wen and B.-L. Lu, "Incremental learning of support vector machines by classifier combining," in *Proc. 11th Pacific-Asia Conf. Adv. Knowl. Discovery Data Mining*, 2007, pp. 904–911.
[18] Z. Erdem, R. Polikar, F. Gurgen, and N. Yumusak, "Ensemble of SVMs for incremental learning," in *Multiple Classifier Systems* (Lecture Notes in Computer Science). Berlin, Germany: Springer-Verlag, 2005.
[19] N. Bhattacharyya, A. Metla, R. Bandyopadhyay, B. Tudu, and A. Jana, "Incremental PNN classifier for a versatile electronic nose," in *Proc. 3rd Int. Conf. Sens. Technol.*, Tainan, Taiwan, Nov./Dec. 2008, pp. 242–247.
[20] P. M. Ciarelli, E. Oliveira, and E. O. T. Salles, "An evolving system based on probabilistic neural network," in *Proc. 11th Brazilian Symp. Neural Netw.*, Oct. 2010, pp. 182–187.
[21] P. M. Ciarelli, E. Oliveira, and E. O. T. Salles, "An incremental neural network with a reduced architecture," *Neural Netw.*, vol. 35, pp. 70–81, Nov. 2012.
[22] D. Skocaj, M. Uray, A. Leonardis, and H. Bischof, "Why to combine reconstructive and discriminative information for incremental subspace learning," in *Proc. Comput. Vis. Winter Workshop*, 2006, pp. 1–6.
[23] L. Hartert and M. Sayed-Mouchaweh, "Dynamic supervised classification method for online monitoring in non-stationary environments," *Neurocomputing*, vol. 126, pp. 118–131, Feb. 2014.
[24] D. Kalles and T. Morris, "Efficient incremental induction of decision trees," in *Machine Learning*. Boston, MA, USA: Kluwer, 1996, pp. 231–242.
[25] A. S. Al-Hegami, "Classical and incremental classification in data mining process," *Int. J. Comput. Sci. Netw. Security*, vol. 7, no. 12, pp. 179–187, 2007.
[26] J. R. Alcobe, "Incremental augmented Naïve Bayes classifiers," in *Proc. 16th Eur. Conf. Artif. Intell.*, 2004.
[27] J. R. Alcobe, "Incremental learning of tree augmented Naïve Bayes classifiers," in *Proc. 8th Ibero-Amer. Conf. Artif. Intell.*, 2002, pp. 32–41.
[28] L. Jiang, Z. Cai, D. Wang, and H. Zhang, "Improving tree augmented Naïve Bayes for class probability estimation," *Knowl.-Based Syst.*, vol. 26, pp. 239–245, Feb. 2012.
[29] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, "Learn^{++}: An incremental learning algorithm for supervised neural networks," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 31, no. 4, pp. 497–508, Nov. 2001.

[30] P. Baraldi, R. Razavi-Far, and E. Zio, "Classifier-ensemble incremental-learning procedure for nuclear transient identification at different operational conditions," *Rel. Eng. Syst. Safety*, vol. 96, no. 4, pp. 480–488, Apr. 2011.
[31] M. D. Muhlbaier, A. Topalis, and R. Polikar, "Learn^{++}.NC: Combining ensemble of classifiers with dynamically weighted consult-and-vote for efficient incremental learning of new classes," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 152–168, Jan. 2009.
[32] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Mach. Learn.*, vol. 36, nos. 1–2, pp. 105–139, 1999.
[33] L. Breiman, "Bagging predictors," Dept. Statist., Univ. California, Berkeley, CA, USA, Tech. Rep. 421, 1994.
[34] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining Knowl. Discovery*, vol. 2, no. 2, pp. 121–167, 1998.
[35] J. E. Jackson, *A User's Guide to Principal Components*. New York, NY, USA: Wiley, 1991.
[36] P. Geladi and B. R. Kowalski, "Partial least-squares regression: A tutorial," *Anal. Chem. Acta*, vol. 185, pp. 1–17, 1986.
[37] M. E. Tipping, "Sparse Bayesian learning and the relevance vector machine," *J. Mach. Learn. Res.*, vol. 1, pp. 211–244, Sep. 2001.
[38] *Electronic Throttle Control*. [Online]. Available: http://www.picoauto.com/applications/electronic-throttle-control.html, accessed Sep. 14, 2014.
[39] R. Battiti, "Using mutual information for selecting features in supervised neural net learning," *IEEE Trans. Neural Netw.*, vol. 5, no. 4, pp. 537–550, Jul. 1994.
[40] K. Choi *et al.*, "Novel classifier fusion approaches for fault diagnosis in automotive systems," *IEEE Trans. Instrum. Meas.*, vol. 58, no. 3, pp. 602–611, Mar. 2009.
[41] W. Donat, K. Choi, W. An, S. Singh, and K. Pattipati, "Data visualization, data reduction and classifier fusion for intelligent fault diagnosis in gas turbine engines," *J. Eng. Gas Turbines Power*, vol. 130, no. 4, p. 041602, Apr. 2008.
[42] C. Sankavaram, B. Pattipati, K. R. Pattipati, Y. Zhang, and M. Howell, "Fault diagnosis in hybrid electric vehicle regenerative braking system," *IEEE Access*, vol. 2, pp. 1225–1239, Oct. 2014.

CHAITANYA SANKAVARAM (M'11) received the B.Tech. degree in electrical and electronics engineering from Sri Venkateswara University, Tirupathi, India, in 2005. She received the M.S. degree from the University of Connecticut, Storrs, CT, USA, where she is currently pursuing the Ph.D. degree in electrical and computer engineering. She was a Project Engineer with Wipro Technologies, Bangalore, India. She has been a Researcher with the Vehicle Systems Research Laboratory, General Motors Global Research and Development Center, Warren, MI, since 2013. Her work involves research and development of diagnostics, prognostics, and health management solutions for automotive systems. Her research interests include fault diagnosis and prognosis, machine learning, data mining, pattern recognition, reliability analysis, and optimization
theory.

ANURADHA KODALI received the B.E. degree in electronics and communications engineering from Andhra University, Visakhapatnam, India, in 2006, and the Ph.D. degree in electrical and computer engineering from the University of Connecticut, Storrs, in 2012. She is currently with the NASA Ames Research Center. Her research interests include data mining, pattern recognition, fault detection and diagnosis, and optimization theory. She served as a Reviewer in several IEEE journals.

KRISHNA R. PATTIPATI (S'77–M'80–SM'91–F'95) received the B.Tech. (Hons.) degree in electrical engineering from IIT Kharagpur, in 1975, and the M.S. and Ph.D. degrees in systems engineering from the University of Connecticut (UCONN), Storrs, in 1977 and 1980, respectively. He was with ALPHATECH, Inc., Burlington, MA, from 1980 to 1986. He has been with the Department of Electrical and Computer Engineering, UCONN, where he is currently the UTC Professor of Systems Engineering and serves as the Interim Director of the UTC Institute for Advanced Systems Engineering. His current research activities are in the areas of agile planning, diagnosis and prognosis techniques for cyber-physical systems, multiobject tracking, and combinatorial optimization. A common theme among these applications is that they are characterized by a great deal of uncertainty, complexity, and computational intractability. He is the Co-Founder of Qualtech Systems, Inc., a firm specializing in advanced integrated diagnostics software tools (TEAMS, TEAMS-RT, TEAMS-RDS, and TEAMATE), and serves on the Board of Aptima, Inc.

Dr. Pattipati is an Elected Fellow of the Connecticut Academy of Science and Engineering. He was a co-recipient of the Andrew P. Sage Award for the Best IEEE Systems, Man, and Cybernetics (SMC) Transactions Paper in 1999, the Barry Carlton Award for the Best AES Transactions Paper in 2000, the NASA Space Act Awards for A Comprehensive Toolset for Model-Based Health Monitoring and Diagnosis, and Real-Time Update of Fault-Test Dependencies of Dynamic Systems: A Comprehensive Toolset for Model-Based Health Monitoring and Diagnostics in 2002 and 2008, and the AAUP Research Excellence Award at UCONN in 2003. He received the best technical paper awards at the 1985, 1990, 1994, 2002, 2004, 2005, and 2011 IEEE AUTOTEST Conferences, and the 1997 and 2004 Command and Control Conference. He was selected as the Outstanding Young Engineer by the SMC Society in 1984, and received the Centennial Key to the Future Award. He served as the Editor-in-Chief of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS–PART B from 1998 to 2001, the Vice President of Technical Activities of the IEEE SMC Society from 1998 to 1999, and the Vice President of Conferences and Meetings of the IEEE SMC Society from 2000 to 2001.

SATNAM SINGH (S'00–M'08–SM'13) received the B.E. degree from IIT Roorkee, the master's degree in electrical and computer engineering from the University of Wyoming, and the Ph.D. degree in electrical and computer engineering from the University of Connecticut. To his credit, he holds eight granted patents, 15 patent applications, and has authored 32 journal and conference publications. He is a speaker in various Big Data and Data Science Conferences.

He spent nearly five years with General Motors Research as a Senior Researcher, with responsibility for conceptualizing and developing data products from scratch for several verticals.

As a Data Geek, Researcher, and Business and Team Builder, he has a 10-year track record of successfully building data products from concept to production. As a Data Scientist and Team Leader with Samsung Research India, Bangalore, he built data science team and developed several analytics features in smartphone. He is currently a Data Scientist with CA Technologies.