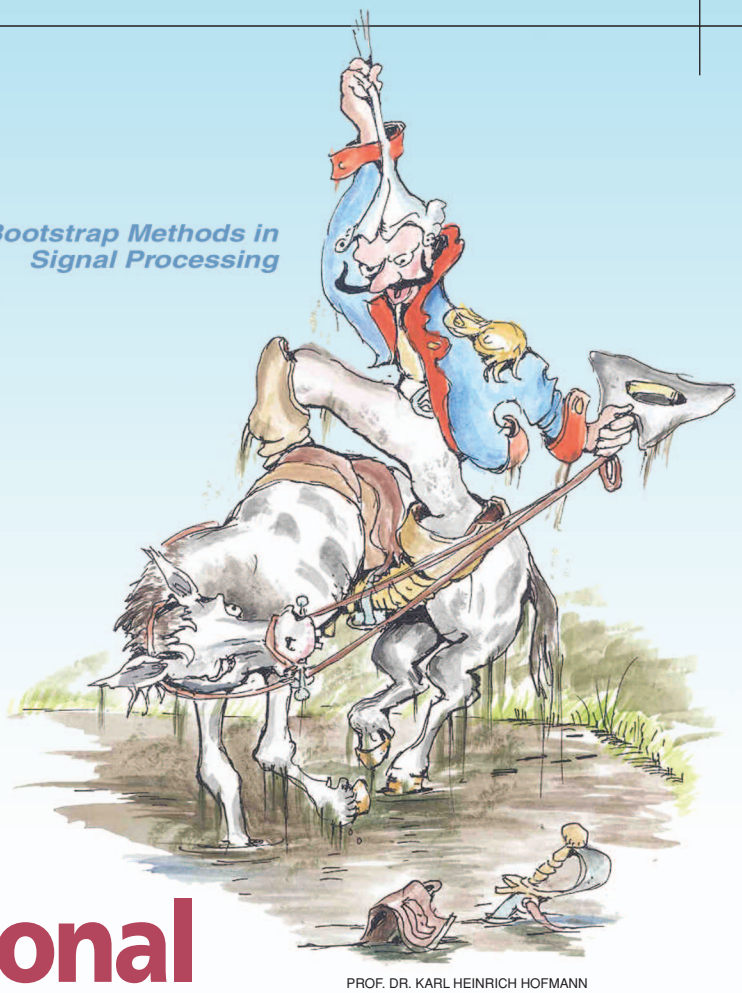
Robi Polikar

Bootstrap Methods in Signal Processing



PROF. DR. KARL HEINRICH HOFMANN

Bootstrap-Inspired Techniques in Computational Intelligence

Ensemble of classifiers for incremental learning, data fusion, and missing feature analysis

This article is about the success story of a seemingly simple yet extremely powerful approach that has recently reached a celebrity status in statistical and engineering sciences. The hero of this story—bootstrap resampling—is relatively young, but the story itself is a familiar one within the scientific community: a mathematician or a statistician conceives and formulates a theory that is first developed by fellow mathematicians and then brought to fame by other professionals, typically engineers, who point to many applications that can benefit from just such an approach. Signal processing boasts some of the finest examples of such stories, such as the classic story of Fourier transforms or the more contemporary tale of wavelet transforms.

Originally developed for estimating sampling distributions of statistical estimators from limited data, bootstrap techniques have since found applications in many areas of engineering—including signal processing—several examples of which appear elsewhere in this issue. This

1053-5888/07/$25.00©2007IEEE

article, however, is about bootstrap-inspired techniques in computational intelligence; specifically in ensemble of classifiers-based algorithms, which themselves have many applications in signal processing.

We start with a brief review of the bootstrap-based approaches used in computational intelligence, where the primary parameter of interest is the true prediction error of a classifier on previously unseen data. We then describe how bootstrap-inspired techniques enabled development of ensemble-based algorithms and describe some of the more popular examples of such algorithms. These algorithms, which generate an ensemble of classifiers by training each classifier on a different bootstrap sample of the training data, have the unique ability to create a strong classifier from a collection of weak classifiers that can barely do better than random guessing.

Our primary focus in this article, however, is some of the more challenging problems of computational intelligence that can also be addressed by bootstrap-inspired techniques, including incremental learning, data fusion, and the missing feature problem. In incremental learning, the goal is to learn novel and supplementary information from new data that later become available, even in such hostile learning environments that introduce new classes. In data fusion, we are interested in integrating complementary information into an existing classifier's knowledge base, particularly when such information comes from different sources. Finally, in the missing feature problem, the challenge is to classify data whose certain features (predictors) used to train the classifier are missing.

The central theme in addressing each of these problems using ensemble-based systems is to generate an ensemble of diverse classifiers, where each classifier is trained on a strategically selected bootstrap sample of the original data. We discuss the ability of bootstrap-based approaches to address these issues and present the outcomes of implementations of such approaches on a variety of real-world problems.

BOOTSTRAP TECHNIQUES IN COMPUTATIONAL INTELLIGENCE

Bootstrap techniques are concerned with the following fundamental problem: "if we make an estimate about a parameter of a population of unknown distribution based on a single dataset of finite size, how good is this estimate?" Precise formulas to determine the goodness of such an estimate are well established in statistics; however, these formulas make certain assumptions about the underlying distributions that are often violated in real-world applications. Even then, most formulations calculate goodness of the estimate only when the parameter of interest is the sample mean.

In 1979, Efron proposed a solution that only became practically feasible by the recent availability of computing power. His approach was simple and elegant, if not controversial at first: treat the available dataset as if it were the entire population, and take repeated samples from this (pseudo) distribution [1]. He called each such sample a bootstrap sample, from which the statistic of interest is estimated. Repeating this process many times, we can simulate having many samples from the original distribution and hence calculate a meaningful confidence interval of the estimate. Furthermore, such an estimate can be obtained for any statistic of interest, and not just for the mean.

In signal processing, bootstrap methods are widely used for signal detection and spectral estimation, details and many applications of which can be found elsewhere in this issue as well as in Zoubir and Iskander's recent text [2]. In computational intelligence, however, the statistic of particular interest is often the true generalization error of a classifier that is trained on a finite-size training dataset. Given such a classifier, how can we estimate its performance in classifying previously unseen field data? To answer this question, let us formally define the classification problem.

In supervised classification, we are given a training dataset $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}_i \in X$ is the ith instance in the feature space X, along with their correct labels $y_i \in \Omega$, $\Omega = \{\omega_1, \omega_2, \ldots, \omega_C\}$ for a C-class problem. S is drawn from a fixed but unknown distribution D of labeled instances. The true class for each instance is determined by a fixed mapping function $g: X \to \Omega$, also unknown to us. A classifier is trained on S, producing a hypothesis h, whose goal is to satisfy $h(\mathbf{x}) = g(\mathbf{x}), \forall \mathbf{x} \in X$. However, we usually end up with a hypothesis that can only provide $u = h(\mathbf{x}) = \hat{g}(\mathbf{x})$, $u \in \Omega$, as an estimate of $g(x)$. In order to evaluate the goodness of this estimate, we define the "classification performance evaluation function" $Q(h(\mathbf{x})) = I[\![y \neq u]\!]$, ($Q(\mathbf{x})$ in short), where $I[\![\bullet]\!]$ is the indicator function. The true error is the probability that h will incorrectly classify instance $\mathbf{x}$ drawn from D, i.e., $\mathrm{Err} = P(h(\mathbf{x}) \neq y) = E_D[Q(\mathbf{x})]$, where $E_D[.]$ indicates expectation with respect to D. We can now define several estimates of this true error Err.

In *resubstitution (empirical) error,* the classifier is evaluated on the training data S:

$$\widehat{\mathrm{Err}}_R = \frac{1}{n}\sum_{i=1}^{n} Q(\mathbf{x}_i), \quad \mathbf{x} \in S. \tag{1}$$

Since the same dataset S is used both for training and testing, $\widehat{\mathrm{Err}}_R$ is usually optimistic, underestimating the true error. The difference between the two is the bias of the estimate, whereas the variation of this estimate among several trials is its variance. We wish to minimize both, which may often be contradictory to each other. In a *hold-out estimate*, S is partitioned into two subsets, S_{TR} and S_{TE}, for training and testing, respectively. The classifier is trained on S_{TR} and evaluated on S_{TE}:

$$\widehat{\mathrm{Err}}_H = \frac{1}{n_{TE}}\sum_{i=1}^{n_{TE}} Q(\mathbf{x}_i), \quad \mathbf{x} \in S_{TE}, \tag{2}$$

where n_{TE} is the cardinality of S_{TE}. There are two problems with this approach, however: a suboptimal classifier is obtained by using fewer training data points, and error estimate depends on the exact split of S into training and test subsets. Both issues can be avoided by the *cross-validation (or jackknife) estimate*,

where the dataset S is divided into K-blocks, each of size n/K. Concatenating the first $K-1$ blocks produces the first training subset S_{TR}^1. The remaining Kth block, denoted S_{TE}^K, then serves as the test subset, on which the first error estimate is obtained by using (2). This process is repeated K times, leaving a different block out in each case. The K-fold cross validation (CV) error estimate is then the average of K individual error estimates:

$$\widehat{\mathrm{Err}_{CV}} = \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{n_{TE}^k}\sum_{i=1}^{n_{TE}^k} Q\left(\mathbf{x}_i^k\right)\right), \quad \mathbf{x}^k \in S_{TE}^k. \tag{3}$$

A special case of the K-fold CV, called *leave-one-out* (LOO), is obtained by choosing $K=n$, where S_{TR} consists of all of S except a single instance $\mathbf{x}$, which then serves as the test set. Let $S_{TE}^{(i)}$ be the test point left out during ith training session. Equation (3) then reduces to

$$\widehat{\mathrm{Err}_{\mathrm{LOO}}} = \frac{1}{n}\sum_{i=1}^{n} Q(\mathbf{x}_i), \quad \mathbf{x}_i \in S_{TE}^{(i)}. \tag{4}$$

The LOO estimate is an unbiased estimate of the true error; however, it suffers from high variance due to individual estimates being calculated on single $\mathbf{x}_i$. In the *bootstrap estimate*, specifically designed to reduce this variance, S is treated as the entire population, with a distribution equal to $1/n$—per occurrence—for each instance $\mathbf{x}_i$ (as discussed below, incremental learning, data fusion, and missing feature can all be addressed by modifying this very distribution). Sample subsets of size n are then drawn from this distribution with replacement. Let each such dataset be S_{TR}^{*b}, where $^*b$ indicates the bth bootstrap sample. Certain instances of S will appear multiple times in S_{TR}^{*b}, whereas others will not appear at all. Those $\mathbf{x}_i$ not included in S_{TR}^{*b} constitute the test set, S_{TE}^{*b} of cardinality n_{TE}^b ,on which the bootstrap error estimate ε_b is computed. Repeating this process B times gives B such error estimates. The final error estimate is then the mean of the individual bootstrap errors [3]:

$$\widehat{\mathrm{Err}^*} = \frac{1}{B}\sum_{b=1}^{B}\varepsilon_b = \frac{1}{B}\sum_{b=1}^{B}\left(\frac{1}{n_{Te}^b}\sum_{i=1}^{n_{Te}^b} Q(\mathbf{x}_i^b)\right), \quad \mathbf{x}^b \in S_{TE}^{*b}. \tag{5}$$

Choosing a sufficiently large B, the large variance among individual estimates can be reduced thanks to averaging [3]. Several improvements have since been made to this estimate such as the 0.632 estimator, which recognizes that the probability of any given $\mathbf{x}_i$ to appear in a sample is $1-(1-1/n)^n \approx 0.632$ for large n. The 0.632 estimator calculates the error as the weighted average of the bootstrap and resubstitution errors, using weights 0.632 and 0.368, respectively:

$$\widehat{\mathrm{Err}_{.632}} = \frac{1}{B}\sum_{b=1}^{B}\left(0.632\cdot\varepsilon_b + 0.368\widehat{\mathrm{Err}_R}\right). \tag{6}$$

While this estimate further reduces the variance, it may increase the bias if the classifier memorizes the training data. The more recently proposed 0.632^+ bootstrap estimator [4] detects when and how much the classifier is overfitting and proportionately reduces the weight of $\widehat{\mathrm{Err}_R}$. The comparison of these different estimates has been well researched and can be found in [3]–[7].

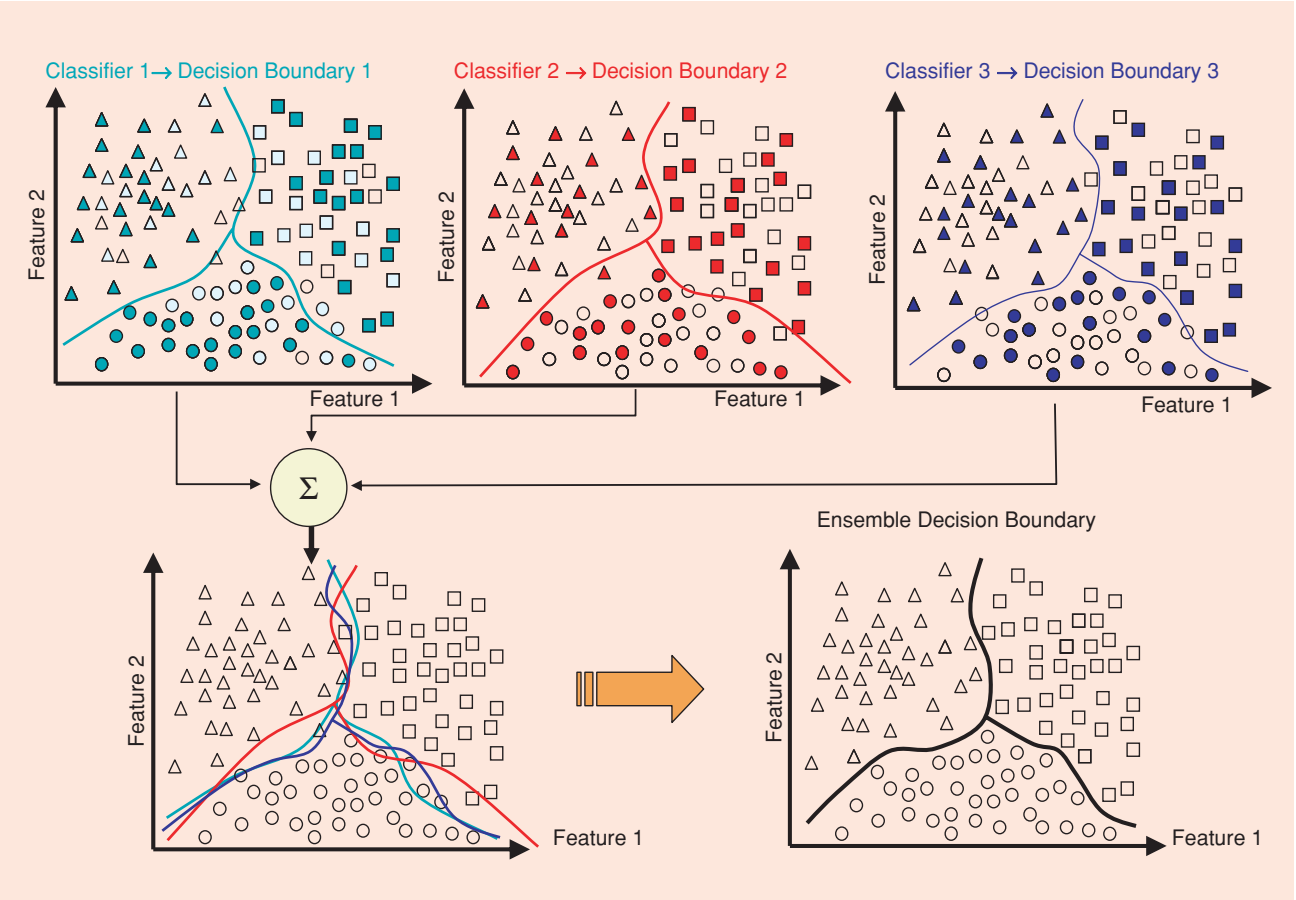
BOOTSTRAP-INSPIRED TECHNIQUES IN ENSEMBLE SYSTEMS: AN OVERVIEW

Beyond error estimation, bootstrap-based ideas have also been used in recent development of many ensemble-based algorithms. These algorithms use multiple classifiers, generally to improve classification performance: each classifier provides an alternative solution whose combination may provide a superior solution than the one provided by any single classifier. The underlying idea is in fact an intuitive one that we routinely use in our daily lives when we seek multiple opinions before making an important decision. In doing so, we weigh and combine individual opinions, in hope of making a more informed decision. Consulting several doctors before agreeing to a major surgery, or asking for references before hiring an employee, are examples of such behavior.

The primary benefit of using ensemble systems is the reduction of variance and increase in confidence of the decision. Due to many random variations in a given classifier model (different training data, different initialization, etc.), the decision obtained by any given classifier may vary substantially from one training trial to another—even if the model structure is kept constant. Then, combining the outputs of several such classifiers by, for example, averaging the output decisions, can reduce the risk of an unfortunate selection of a poorly performing classifier.

Another use of ensemble systems includes splitting large datasets into smaller and logical partitions, each used to train a separate classifier. This can be more efficient than using a single model to describe the entire data. The opposite problem, having too little data, can also be handled using ensemble systems, and this is where bootstrap-based ideas start surfacing: generate multiple classifiers, each trained on a different subset of the data, obtained through bootstrap resampling.

While the history of ensemble systems can be traced back to some earlier studies such as [8], [9], it is Schapire's 1990 paper that is widely recognized as the seminal work on ensemble systems. In *strength of weak learnability* [10], Schapire introduced *boosting*, an elegant approach to generate a strong classifier by combining weaker ones. The boosting algorithm, and its popular successor *AdaBoost* [11], generate an ensemble of classifiers, each trained with a subset of the training data resampled from the original training dataset; hence, a bootstrap approach. Unlike the standard bootstrap, however, the data distribution is strategically altered for each new classifier, giving boosting its unique attributes. Two more recent competitors to boosting are Breiman's bagging (bootstrap aggregation) used for small datasets [12], and pasting small votes used for large datasets [13], both of which follow a standard bootstrap resampling process. Other ensemble architectures that use the cross validation/jackknife approach for splitting training data include



[FIG1] Bootstrap data subsets are used to train different classifiers, which form different decision boundaries. These boundaries can be averaged to obtain a more accurate decision boundary.

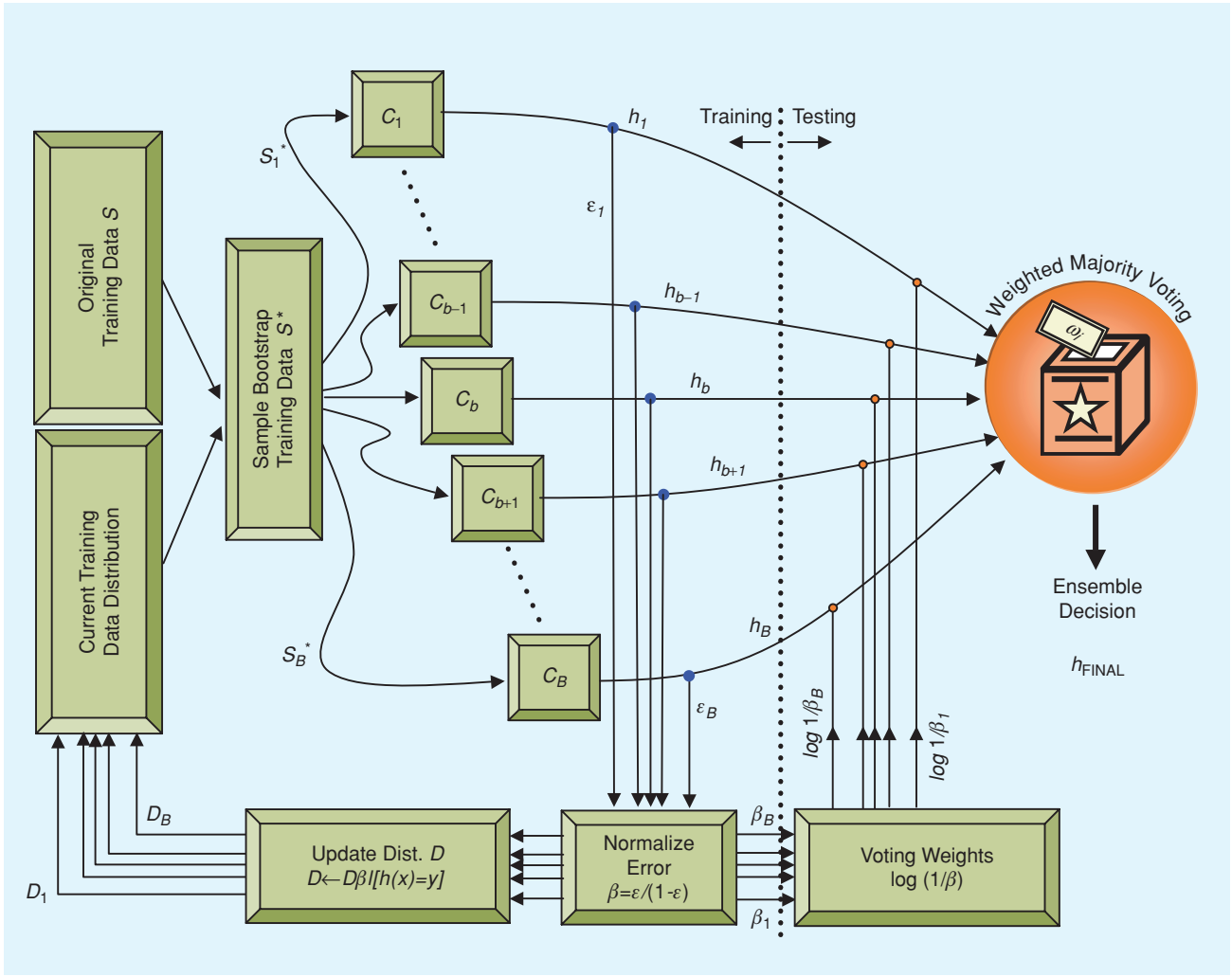
Wolpert's stacked generalization [14] and Jacob and Jordan's mixture of experts [15], [16].

The key enabling concept in all ensemble based systems is diversity. Clearly, there is no advantage in combining classifiers that provide identical outputs. An ensemble system is most beneficial if classifier outputs are independent, or better yet, negatively correlated. Specifically, we need classifiers that only differ in their misclassification but agree otherwise. Then, the ensemble can augment the correct decision and average out the individual errors (Figure 1). Diversity among classifiers can be achieved in many ways, such as training classifiers with different subsets of the features (so-called random subspace methods [17]). However, using different training data subsets obtained by resampling of the original training data is most commonly used and constitutes the link between ensemble systems and bootstrap techniques.

Of course, once the classifiers are generated, a strategy is needed to combine their outputs. In *simple majority voting*, a commonly used combination rule, each classifier votes on the class it predicts, and the class receiving the largest number of votes is the ensemble decision. It can be shown that if classifier outputs are independent, and each classifier predicts the correct class with a probability of one half or higher, the correct classification performance of the ensemble approaches "one" as the number of classifiers increases (Condorcet Jury Theorem (1786) [18]). In *weighted majority voting*, each classifier is given a voting weight inversely proportional to its resubstitution error. The class with the largest total vote is then declared the winner. Algebraic combination (e.g., sum, product) of the class-specific outputs can also be used, where the class receiving the highest combined support is then chosen by the ensemble. Theoretical analyses of these and other combination rules can be found in [19], [20].

BAGGING

Let S be the original training dataset of n instances, and S_b^*, $b = 1, \ldots, B$ be the bth bootstrap sample of size n drawn from S. One classifier is trained with each S_b^*. This resampling will result in substantial overlap in the composition of individual S_b^*. In order to ensure that a diverse set of classifiers is generated despite similar training datasets, individual classifiers are often forced to be weak (by using a small architecture, early termination of training, etc.). Once the training is complete, an ensemble decision is obtained by simple majority voting of B classifier outputs (Algorithm 1). Note that bagging, just like all other ensemble algorithms discussed here, is independent of the



[FIG2] **Block diagram of AdaBoost.M1.**

model chosen for the individual classifier and can be used with any supervised classifier.

Algorithm 1: Bagging

Inputs for Algorithm Bagging

- Training data $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, $\mathbf{x}_i \in X$, with correct labels $\omega_i \in \Omega = \{\omega_1, \cdots, \omega_C\}$
- Weak learning algorithm **WeakLearn**,
- Integer B, specifying number of iterations.

Do $b = 1, \ldots, B$

1) Obtain bootstrap sample S_b^* by randomly drawing n instances, with replacement, from S.
2) Call **WeakLearn** with S_b^* and receive the hypothesis (classifier) $h_b : X \to \Omega$.
3) Add h_b to the ensemble, E.

End Do Loop

Test: Simple Majority Voting—Given unlabeled instance $\mathbf{z}$

1) Evaluate the ensemble $\mathrm{E} = \{h_1, \ldots, h_B\}$ on $\mathbf{z}$.
2) Let the vote given to class ω_j by classifier h_b be

$$v_{b,j} = \begin{cases} 1, & \text{if } h_b \text{ picks class } \omega_j \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

3) Obtain total vote received by each class

$$V_j = \sum_{b=1}^{B} v_{b,j}, \; j = 1, \ldots, C. \tag{8}$$

4) Choose the class that receives the highest total vote as the final classification.

BOOSTING

Unlike bagging, boosting alters the training data distribution before each new bootstrap sample is obtained. The altered distribution ensures that more informative instances are drawn into the next dataset. It is this iterative distribution update that

allows boosting to make its boldest claim: a strong learner with arbitrarily high accuracy can be obtained by combining weak learners that can merely do better than random guessing; i.e., strong and weak learnability are equivalent [10].

Given training dataset S of n instances for a binary classification problem, the first classifier C_1 is trained using a bootstrap sample of $m < n$ instances. The training data subset S_2 for the second classifier C_2 is chosen such that exactly half of S_2 is correctly classified by C_1. The third classifier C_3 is then trained with instances on which C_1 and C_2 disagree. The three classifiers are combined through a three-way majority vote. Schapire showed that the error of this algorithm has an upper bound: if the largest individual error of all three classifiers (as computed on S) is ε, then the error of the ensemble is bounded above by $f(\varepsilon) = 3\varepsilon^2 - 2\varepsilon^3$. Note that $f(\varepsilon) \leq \varepsilon$ for $\varepsilon < ½$. That is, as long as all classifiers can do at least better than random guessing, then the boosting ensemble will always outperform the best classifier in the ensemble. The performance can be further improved by repeated application of the boosting process. The pseudocode of boosting is shown in detail in Algorithm 2, whereas its theoretical development can be found in [10].

Algorithm 2: Boosting

Inputs for Algorithm Boosting

- Training data S of size n, correct labels $\omega_i \in \Omega = \{\omega_1, \omega_2\}$;
- Classifier model **WeakLearn**.

Training

1) Select $m < n$ patterns *without* replacement from S to create data subset S_1.
2) Call **WeakLearn** and train with S_1 to create classifier C_1.
3) Create dataset S_2 as the most informative dataset, given C_1, such that half of S_2 is correctly classified by C_2, and the other half is misclassified. To do so:
   a) Flip a fair coin. If Head, select samples from S and present them to C_1 until the first instance is misclassified. Add this instance to S_2.
   b) If Tail, select samples from S and present them to C_1 until the first one is correctly classified. Add this instance to S_2.
   c) Continue flipping coins until no more patterns can be added to S_2.
4) Train the second classifier C_2 with S_2.
5) Create S_3 by selecting those instances for which C_1 and C_2 disagree. Train C_3 with S_3.

Testing—Given a test instance $\mathbf{z}$

1) Classify $\mathbf{z}$ by C_1 and C_2. If they agree on the class, this class is the final classification.
2) If they disagree, choose the class predicted by C_3 as the final classification.

ADABOOST

Arguably the best known of all ensemble-based algorithms, AdaBoost (Adaptive Boosting) extends boosting to multiclass and regression problems [11]. Its intuitive structure, elegant theory, and its precise performance guarantee make AdaBoost one of the most influential algorithms in recent history of computational intelligence. The most popular of AdaBoost's variations, AdaBoost.M1 for multiclass problems, is described here, whose pseudocode appears in Algorithm 3 and its conceptual block diagram in Figure 2.

Algorithm 3: AdaBoost

Inputs for Algorithm AdaBoost.M1

- Training data $S = \{\mathbf{x}_1, \mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in X$ with correct labels $y_i \in \Omega$, $\Omega = \{\omega_1, \ldots, \omega_C\}$;
- Weak learning algorithm **WeakLearn**; integer B, number of classifiers

Initialize

$$D_1(i) = 1/n. \tag{9}$$

Do for $b = 1, 2, \ldots, B$:

1) Draw bootstrap training data subset S_b^* according to current distribution D_b.
2) Train **WeakLearn** with S_b^*, receive hypothesis $h_b : X \to \Omega$.
3) Calculate the error of h_b:

$$\varepsilon_b = \sum_i I\,[\![h_b(\mathbf{x}_i) \neq y_i]\!] \cdot D_b(i). \quad \text{If } \varepsilon_b > \frac{1}{2}, \text{ abort}. \tag{10}$$

4) Calculate normalized error

$$\beta_b = \varepsilon_b/(1-\varepsilon_b) \Rightarrow 0 \leq \beta_b \leq 1 \tag{11}$$

5) Update distribution D_b:

$$D_{b+1}(i) = \frac{D_b(i)}{Z_b} \times \begin{cases} \beta_b, & \text{if } h_b(\mathbf{x}_i) = y_i \\ 1, & \text{otherwise} \end{cases} \tag{12}$$

where Z_b is a normalization constant chosen so that D_{b+1} is a proper distribution.

End Do Loop

Test—Weighted Majority Voting: Given an unlabeled instance $\mathbf{z}$,

1) Obtain total vote received by each class

$$V_j = \sum_{b:h_b(\mathbf{z})=\omega_j} \log(1/\beta_b), \quad j = 1, \ldots, C. \tag{13}$$

2) Choose the class that receives the highest total vote as the final classification.

In AdaBoost.M1, bootstrap training data samples are drawn from a distribution D that is iteratively updated such that subsequent classifiers focus on increasingly difficult instances. This is done by adjusting D such that previously misclassified instances are more likely to appear in the next bootstrap sample. The classifiers are then combined through weighted majority voting.

The distribution D starts out as uniform (9), so that all instances have equal probability to be drawn into S_1^*. At each iteration b, a new training set is drawn, and a weak classifier is trained to produce a hypothesis h_b. The error of this hypothesis with respect to the current distribution is calculated (10) as the sum of distribution weights of the instances misclassified by h_b. AdaBoost.M1 requires that this error be less than one half, a requirement with its roots in the Jury Condorcet theorem [18] mentioned above. The normalized error is obtained in (11) as β_b, which is then used in the distribution update rule of (12). Note that $D_b(i)$ is reduced by a factor of β_b, if $\mathbf{x}_i$ is correctly classified by h_b, and left unchanged otherwise. When the distribution is normalized so that $D_{b+1}(i)$ is a proper distribution, the weights of those instances that are misclassified are effectively increased. Once the training is complete, test data are classified by this ensemble of B classifiers using weighted majority voting, where each classifier receives a voting weight that is inversely proportional to its normalized error [(13) and (14)]. The weighted majority voting then chooses the class ω receiving the highest total vote from all classifiers:

$$h_{\text{final}}(\mathbf{x}_i) = \arg\max_{\omega \in \Omega} \sum_{b=1}^{B} I\left[\left[h_b(\mathbf{x}_i) = \omega\right]\right] \cdot \log(1/\beta_b). \tag{14}$$

The theoretical analysis of this algorithm shows that the ensemble error E of AdaBoost is bounded above by [11]

$$E < 2^B \Pi_{b=1}^{B} \sqrt{\varepsilon_b(1-\varepsilon_b)}. \tag{15}$$

Since $\varepsilon_b < 1/2$, ensemble error E monotonically decreases as new classifiers are added. Conventional wisdom indicates that adding classifiers—beyond a certain limit—would eventually lead to overfitting of the data. One of the most celebrated features of AdaBoost, however, is its surprising resistance to overfitting, a phenomenon whose explanation is based on the margin theory [21].

STACKED GENERALIZATION AND MIXTURE OF EXPERTS

The idea in stacked generalization is to learn whether training data have been properly learned. Bootstrap-inspired resampling provides a natural mechanism to do so: classifiers $C_1, \ldots, C_B$, called tier-1 classifiers, are first generated on bootstrap (or similarly obtained) subsets of the training data. A meta-classifier C^* is then trained to map the tier-1 classifier outputs to their correct labels [14]. CV-type selection is typically used: specifically, the entire training dataset is divided into B blocks, and each tier-1 classifier is first trained on (a different set of) $B-1$ blocks of the training data. Each classifier is then evaluated on the Bth (pseudo-test) block, not seen during training. The outputs of these classifiers on their pseudo-training blocks along with the actual correct labels for those blocks constitute the training dataset for C^*.

Mixture of experts (MoE) is a similar algorithm, but tier-1 classifiers are combined through a weighted combination rule that uses a gating network [15]. The gating network typically implements the expectation-maximization (EM) algorithm on the training data to determine the gating weights [16]. Figure 3 illustrates the structure for both algorithms, where boxes and connections in solid lines are common to both, and dashed lines indicate MoE-specific components only.

EMERGING AREAS FOR BOOTSTRAP INSPIRED METHODS IN ENSEMBLE SYSTEMS

New areas have recently emerged that make creative use of bootstrap-inspired ensemble systems. Three of these, incremental learning, data fusion, and missing feature, are described below.

INCREMENTAL LEARNING

Many real-world applications, where data become available in batches over a period of time, require that the information provided by each dataset be incrementally learned. The problem becomes particularly challenging if the previously seen data are no longer available when new data arrive, as the traditional approach of combining old and new data to train a new classifier then becomes infeasible. Learning from new data without having access to old data, while retaining previously acquired knowledge, is called incremental learning. For a sequence of training datasets, an incremental learning algorithm produces a sequence of hypotheses, where the current hypothesis describes all data seen thus far but depends only on previous hypotheses and the current data.

This definition raises the stability–plasticity dilemma: some information will inevitably be lost to learn new information [22]. This is because stability is the ability of a classifier to retain its knowledge, whereas plasticity is the ability to learn new knowledge; and the two phenomena typically contradict each other. Many popular classifiers, such as the multilayer perceptron (MLP), radial basis function networks, and support vector machines are all stable classifiers. In their native form, they are not structurally suitable for incremental learning.

While there are several approaches for incremental learning, we focus on recently developed ensemble-based systems that make creative use of bootstrap based ideas. These approaches involve generating an ensemble of classifiers for each dataset that becomes available, resulting in an ensemble of ensembles. A suitably modified version of AdaBoost, run iteratively for each new dataset, can learn incrementally if the data distribution does not change in time (stationary learning) [23]. However, a more interesting—and arguably more challenging—problem is the introduction of new classes, or different number of classes being represented in each new dataset. By making strategic modifications to the bootstrap resampling distribution, a similar approach can still be used to learn incrementally under these scenarios. Learn^{++} is such an algorithm, shown to learn incrementally from new data, even when such data introduce new classes [23], [24].

Recall that the distribution update rule in AdaBoost is designed to focus its bootstrap samples on increasingly difficult

instances, determined according to the performance of the last classifier (12). However, in incremental learning, the algorithm must also focus on those instances that carry novel information. This can be accomplished more effectively by controlling the distribution update rule (and the bootstrap resampling) by the performance of the entire ensemble instead of the previous classifier, and appropriately reinitializing the distribution every time a new dataset is introduced, so that the algorithm can immediately focus on the novel information in the new data. Algorithm 4 shows the pseudocode of Learn^{++}, which implements these ideas.

Algorithm 4: Learn^{++}

Inputs for Algorithm Learn^{++}

For each dataset drawn from DS_k $k = 1, 2, \ldots, K$

- Training data $S^k = \{\mathbf{x}_i | \mathbf{x} \in X, i = 1, \ldots, n_k\}$ with correct labels $y_i \in \Omega$, $\Omega = \{\omega_1, \ldots, \omega_C\}$;
- Supervised algorithm **BaseClassifier**, and the number of classifiers B_k to be generated

Do for each dataset DS_k, $k = 1, 2, \ldots, K$:

Initialize

$$w_1(i) = D_1(i) = 1/n_k, \ \forall i, \ \ i = 1, 2, \cdots, n_k. \tag{16}$$

If $k > 1$, Go to Step 6: Evaluate composite hypothesis $H^{k-1}_{B_{k-1}}$ on new S^k, update weights; End If

Do for $b = 1, 2, \ldots, B_k$:

1) Set

$$D_b = \mathrm{w} \Big/ \sum_{i=1}^{n_k} w_b(i) \text{ so that } \boldsymbol{D_b} \text{is a distribution} \tag{17}$$

2) Draw a bootstrap sample S^k_b of size $n < n_k$ from D_b, and train **BaseClassifier** on S^k_b

3) Obtain hypothesis h^k_b, calculate its error

$$\varepsilon^k_b = \sum_i I\left[\left[h^k_b(\mathbf{x}_i) \neq y_i\right]\right] D_b(i) \text{ on } S^k_b. \tag{18}$$

If $\varepsilon^k_b > 1/2$, discard h^k_b, go to Step 2. Otherwise,

$$\beta^k_b = \varepsilon^k_b \Big/ \left(1 - \varepsilon^k_b\right) \tag{19}$$

4) Obtain composite hypothesis

$$H^k_b(\mathbf{x}) = \arg\max_{\omega_c \in \Omega} \sum_{p=1}^{k} \sum_{q=1}^{b} \log\left(1/\beta^p_q\right) I\left[\left[h^p_q(\mathbf{x}) = \omega_c\right]\right] \tag{20}$$

5) Compute the error of H^k_b on S^k:

$$E^k_b = \sum_{i=1}^{m_k} D_b(i) \cdot I\left[\left[|H^k_b(\mathbf{x}_i) \neq y_i|\right]\right] \tag{21}$$

6) Set

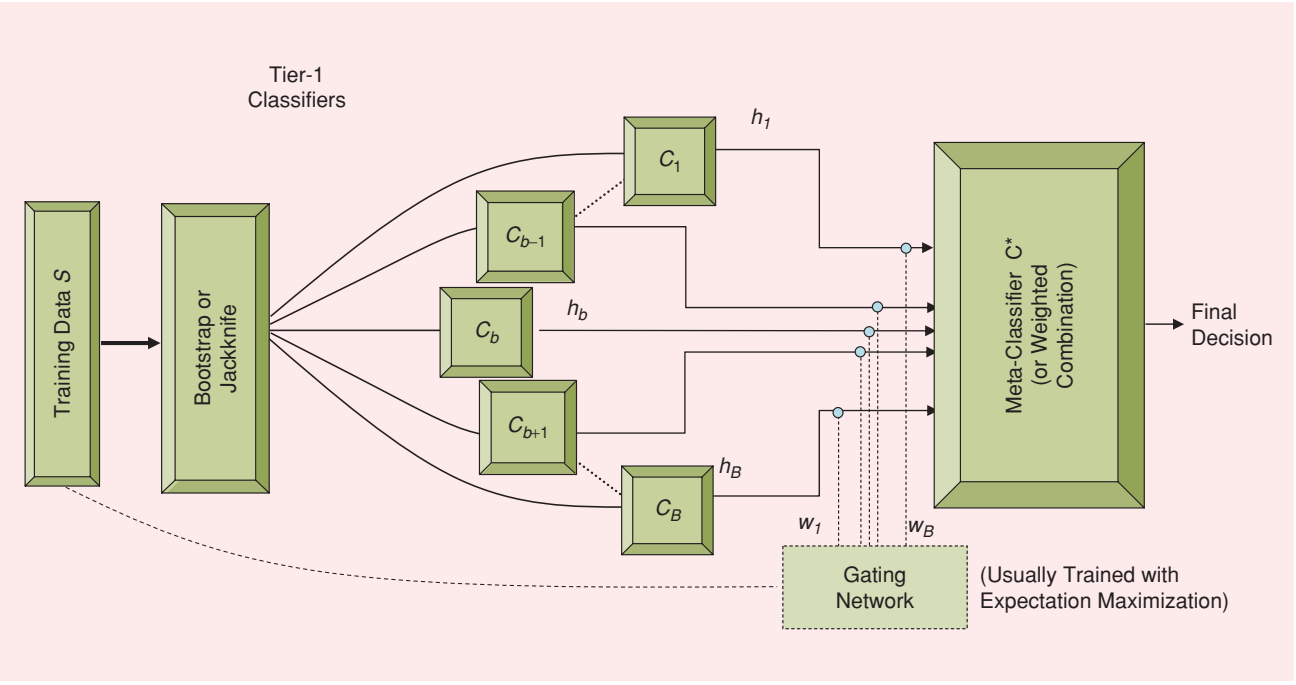
$$B^k_b = E^k_b \Big/ \left(1 - E^k_b\right), \tag{22}$$

update the weights:

$$w_{b+1}(i) = w_b(i) \times \begin{cases} B^k_b, & H^k_b(\mathbf{x_i}) = y_i \\ 1, & \text{otherwise.} \end{cases} \tag{23}$$

End Do Loop

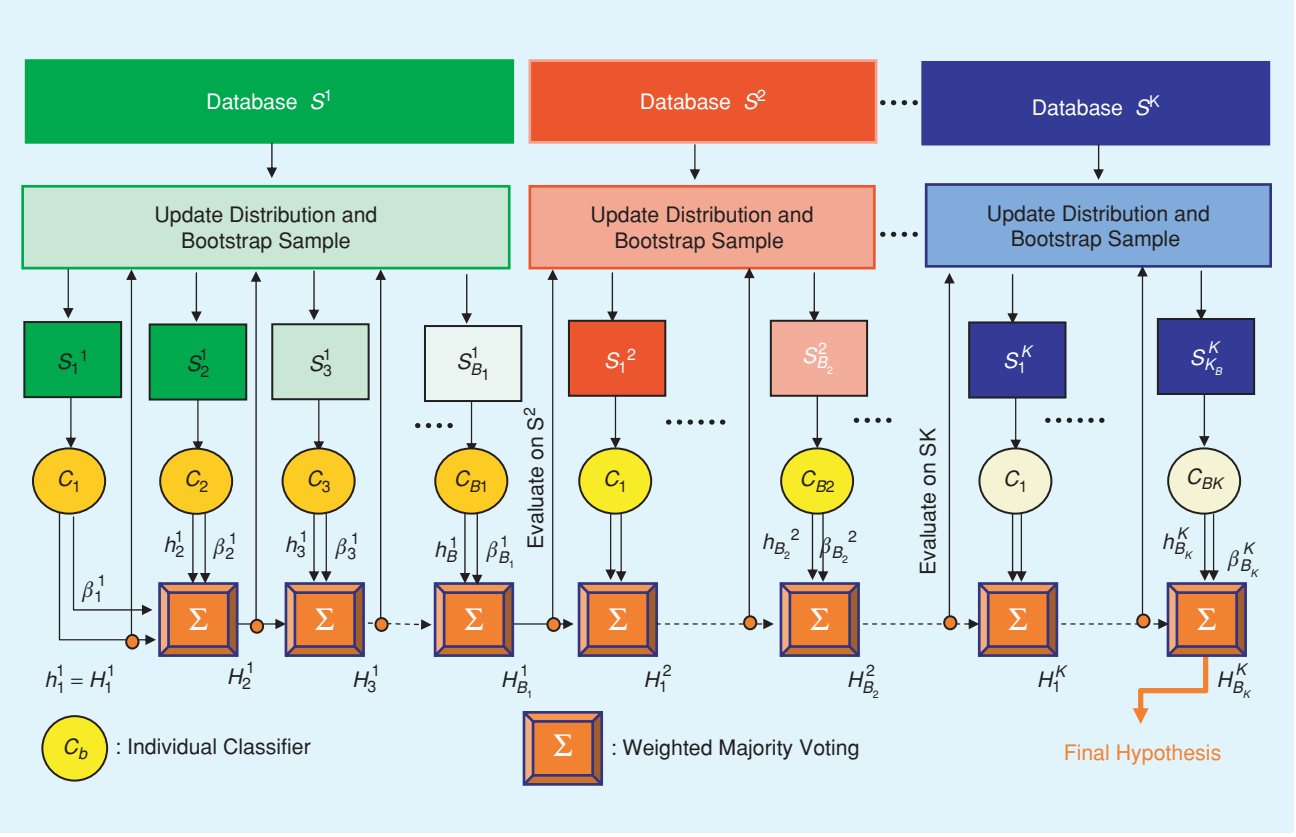
End Do Loop



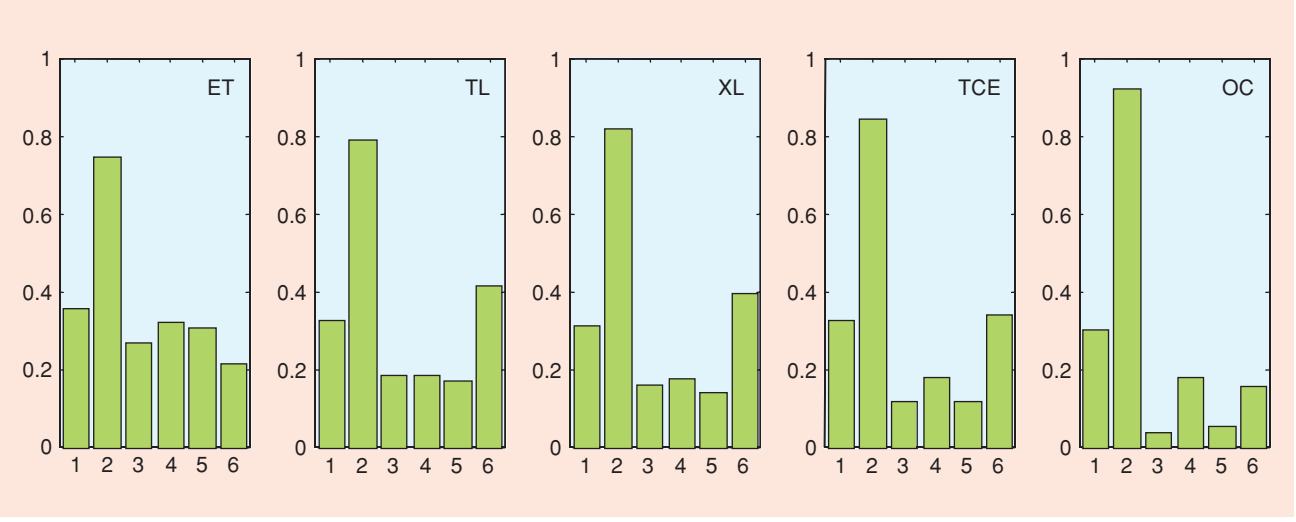
[FIG3] **Block diagram for stacked generalization and mixture of expert models.**

[TABLE 1] GENERALIZATION PERFORMANCE (%) OF LEARN^{++} ON VOC DATA.

CLASS→	ET	OC	TL	TCE	XL	TEST
TS_1 **(3)**	80 ± 4.1	93 ± 2.4	89 ± 1.9	-	-	**52 ± 0.9**
TS_2 **(7)**	86 ± 3.1	94 ± 1.9	78 ± 3.1	83 ± 6.9	-	**68 ± 1.4**
TS_3 **(14)**	86 ± 3.5	95 ± 1.8	68 ± 2.9	92 ± 2.3	80 ± 8.6	**84 ± 1.9**



[FIG4] **Block diagram of the Learn^{++} algorithm.**



[FIG5] **Typical response patterns for the VOC database.**

[TABLE 2] DATA DISTRIBUTION AND INCREMENTAL LEARNING GENERALIZATION PERFORMANCE ON OCR DATA.

DATASET ↓	0	1	2	3	4	5	6	7	8	9	TEST (%)
S^1	250	250	250	0	0	250	250	0	0	250	**59.16 ± 0.11**
S^2	100	100	100	250	250	100	100	0	0	100	**77.25 ± 0.36**
S^3	0	0	50	150	150	50	50	400	400	0	**89.26 ± 0.97**
TEST	100	100	100	100	100	100	100	100	100	100	—

Call Weighted Majority Voting and output the final hypothesis.

$$H_{\text{final}}(\mathbf{x}_i) = \arg \max_{\omega_c \in \Omega} \sum_k \sum_b \log\left(1 \Big/ \beta_b^k\right) \cdot \left(I\left[\left[h_b^k(\mathbf{x}_i) = \omega_c\right]\right]\right). \tag{24}$$

First, note that the distribution update rule in (23) is now a function of the composite hypothesis H_b^k, which represents the overall ensemble decision. It is obtained as the weighted majority voting of all classifiers generated thus far, as shown in (20). Hence, the bootstrap sample for training the next classifier focuses specifically on what the ensemble has not seen and/or learned thus far, giving it the incremental learning ability. Also note that when new data become available, Learn^{++} first initializes the distribution to be uniform on the new data in (16) and calls the then-current composite hypothesis $H_{B_{k-1}}^{k-1}$ to evaluate the existing ensemble on the new data. The distribution is adjusted based on this evaluation, before the first bootstrap sample is even drawn from the new data. In doing so, the algorithm tracks those instances of the new data that have already been learned by the current ensemble, and instead focuses on other instances that carry the novel information. Finally, note that all classifiers are retained to prevent loss of information, assuming that all previous data still carry relevant information. If previous information is no longer relevant, then a forgetting mechanism can be introduced to remove irrelevant classifiers [25], [26]. The block diagram of the entire algorithm is illustrated Figure 4. We now look at two real-world applications where Learn $^{++}$ can be used to learn incrementally from new data that subsequently introduce instances of previously unseen classes.

A BOOTSTRAP-BASED METHOD CAN PROVIDE AN ALTERNATIVE APPROACH TO THE MISSING DATA PROBLEM BY GENERATING AN ENSEMBLE OF CLASSIFIERS, EACH TRAINED WITH A RANDOM SUBSET OF THE FEATURES.

APPLICATION: IDENTIFICATION OF VOLATILE ORGANIC COMPOUNDS (VOCS)

This is a challenging real-world problem of identifying VOCs from the responses of six quartz crystal microbalance (QCM) type chemical sensors. The five classes are the individual VOCs to be identified: ethanol (ET), toluene (TL), xylene (XL), thricholoroethylene (TEC), and octane (OC). The features are changes in resonant frequencies of six QCMs when exposed to a particular VOC. Figure 5 illustrates the typical normalized response patterns, where each bar represents the response from one QCM. Note that TL and XL responses are particularly similar. The training data comes in three batches. In the first dataset S^1, data from ET (20 instances), OC (20 instances), and TL (40 instances) were available. Dataset S^2 introduced 25 TCE instances as well as 10 from ET, OC, and XL each, and the last dataset S^3 introduced 40 XL instances with only ten from each of the previous four. A validation set was used to determine free algorithm parameters such as ensemble size, and a separate TEST set (15 to 30 instances in each class) was used for evaluating the performance. The small size of this database makes this problem challenging but also a good fit for bootstrap-based approaches. The algorithm was trained in three training sessions $TS_1 \sim TS_3$, during each of which only the current dataset was seen by the algorithm; hence, incremental learning.

Table 1 presents the class-specific classification performance on the TEST data, as well as the overall generalization performance on the entire TEST data after each training session. Ensemble size for each session is shown in parentheses in the first column. The 95% confidence intervals are obtained through ten independent trials. The class-specific performances on instances of the classes seen during training are very good, but the overall performance is only 52% on the entire TEST data after the first training session TS_1. This makes sense, as the algorithm has only seen three of the five classes of the overall database. The performance increases steadily, reaching 68% after TS_2, and 84% after TS_3. This demonstrates that the algorithm can learn incrementally and also retain previously acquired knowledge as indicated by retained class-specific performances.

APPLICATION: OPTICAL CHARACTER RECOGNITION (OCR)

The OCR database [27] consists of handwritten numerical characters $0 \sim 9$, digitized on an 8x8 grid. This database was available in its entirety at once; however, it was partitioned into four sets to simulate an incremental learning environment. The larger number of classes and the relatively larger size of the database allowed us to experiment with challenging scenarios, such as different classes being represented in each database. Table 2 shows the data distribution for this experiment. Similar to the VOC

problem, the algorithm was trained in three training sessions, using five MLPs as base classifiers in each session. For space considerations, and due to large number of classes, generalization performance on the entire TEST data (but not class-specific performances) is also provided in Table 2, following each of the training sessions. The increasing performance on the test data once again demonstrates the incremental learning ability of the approach. Incremental learning performance of the algorithm on other datasets, with comparison to AdaBoost, can be found in [23].

STABILITY IS THE ABILITY OF A CLASSIFIER TO RETAIN ITS KNOWLEDGE, WHEREAS PLASTICITY IS THE ABILITY TO LEARN NEW KNOWLEDGE; AND THE TWO PHENOMENA TYPICALLY CONTRADICT EACH OTHER.

DATA FUSION

In many applications of automated decision making, it is also common to receive data from different sources that provide complementary information. A suitable combination of such information is known as data fusion and can lead to improved accuracy of the classification decision compared to a decision based on any of the individual data sources alone. The conceptual similarity between incremental learning and data fusion allows an ensemble-based system for the former to be suitably modified for data fusion: both involve learning from different sets of data, except that the consecutive datasets in data fusion are obtained from different sources and/or consist of different features. Apart from regular voting weights, feature specific voting weights are also needed to incorporate any prior information on the usefulness of a specific feature set.

APPLICATION: NONDESTRUCTIVE TESTING OF NATURAL GAS PIPELINES

Nondestructive testing (NDT) is the evaluation of an object to detect internal defects without compromising the object's structural integrity. The defect is typically an anomaly in the object's structure, such as crack, corrosion, mechanical damage, porosity, etc. The testing usually involves launching some form of energy into the material and analyzing the received signal reflected from the discontinuities within the material. Common NDT methods include ultrasonic testing, magnetic flux leakage imaging, x ray, and eddy-current imaging. NDT applications are numerous, such as defect identification in gas transmission pipelines, power plant tubings, engine components, etc.

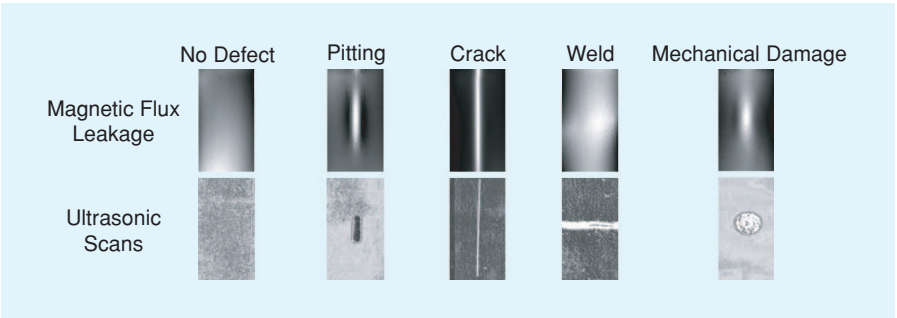
In this application, we are interested in automated identification of four types of discontinuities commonly occurring in gas transmission pipelines. Nondestructive inspection of these pipelines is critical, as certain defects, such as cracks, may eventually cause gas leak and explosion, if not detected early. Figure 6 illustrates typical images of five categories of interest, obtained through two physically very different imaging modalities: magnetic flux leakage and ultrasonic testing.

Due to difficulties in collecting real data, this dataset was also very small: 21 images for five categories. Two images from each class were chosen randomly for training, and the remaining 11 were used for testing. Keeping the training and test sizes constant, the process was repeated 40 times, drawing a different bootstrap sample in each trial. Data from each imaging modality was first used individually to train an ensemble of 30 classifiers using Learn^{++} with MLPs as base classifiers and discrete cosine transform coefficients as features. These classifiers were then fused by weighted majority voting. While identification performances of individual sources were $81.60 \pm 3.62\%$ (MFL) and $79.87 \pm 2.69\%$ (UT), the performance of the ensemble-based fusion (of 60 classifiers) was $95.02 \pm 1.99\%$. The significant improvement over individual performances indicates that the two data sources do provide complementary information, and that such information can be fused using the ensemble-based approach described above. Implementation details and results on other applications can be found in [28].

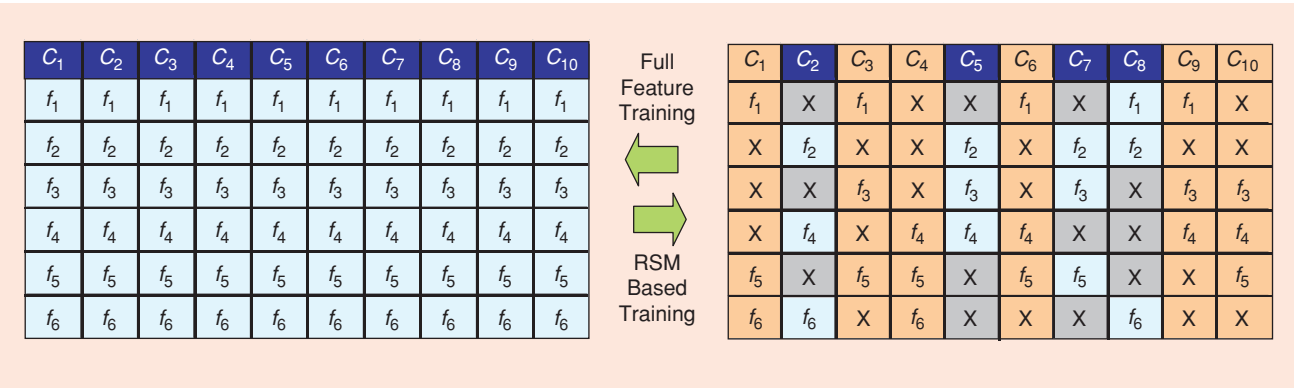
MISSING FEATURES/MISSING DATA

It is not unusual for training, validation, or field data to have missing features, as bad sensors, failed pixels, malfunctioning equipment, data corruption, etc., are all familiar scenarios in real-world applications. While theoretically rigorous and well-established approaches exist to handle missing data, these typically involve estimating the values of the missing data, and they require either sufficiently dense training data, some prior knowledge of the underlying distribution that generated the data, or both. In such cases, particularly if each instance is missing several features, methods that estimate the value of the missing data become less reliable. Simple data imputation, Bayesian estimation, and expectation maximization are examples of such approaches.

A bootstrap-based method can provide an alternative approach to the missing data problem by generating an ensemble



[FIG6] **Typical MFL and UT images for four discontinuities compared to no-defect.**



[FIG7] **Training an ensemble of classifiers on random feature subsets.**

of classifiers, each trained with a random subset of the features. Such an approach is similar to algorithms that employ random subspace methods (RSM) [17], which are typically used to improve classification accuracy, or to select optimal subset of features when the feature space is redundant [29]. Using RSM for missing features was recently addressed in [30], where random feature selection was used within the Learn^{++} structure under the name Learn^{++}.MF.

Learn^{++}.MF generates a sufficiently large number of classifiers; each trained with a random subset of the features. Since each classifier is trained on a subset of the features, an instance with missing features can still be classified by majority voting of those classifiers whose training data did not include the features currently missing. In doing so, Learn^{++}.MF takes full advantage of the existing data, rather than trying to estimate the values of the missing data. The algorithm makes the basic assumption that there is randomly distributed redundancy in the features, an assumption satisfied by many applications of nontime-series data. Figure 7 illustrates this approach. On the left, an ensemble of ten classifiers are trained on a six-feature problem, $f_1, \ldots, f_6$, using the full set of features. At the time of testing, if an instance is missing even a single feature, say f_2, none of the classifiers can be used, since each classifier needs a value for f_2. On the right, the same ensemble is constructed by training each classifier using only three randomly selected features. Those features not used in training are indicated with an X for each classifier. For example, C_1 was trained using features f_1, f_5, and f_6; C_2 was trained using f_2, f_4, and f_6, etc. In this case, a test instance missing feature f_2 can still be classified by (highlighted) classifiers C_1, C_3, C_4, C_6, C_9, and C_{10}. In fact, up to three missing features can be accommodated in this particular example.

Algorithm 5 shows the pseudocode of Learn^{++}.MF, where B classifiers are trained using *nof* features randomly selected from a total of f features. $F_{\text{selection}}(b)$ lists those features used to train bth classifier C_b. Note that each feature subset is an independent bootstrap sample on the features. During testing, we first determine which features are missing in the given test instance $\mathbf{z}_i$, and keep the index of these features in $M_{\text{feat}}(i)$. Classifiers whose training features $F_{\text{selection}}(b)$ do not include any of the features in $M_{\text{feat}}(i)$ are combined through simple majority voting to classify $\mathbf{z}_i$.

Algorithm 5: Learn^{++}.MF

Inputs for Algorithm Learn^{+++}.MF

- Sentinel value *sen*, indicating a missing or corrupt value, and integer B, indicating the ensemble size.
- Training data $S = \{\mathbf{x}_i | \mathbf{x} \in \Re^f;\ i = 1, \ldots, n\}$ with correct labels $y_i \in \Omega = \{\omega_1, \ldots, \omega_C\}$.
- Number of features, $nof < f$, to be used to train each classifier.

Do for $b = 1, 2, \ldots, B$:

1) Draw *nof* bootstrap features into $F_{\text{selection}}(b)$.
2) Generate classifier C_b using features in $F_{\text{selection}}(b)$.
3) If the performance of C_b on S, $Perf_b < 50\%$, discard C_b and go to Step 2.

End Do Loop

Validation/Testing

Given test data $\tilde{S} = \{\mathbf{z}_i | \mathbf{z} \in \Re^f;\ i = 1, \ldots, m\}$

Do for $i = 1, 2, \ldots, m$:

1) Determine missing features of z_i :

$$M_{\text{feat}}(i) = \arg_j(\mathbf{z}_i(j) == sen),\ \forall j,\ j = 1, \ldots, f \tag{25}$$

2) Classify $\mathbf{z}_i$ with majority voting:

$$C(\mathbf{z}_i) = \arg\max_{\omega_c \in \Omega} \sum_{b: C_b(\mathbf{z}_i) = \omega_c} I\left[\left[M_{\text{feat}}(i) \notin F_{\text{selection}}(b)\right]\right]. \tag{26}$$

End Do Loop

The algorithm's primary free parameter *nof*, indicating the number of features to be selected into the feature subset, must be determined judiciously. Choosing *nof* much smaller than f

[TABLE 3] LEARN++ .MF PERFORMANCE ON VOC DATA WITH UP TO 20% MISSING FEATURES (B = 200).

	(nof = 2/6)		(nof = 3/6)	
% MISSING FEATURES (PMF)	% MEAN PERFORMANCE	% INSTANCES PROCESSED (PIP)	% MEAN PERFORMANCE	% INSTANCES PROCESSED (PIP)
0.00%	77.45 ± 0.00	100	85.29 ± 0.00	100
2.50%	77.70 ± 0.47	100	84.80 ± 0.23	100
5.00%	77.89 ± 0.58	100	84.79 ± 0.76	100
7.50%	77.39 ± 0.83	100	84.75 ± 0.87	100
10.00%	77.18 ± 0.60	100	84.28 ± 0.69	100
20.00%	77.08 ± 0.90	100	81.56 ± 0.66	98

[TABLE 4] LEARN++ .MF PERFORMANCE ON OCR DATA WITH UP TO 20% MISSING FEATURES (B = 1,000).

	(nof = 16/64)		(nof = 20/64)		(nof = 24/64)	
(PMF)	% MEAN PERFORMANCE	(PIP)	% MEAN PERFORMANCE	(PIP)	% MEAN PERFORMANCE	(PIP)
0.00%	96.27 +/− 0.00	100	96.69 +/− 0.00	100	97.02 +/− 0.00	100
2.50%	96.25 +/− 0.04	100	96.66 +/− 0.07	100	97.02 +/− 0.08	100
5.00%	96.11 +/− 0.11	100	96.50 +/− 0.09	100	96.93 +/− 0.08	100
7.50%	95.99 +/− 0.11	100	96.41 +/− 0.15	100	96.50 +/− 0.08	100
10.00%	95.78 +/− 0.07	100	96.08 +/− 0.10	100	95.86 +/− 0.17	99
20.00%	92.44 +/− 0.07	97	90.61 +/− 0.29	83	89.89 +/− 0.42	59

allows the algorithm to accommodate instances with larger number of features missing (up to $f - nof$). However, using very few features to train each classifier may result in poor classifier performance, as training with very few features may prevent the classifier from converging to a solution.

APPLICATIONS

We now return to the five-class, six-feature VOC identification problem introduced earlier. Previous experiments using optimized classifiers trained on the entire dataset ($S^1 \cup S^2 \cup S^3$) with *all* six features, and evaluated on the test data with no missing features, provided a generalization performance of 86.2%, setting the benchmark target performance for this database. Two values of *nof* were considered: $nof = 2$ and $nof = 3$, out of six features. Missing features were simulated by randomly removing a certain ratio of the features [percent missing features (PMF)] from the test data. Table 3 summarizes the test performances and the percentage of instances that could be processed—correctly or otherwise—with the existing ensemble. All results indicate 95% confidence intervals obtained through ten independent trials of the entire experiment. The first row with 0.0% PMF is the algorithm's performance when individual classifiers were trained using *nof* features but evaluated on a fully intact test data. The proximity of this number (85.3% for $nof = 3$) to the benchmark target (86.2%, $nof = 6$) indicates that this dataset does in fact have redundant features.

USING DIFFERENT TRAINING DATA SUBSETS OBTAINED BY RESAMPLING OF THE ORIGINAL TRAINING DATA IS MOST COMMONLY USED AND CONSTITUTES THE LINK BETWEEN ENSEMBLE SYSTEMS AND BOOTSTRAP TECHNIQUES.

Since the feature subsets for each classifier are selected at random, it is possible that a particular set of features available for any given instance—after removing the missing ones—do not match the feature combinations selected by any of the classifiers. Such an instance cannot be processed by the ensemble, as there would be no classifier trained with the unique combination of the available features. The column "% Instances Processed (PIP)" in Tables 3 and 4 indicate the percentage of those instances that can be processed by the ensemble. PIP decreases as PMF increases, particularly for datasets with large number of features. In this experiment, 98% of all instances could still be processed (using $nof = 3$), at an accuracy of 81.54% (only 4.5% drop from target rate), even when 20% of data were missing.

Also consider the 64-feature, ten-class OCR database, which had a target performance of 98.5%. Table 4 presents the results of the ensemble using three *nof* values of 16, 20, and 24 out of 64 features. Notice for this dataset that the performances do not change much for different values of *nof*; however, the PIP differs substantially at high PMF cases: while larger *nof* values provide (slightly) higher initial performance, smaller *nof* values—as expected—allow larger portion of instances to be processed as the amount of missing data increase.

CONCLUSIONS

Bootstrap-based algorithms have had profound influence on how we make inference from small sample datasets. Perhaps

somewhat more quietly, bootstrap-inspired ensemble approaches have also enjoyed great attention in computational intelligence problems, particularly for those problems with small data size. This article reviewed several examples of these algorithms that allow us to create strong classifiers from an ensemble of weaker ones. Such algorithms make good use of small datasets by training multiple classifiers on bootstrap samples of the available data.

Bootstrap approaches also allow us to address a suite of new challenges in computational intelligence. Three such problems were reviewed in this article. Incremental learning of new information from additional datasets, particularly when such data introduce new concept classes, can be achieved by creating an ensemble of ensembles. A new ensemble is generated using each new dataset, where individual classifiers are trained with bootstrapped samples of the training data, whose distribution is adjusted to ensure that the novel information is efficiently learned. In data fusion, a similar approach is followed, even when the individual datasets are drawn from different sources, and hence use different feature sets. Finally, for the missing feature problem, an ensemble of classifiers is trained, where training data for each classifier is obtained as a bootstrap sample on the features. The instances missing certain features are then classified by the combination of those classifiers whose training data did not include the missing features.

ACKNOWLEDGMENTS

The work on ensemble-based incremental learning, data fusion, and missing feature algorithm development is supported by National Science Foundation under Grant No. ECS-0239090.

AUTHOR

Robi Polikar (polikar@rowan.edu) is an associate professor of Electrical and Computer Engineering at Rowan University, Glassboro, New Jersey. He received his B.Sc. degree in electronics and communications engineering from Istanbul Technical University, Turkey in 1993 and his M.Sc and Ph.D. degrees, both comajors in electrical engineering and biomedical engineering, from Iowa State University, Ames, in 1995 and 2000, respectively. His current research interests are in pattern recognition, ensemble systems, computational models of learning, incremental and nonstationary learning, and their applications in biomedical engineering. His research is primarily supported by NSF's CAREER program and NIH's Collaborative Research in Computational Neuroscience program. He is a member of the IEEE, ASEE, Tau Beta Pi, and Eta Kappa Nu.

REFERENCES

[1] B. Efron, "Bootstrap methods: Another look at the jackknife," *Ann. Statist.*, vol. 7, no. 1, pp. 1–26, 1979.

[2] A.M. Zoubir and D.R. Iskander, *Bootstrap Techniques for Signal Processing*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[3] B. Efron, "Estimating the error rate of a prediction rule: Improvement on cross-validation," *J. Amer. Stat. Assoc.*, vol. 78, no. 382, pp. 316–331, 1983.

[4] B. Efron and R. Tibshirani, "Improvements on cross-validation: The .632+ bootstrap method," *J. Amer. Stat. Assoc.*, vol. 92, no. 43, pp. 548–560, 1995.

[5] A.K. Jain, R.C. Dubes, and C.C. Chen, "Bootstrap techniques for error estimation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 9, no. 5, pp. 628–633, 1987.

[6] M.R. Chernick, V.K. Murthy, and C.D. Nealy, "Estimation of error rate for linear discriminant functions by resampling: Non-Gaussian populations," *Comput. Math. Appl.*, vol. 15, no. 1, pp. 29–37, 1988.

[7] R. Kohavi, *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*. San Mateo, CA: Morgan Kaufman, 1995, pp. 1137–1143.

[8] B.V. Dasarathy and B.V. Sheela, "Composite classifier system design: Concepts and methodology," *Proc. IEEE*, vol. 67, no. 5, pp. 708–713, 1979.

[9] L.K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, no. 10, pp. 993–1001, 1990.

[10] R.E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, June 1990.

[11] Y. Freund and R. Schapire, "Decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997.

[12] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[13] L. Breiman, "Pasting small votes for classification in large databases and online," *Mach. Learn.*, vol. 36, no. 1-2, pp. 85–103, 1999.

[14] D.H. Wolpert, "Stacked generalization," *Neural Netw.*, vol. 5, no. 2, pp. 241–259, 1992.

[15] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton, "Adaptive mixtures of local experts," *Neural Comput.*, vol. 3, no. 1, pp. 79–87, 1991.

[16] M.J. Jordan and R.A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural Comput.*, vol. 6, no. 2, pp. 181–214, 1994.

[17] T.K. Ho, "Random subspace method for constructing decision forests," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 20, no. 8, pp. 832–844, 1998.

[18] P.J. Boland, "Majority system and the Condorcet jury theorem," *Statistician*, vol. 38, no. 3, pp. 181–189, 1989.

[19] J. Kittler, M. Hatef, R.P.W. Duin, and J. Mates, "On combining classifiers," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 20, no. 3, pp. 226–239, 1998.

[20] L.I. Kuncheva, "A theoretical study on six classifier fusion strategies," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 24, no. 2, pp. 281–286, 2002.

[21] R.E. Schapire, Y. Freund, P. Bartlett, and W.S. Lee, "Boosting the Margin: A new explanation for the effectiveness of voting methods," *Ann. Statist.*, vol. 26, no. 5, pp. 1651–1686, 1998.

[22] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural Netw.*, vol. 1, no. 1, pp. 17–61, 1988.

[23] H.S. Mohammed, J. Leander, M. Marbach, and R. Polikar, "Can AdaBoost.M1 learn incrementally? A comparison to Learn++ under different combination rules," in *Lecture Notes in Computer Science*, vol. 4131, S. Kolias, A. Stafylopatis, W. Duch, and E. Oja, Eds. Berlin: Springer-Verlag, 2006, pp. 254–263.

[24] R. Polikar, L. Udpa, S.S. Udpa, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Trans. Systems, Man. Cybern. C*, vol. 31, no. 4, pp. 497–508, 2001.

[25] L.I. Kuncheva, "Classifier ensembles for changing environments," in *Lecture Notes in Computer Science*, vol. 3077, F. Roli, J. Kittler, and T. Windeatt, Eds. Berlin: Springer-Verlag, 2004, pp. 1–15.

[26] D. Muhlbaier and R. Polikar, "An ensemble approach for incremental learning in nonstationary environments," in *Lecture Notes in Computer Science*, M. Haindl and F. Roli, Eds. Berlin: Springer-Verlag, 2007, vol. 4472, pp. 490-500.

[27] D. Newman, S. Hettich, C. Blake, and C. Merz , "UCI repository of machine learning database at Irvine CA" (1988) [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[28] D. Parikh and R. Polikar, "An ensemble based incremental learning approach to data fusion," *IEEE Trans. Syst., Man. Cybern. B*, vol. 37, no. 2, pp. 437–450, 2007.

[29] C. Lai, M.J.T. Reinders, and L. Wessels, "Random subspace method for multivariate feature selection," *Pattern Recognit. Lett.*, vol. 27, no. 10, pp. 1067-1076, 2006.

[30] J. DePasquale and R. Polikar, "Random feature subset selection for ensemble based classification of data with missing features," in *Lecture Notes in Computer Science*, vol. 4472, M. Haindl and F. Roli, Eds. Berlin: Springer-Verlag, vol. 4472, 2007, pp. 251-260. SP