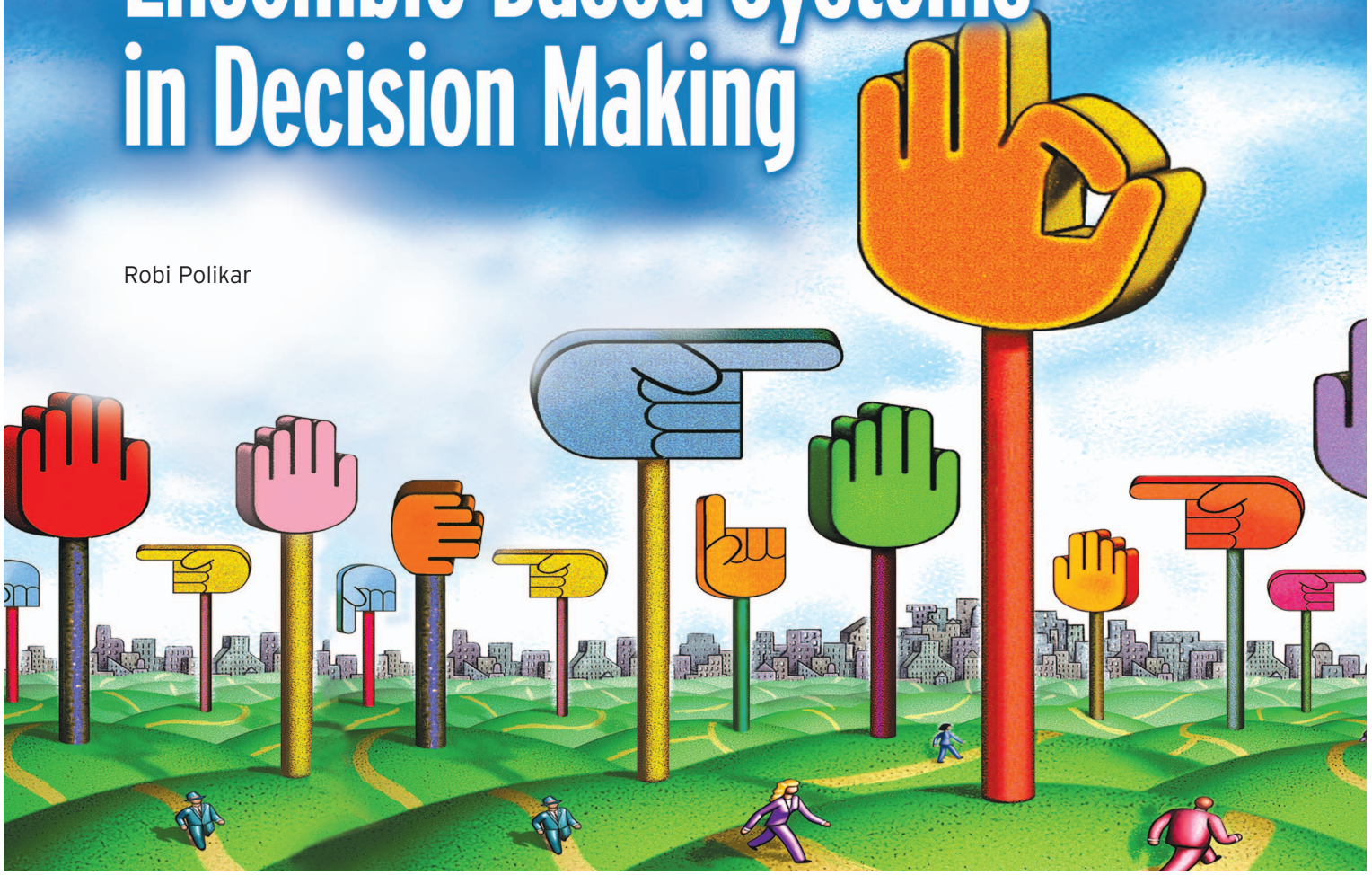
Feature

Ensemble Based Systems in Decision Making

Robi Polikar



Abstract

In matters of great importance that have financial, medical, social, or other implications, we often seek a second opinion before making a decision, sometimes a third, and sometimes many more. In doing so, we weigh the individual opinions, and combine them through some thought process to reach a final decision that is presumably the most informed one. The process of consulting "several experts" before making a final decision is perhaps second nature to us; yet, the extensive benefits of such a process in automated decision making applications have only recently been discovered by computational intelligence community.

Also known under various other names, such as multiple classifier systems, committee of classifiers, or mixture of experts, ensemble based systems have shown to produce favorable results compared to those of single-expert systems for a broad range of applications and under a variety of scenarios. Design, implementation and application of such systems are the main topics of this article. Specifically, this paper reviews conditions under which ensemble based systems may be more beneficial than their single classifier counterparts, algorithms for generating individual components of the ensemble systems, and various procedures through which the individual classifiers can be combined. We discuss popular ensemble based algorithms, such as bagging, boosting, AdaBoost, stacked generalization, and hierarchical mixture of experts; as well as commonly used combination rules, including algebraic combination of outputs, voting based techniques, behavior knowledge space, and decision templates. Finally, we look at current and future research directions for novel applications of ensemble systems. Such applications include incremental learning, data fusion, feature selection, learning with missing features, confidence estimation, and error correcting output codes; all areas in which ensemble systems have shown great promise.

Keywords: Multiple classifier systems, classifier combination, classifier fusion, classifier selection, classifier diversity, incremental learning, data fusion

1. Introduction

1.1. I'd like to Ask the Audience, Please. . .

A popular game show, aired in over 70 countries since 1999, drew the attention of hundreds of millions of TV viewers around the world. The novelty of the show was not so much in the talent it sought from the contestants; just like many of its predecessors, the show quizzed the knowledge of its contestants on a series of increasingly non-trivial topics. What made the show so popular was its previously unheard of large payout, but at the same time its rather unorthodox use of so-called "lifelines." If the contestant did not know the answer to the question, s/he could elect to have two incorrect answers be removed from the four possible choices, telephone a friend whom s/he thought to be knowledgeable on the subject matter, or simply poll the audience. The idea of giving such choices to the contestants was so intriguing that the question "which of the three choices is the best one for any given condition?" became a topic of light hearted conversation among many statistics and computational intelligence researchers.

The reader undoubtedly realizes where we are going with this: this article is about ensemble systems, and one of those three choices includes the use of an ensemble of decision makers: the audience. We will show that under relatively mild assumptions, asking the audience is in fact the best solution, and these assumptions provide clues for the specific scenarios in which ensemble systems may provide a clear advantage over single-expert systems.

As discussed below, there are several mathematically sound reasons for considering ensemble systems, but the intrinsic connection to our daily life experiences provides an undeniably strong psychological pretext: we use them all the time! Seeking additional opinions before making a decision is an innate behavior for most of us, particularly if the decision has important financial, medical or social consequences. Asking different doctors' opinions before undergoing a major surgery, reading user reviews before purchasing a product, or requesting references before hiring someone are just a few of countless number of examples where we consider the decisions of *multiple experts* in our daily lives. Our goal in doing so, of course, is to improve our confidence that we are making the right decision, by weighing various opinions, and combining them through some thought process to reach a final decision. Why not, then, follow the same process, and ask the opinion of additional experts in data analysis and automated decision making applications? Ensemble systems follow exactly such an approach to data analysis, and the design, implementation and applications of such systems constitute the main focus of this paper.

In this paper, the terms *expert*, *classifier* and *hypothesis* are used interchangeably: the goal of an expert is to make a decision, by choosing one option from a previously defined set of options. This process can be cast as a classification problem: the expert (classifier) makes a hypothesis about the classification of a given data instance into one of predefined categories that represent different decisions. The decision is based on prior training of the classifier, using a set of representative training data, for which the correct decisions are a *priori* known.

1.2. Reasons for Using Ensemble Based Systems

There are several theoretical and practical reasons why we may prefer an ensemble system:

Statistical Reasons: Readers familiar with neural networks or other automated classifiers are painfully aware that good performance on training data does not predict good generalization performance, defined as the performance of the classifier on data not seen during training. A set of classifiers with similar training performances may have different generalization performances. In fact, even classifiers with similar generalization performances may perform differently in the field, particularly if the test dataset used to determine the generalization performance is not sufficiently representative of the future field data. In such cases, combining the outputs of several classifiers by averaging may reduce the risk of an unfortunate selection of a poorly performing classifier. The averaging may or may not beat the performance of the best classifier in the ensemble, but it certainly reduces the overall risk of making a particularly poor selection. This is precisely the reason we consult the opinions of other experts: having several doctors agree on a diagnosis, or several former users agree on the quality (or lack thereof) of a product, reduces the risk of following the advice of a single doctor (or a single user) whose specific experience may be significantly different than those of others.

Large Volumes of Data: In certain applications, the amount of data to be analyzed can be too large to be effectively handled by a single classifier. For example, inspection of gas transmission pipelines using magnetic flux leakage techniques generates 10 GB of data for every 100 km of pipeline, of which there are over 2 million km. Training a classifier with such a vast amount of data is usually not practical; partitioning the data into smaller subsets, training different classifiers with different partitions of data, and combining their outputs using

Robi Polikar is with Electrical and Computer Engineering, Rowan University, Glassboro, NJ 08028 USA.

an intelligent combination rule often proves to be a more efficient approach.

Too Little Data: Ensemble systems can also be used to address the exact opposite problem of having too little data. Availability of an adequate and representative set of training data is of paramount importance for a classification algorithm to successfully learn the underlying data distribution. In the absence of adequate training data, resampling techniques can be used for drawing overlapping random subsets of the available data, each of which can be used to train a different classifier, creating the ensemble. Such approaches have also proven to be very effective.

Divide and Conquer: Regardless of the amount of available data, certain problems are just too difficult for a given classifier to solve. More specifically, the decision boundary that separates data from different classes may be too complex, or lie outside the space of functions that can be implemented by the chosen classifier model. Consider the two dimensional, two-class problem with a complex decision boundary depicted in Figure 1. A linear classifier, one that is capable of learning linear boundaries, cannot learn this complex non-linear boundary. However, appropriate combination of an ensemble of such linear classifiers can learn this (or any other, for that matter) non-linear boundary.

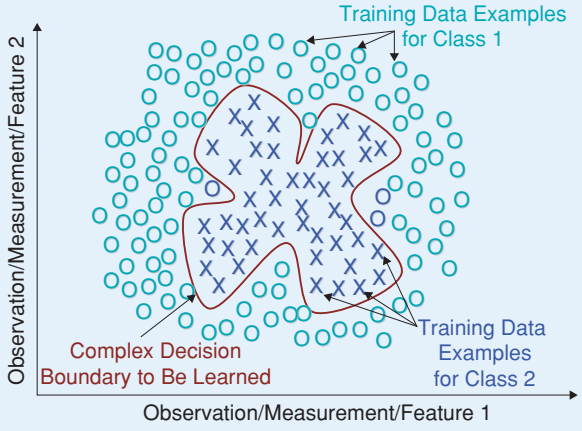


Figure 1. Complex decision boundary that cannot be learned by linear or circular classifiers.

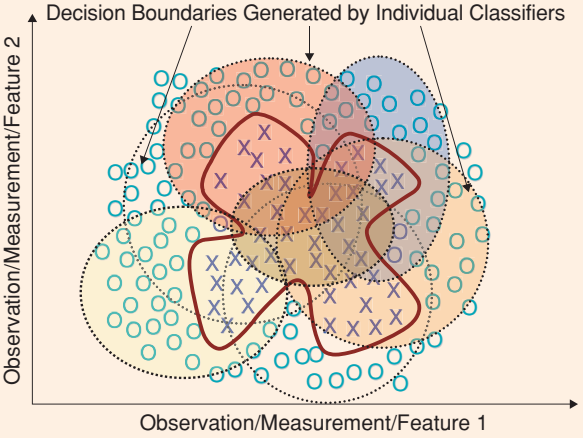


Figure 2. Ensemble of classifiers spanning the decision space.

As an example, let us assume that we have access to a classifier model that can generate elliptic/circular shaped boundaries. Such a classifier cannot learn the boundary shown in Figure 1. Now consider a collection of circular decision boundaries generated by an ensemble of such classifiers as shown in Figure 2, where each classifier labels the data as class1 (O) or class 2 (X), based on whether the instances fall within or outside of its boundary. A decision based on the majority voting of a sufficient number of such classifiers can easily learn this complex non-circular boundary. In a sense, the classification system follows a divide-and-conquer approach by dividing the data space into smaller and easier-to-learn partitions, where each classifier learns only one of the simpler partitions. The underlying complex decision boundary can then be approximated by an appropriate combination of different classifiers.

Data Fusion: If we have several sets of data obtained from various sources, where the nature of features are different (heterogeneous features), a single classifier cannot be used to learn the information contained in all of the data. In diagnosing a neurological disorder, for example, the neurologist may order several tests, such as an MRI scan, EEG recording, blood tests, etc. Each test generates data with a different number and type of features, which cannot be used collectively to train a single classifier. In such cases, data from each testing modality can be used to train a different classifier, whose outputs can then be combined. Applications in which data from different sources are combined to make a more informed decision are referred to as *data fusion* applications, and ensemble based approaches have successfully been used for such applications.

There are many other scenarios in which ensemble base systems can be very beneficial; however, discussion on these more specialized scenarios require a deeper understanding of how, why and when ensemble systems work. The rest of this paper is therefore organized as follows. In Section 2, we discuss some of the seminal work that has paved the way for today's active research area in ensemble systems, followed by a discussion on *diversity*, a keystone and a fundamental strategy shared by all ensemble systems. We close Section 2 by pointing out that all ensemble systems must have two key components: an algorithm to generate the individual classifiers

of the ensemble, and a method for combining the outputs of these classifiers. Various algorithms for the former are discussed in Section 3, whereas different procedures for the latter are discussed in Section 4. In Section 5, we look at current and emerging directions in ensemble system research, including some novel applications, for which the ensemble systems are naturally suited. Section 6 completes our overview of ensemble systems with some concluding remarks.

2. Ensemble Based Systems

2.1. A Brief History of Ensemble Systems

Perhaps one of the earliest work on ensemble systems is Dasarathy and Sheela's 1979 paper [1], which discusses partitioning the feature space using two or more classifiers. In 1990, Hansen and Salamon showed that the generalization performance of a neural network can be improved using an ensemble of similarly configured neural networks.[2], while Schapire proved that a strong classifier in *probably approximately correct (PAC)* sense can be generated by combining weak classifiers through boosting [3], the predecessor of the suite of AdaBoost algorithms. Since these seminal works, research in ensemble systems have expanded rapidly, appearing often in the literature under many creative names and ideas. The long list includes composite classifier systems [1], mixture of experts [4], [5], stacked generalization [6], consensus aggregation [7], combination of multiple classifiers [8]–[12], change-glasses approach to classifier selection [13], dynamic classifier selection [12], classifier fusion [14]–[16], committees of neural networks [17], voting pool of classifiers [18], classifier ensembles [17], [19], and pandemonium system of reflective agents [20], among many others.

The paradigms of these approaches usually differ from each other with respect to the specific procedure used for generating individual classifiers, and/or the strategy employed for combining the classifiers. There are generally two types of combination: classifier selection and classifier fusion [12], [21]. In *classifier selection*, each classifier is trained to become an expert in some local area of the total feature space (as in Figure 2). The combination of the classifiers is then based on the given feature vector: the classifier trained with data closest to the vicinity of the feature vector, in some distance metric sense, is given the highest credit. One or more local experts can be nominated to make the decision [4], [12], [22–24]. In *classifier fusion*, however, all classifiers are trained over the entire feature space. In this case, the classifier combination process involves merging the individual (weaker) classifier designs to obtain a single (stronger) expert of superior performance. Examples of this approach include bagging predictors [25], boosting [3], [26] and its many variations. The combination may apply to classification labels only, or to the class-specific continuous valued outputs of the individual experts [16], [27], [28]. In the latter case, classifier outputs are often normalized to the [0, 1] interval, and these values are interpreted as the support given by the classifier to each class, or even as class-conditional posterior probabilities [16], [29]. Such interpretation allows forming an ensemble through algebraic combination rules (majority voting, maximum/minimum/sum/product or other combinations of posterior probabilities) [9], [28], [30], [31], fuzzy integral for combination [15], [32], [33], the Dempster-Shafer based classifier fusion [10], [34], and more recently, the decision templates [16], [27], [30], [35]–[37].

In other related efforts, input decimation has been proposed for naturally grouping different modalities for independent classifiers in [38], [39], and the conditions under which such combination of modalities work best is described in [40]. A number of authors have provided theoretical analyses of various strategies commonly used in multiple expert fusion: for example, theoretical models were developed in [41], [42] for combining discriminant functions; six commonly used combination rules were compared for their ability to predict posterior probabilities in [31]; and a Bayesian theoretical framework for multiple expert fusion was developed in [28], where the sensitivity of various combination schemes to estimation errors was analyzed to propose a plausible model that explains such behavior.

A sample of the immense literature on classifier combination can be found in Kuncheva's recent book [21], the first text devoted to theory and implementation of ensemble based classifiers, and references therein. The field has been developing so rapidly that an international workshop on multiple classifier systems (MCS) has recently been established, and the most current developments can be found in its proceedings [43].

2.2. Diversity: Cornerstone of Ensemble Systems

If we had access to a classifier with perfect generalization performance, there would be no need to resort to ensemble techniques. The realities of noise, outliers and overlapping data distributions, however, make such a classifier an impossible proposition. At best, we can hope for classifiers that correctly classify the field data *most of the time*. The strategy in ensemble systems is therefore to create many classifiers, and combine their outputs such that the combination improves upon the performance of a single classifier. This requires, however, that individual classifiers make errors on different instances. The intuition is that if each classifier makes different errors, then a strategic combination of these classifiers can reduce the total error, a concept not too dissimilar to low pass

filtering of the noise. The overarching principal in ensemble systems is therefore to make each classifier as unique as possible, particularly with respect to misclassified instances. Specifically, we need classifiers whose decision boundaries are adequately different from those of others. Such a set of classifiers is said to be *diverse*.

Classifier diversity can be achieved in several ways. The most popular method is to use different training datasets to train individual classifiers. Such datasets are often obtained through resampling techniques, such as bootstrapping or bagging, where training data subsets are drawn randomly, usually with replacement, from the entire training data. This is illustrated in Figure 3, where random and overlapping training data subsets are selected to train three classifiers, which then form three different decision boundaries. These boundaries are combined to obtain a more accurate classification.

To ensure that individual boundaries are adequately different, despite using substantially similar training data, *unstable classifiers* are used as base models, since they can generate sufficiently different decision boundaries even for small perturbations in their training parameters. If the training data subsets are drawn without replacement, the procedure is also called jackknife or *k*-fold data split: the entire dataset is split into *k* blocks, and each classifier is trained only on *k*-1 of them. A different subset of *k* blocks is selected for each classifier as shown in Figure 4.

Another approach to achieve diversity is to use different training parameters for different classifiers. For example, a series of multilayer perceptron (MLP) neural networks can be trained by using different weight initializations, number of layers/nodes, error goals, etc. Adjusting such parameters allows one to control the instability of the individual classifiers, and hence contribute to their diversity. The ability to control the instability of neural network and decision tree type classifiers make them suitable candidates to be used in an ensemble

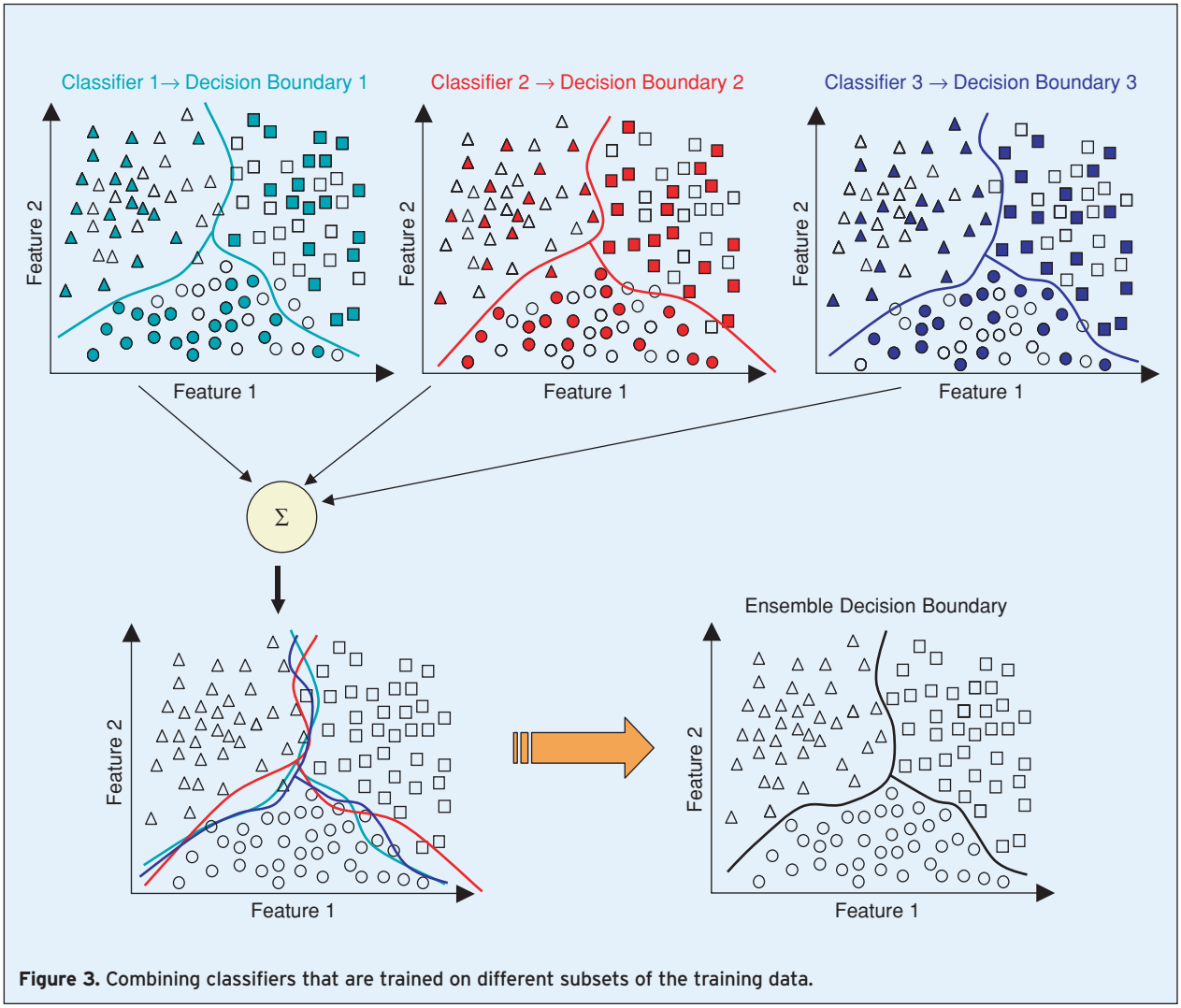


Figure 3. Combining classifiers that are trained on different subsets of the training data.

setting. Alternatively, entirely different type of classifiers, such MLPs, decision trees, nearest neighbor classifiers, and support vector machines can also be combined for added diversity. However, combining different models, or even different architectures of the same model, is used only for specific applications that warrant them. Diversity is typically obtained through resampling of the training data, as this procedure is theoretically more tractable. Finally, diversity can also be achieved by using different features. In fact, generating different classifiers using random feature subsets is known as the *random subspace method* [44], and it has found widespread use in certain applications, which are discussed later in future research areas.

2.3. Measures of Diversity

Several measures have been defined for quantitative assessment of diversity. The simplest ones are pair-wise measures, defined between two classifiers. For T classifiers, we can calculate $T(T\text{-}1)/2$ pair-wise diversity measures, and an overall diversity of the ensemble can be obtained by averaging these pair-wise measures. Given two hypotheses h_i and h_j, we use the notations

	h_j is correct	h_j is incorrect
h_i is correct	a	b
h_i is incorrect	c	d

where a is the fraction of instances that are correctly classified by both classifiers, b is the fraction of instances correctly classified by h_i but incorrectly classified by h_j, and so on. Of course, $a+b+c+d=\mathbf{1}$. Then, the following pair-wise diversity measures can be defined:

Correlation Diversity is measured as the correlation between two classifier outputs, defined as

$$\rho_{i,j}=\frac{ad-bc}{\sqrt{(a+b)\,(c+d)\,(a+c)\,(b+d)}},\quad 0\le\rho\le 1. \qquad (1)$$

Maximum diversity is obtained for $\rho=0$, indicating that the classifiers are uncorrelated.

Q-Statistic Defined as

$$Q_{i,j}=(ad-bc)/(ad+bc) \qquad (2)$$

Q assumes positive values if the same instances are correctly classified by both classifiers; and negative values, otherwise. Maximum diversity is, once again, obtained for $Q=0$.

Disagreement and Double Fault Measures The disagreement is the probability that the two classifiers will disagree, whereas the double fault measure is the

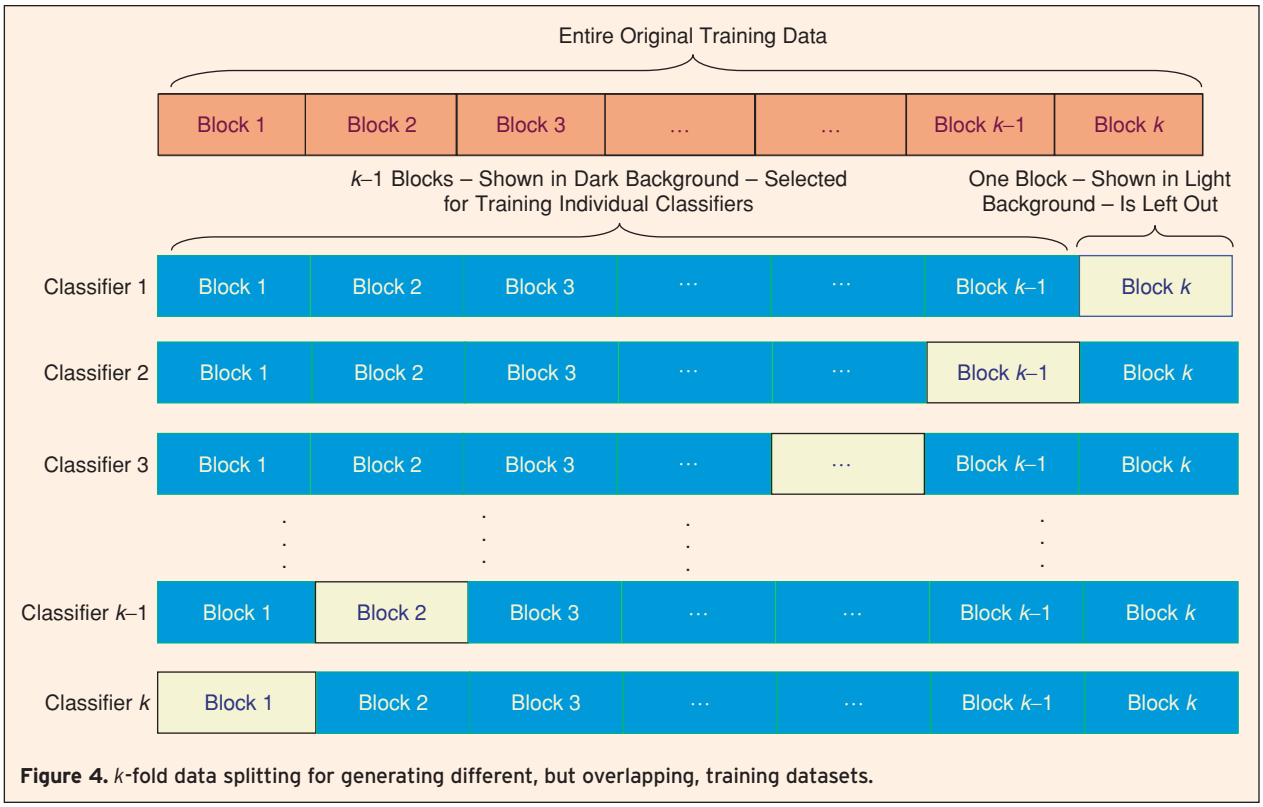


Figure 4. k-fold data splitting for generating different, but overlapping, training datasets.

probability that both classifiers are incorrect. The diversity increases with both the disagreement and the double fault value.

$$D_{i,j} = b + c, \tag{3}$$

$$DF_{i,j} = d. \tag{4}$$

There are also non pair-wise measures that take the ensemble's decision into consideration.

Entropy Measure makes the assumption that the diversity is highest if half of the classifiers are correct, and the remaining ones are incorrect. Based on this assumption, and defining ζ_i as the number of classifiers, out of T, that misclassifies instance $\mathbf{x}_i$, the entropy measure is defined as

$$E = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{T - \lceil T/2 \rceil}\min\{\zeta_i, (T - \zeta_i)\} \tag{5}$$

where $\lceil \cdot \rceil$ is the ceiling operator, and N is the dataset cardinality. Entropy varies between 0 and 1: 0 indicates that all classifiers are practically the same, and 1 indicates highest diversity [45].

Kohavi–Wolpert Variance defined as

$$KW = \frac{1}{NT^2}\sum_{i=1}^{N}\zeta_i \cdot (T - \zeta_i) \tag{6}$$

follows a similar approach to the disagreement measure, and can be shown to be related to the disagreement measure averaged over all classifiers by a constant factor [45].

Measure of difficulty, originally proposed in [2], uses the random variable $Z(\mathbf{x}_i) = \{0, 1/T, 2/T, \cdots, 1\}$, defined as the fraction of classifiers that misclassify $\mathbf{x}_i$. The measure of difficulty, θ, is then the variance of the random variable Z:

$$\theta = \frac{1}{T}\sum_{t=0}^{T}(z_t - \bar{z})^2 \tag{7}$$

where $\bar{z}$ is the mean of z; hence it is the average fraction of classifiers that misclassify any given input. Hansen and Salamon argue that if classifiers are positively correlated, that is, they all tend to correctly classify the same instances, and misclassify same other instances, then θ is large and there is little diversity. In the extreme case, where all classifiers are identical, the distribution of Z will be two singletons: one at 0, with a mass proportional to the number of missed instances, and the other at 1, with a mass proportional to the number of correctly classified instances. Then, Z attains its maximum value. However, if classifiers are negatively correlated, that is, they tend to misclassify different instances, then θ is relatively small, and there is more diversity.

Several other diversity measures have also been proposed, such as measurement of interrater agreement or generalized diversity, whose comparisons can be found in [45], [46]. Kuncheva's conclusion, reminiscent of the *No Free Lunch* theorem (which states that no classification algorithm is universally superior to others [47]), is that there is no diversity measure that consistently correlates with higher accuracy. In the absence of additional information, she suggests the Q statistic because of its intuitive meaning and simple implementation.

2.4. Two Key Components of an Ensemble System

All ensemble systems consist of two key components. First, a strategy is needed to build an ensemble that is as diverse as possible. Some of the more popular ones, such as bagging, boosting, AdaBoost, stacked generalization, and mixture of experts are discussed in Section 3. A second strategy is needed to combine the outputs of individual classifiers that make up the ensemble in such a way that the correct decisions are amplified, and incorrect ones are cancelled out. Several choices are available for this purpose as well, which are discussed in Section 4.

3. Creating an Ensemble

Two interrelated questions need to be answered in designing an ensemble system: i) how will individual classifiers (base classifiers) be generated? and ii) how will they differ from each other? The answers ultimately determine the diversity of the classifiers, and hence affect the performance of the overall system. Therefore, any strategy for generating the ensemble members must seek to improve the ensemble's diversity. In general, however, ensemble algorithms do not attempt to maximize a specific diversity measure (see [46], [48] for exceptions). Rather, increased diversity is usually sought—somewhat heuristically—through various resampling procedures or selection of different training parameters. The above defined diversity measures can then be used to compare the diversities of the ensembles generated by different algorithms.

3.1. Bagging

Breiman's *bagging*, short for *bootstrap aggregating*, is one of the earliest ensemble based algorithms. It is also one of the most intuitive and simplest to implement, with a surprisingly good performance [25]. Diversity in bagging is obtained by using bootstrapped replicas of the training data: different training data subsets are randomly drawn—with replacement—from the entire training data. Each training data subset is used to train a different classifier of the same type. Individual classifiers are then combined by taking a majority vote of their decisions. For any given instance, the class chosen by most classifiers is the ensemble decision.

Bagging is particularly appealing when available data is of limited size. To ensure that there are sufficient training

samples in each subset, relatively large portions of the samples (75% to 100%) are drawn into each subset. This causes individual training subsets to overlap significantly, with many of the same instances appearing in most subsets, and some instances appearing multiple times in a given subset. In order to ensure diversity under this scenario, a relatively *unstable* model is used so that sufficiently different decision boundaries can be obtained for small perturbations in different training datasets. As mentioned above, neural networks and decision trees are good candidates for this purpose, as their instability can be controlled by the selection of their free parameters. The pseudocode for the bagging algorithm is given in Figure 5.

3.2. Variations of Bagging

Random Forests: A variation of the bagging algorithm is the *Random Forests*, so-called because it is constructed from decision trees [49]. A random forest can be created from individual decision trees, whose certain training parameters vary randomly. Such parameters can be bootstrapped replicas of the training data, as in bagging, but they can also be different feature subsets as in random subspace methods.

Pasting Small Votes: Unlike bagging, *pasting small votes* is designed to be used with large datasets [50]. A large dataset is partitioned into smaller subsets, called *bites*, each of which is used to train a different classifier. Two variations of pasting small votes have emerged: one that creates the data subsets at random, called *Rvotes*, and one that creates consecutive datasets based on the importance of the instances, called *Ivotes*.

The latter approach is known to provide better results [51], and is similar to the approach followed by boosting based algorithms, where each classifier focuses on *most important* (or *most informative*) instances for the current ensemble member.

Basically, the important instances are those that improve diversity; and one way to do so is to train each classifier on a dataset that consists of a balanced distribution of *easy* and *difficult* instances. This is achieved by evaluating the current ensemble E_t, consisting of t classifiers, on instances that have not yet been used for training. For any given instance $\mathbf{x}$, those classifiers that did not use $\mathbf{x}$ in their training are called *out-of-bag* classifiers. If $\mathbf{x}$ is misclassified by a simple majority vote of the current ensemble, then it is automatically placed in the training subset of the next classifier. Otherwise, it is still placed in the training set, but only with probability $\varepsilon_t/(1-\varepsilon_t)$, where $0<\varepsilon_t<1/2$ is the error of the t^{th} classifier. Individual classifiers are expected to perform at least 50% to ensure that a meaningful performance can be provided by each classifier. In Section 4, within the context of voting algorithms, we will see that the error threshold of 0.5 is in fact a strategically chosen value. The complete algorithm for pasting small votes is given in Figure 6.

Algorithm: Bagging

Input:

- Training data S with correct labels $\omega_i \in \Omega=\{\omega_1,...,\omega_C\}$ representing C classes
- Weak learning algorithm **WeakLearn,**
- Integer T specifying number of iterations.
- Percent (or fraction) F to create bootstrapped training data

Do $t = 1, \ldots, T$

1. Take a bootstrapped replica S_t by randomly drawing F percent of S.
2. Call **WeakLearn** with S_t and receive the hypothesis (classifier) h_t.
3. Add h_t to the ensemble, E.

End

Test: Simple Majority Voting – Given unlabeled instance $\mathbf{x}$

1. Evaluate the ensemble $\mathrm{E}=\{h_1,\ldots,h_T\}$ on $\mathbf{x}$.
2. Let $v_{t,j}=\begin{cases}1, & \text{if } h_t \text{ picks class } \omega_j\\ 0, & \text{otherwise}\end{cases}$ (8)

   be the vote given to class ω_j by classifier h_t.
3. Obtain total vote received by each class

$$V_j=\sum_{t=1}^{T} v_{t,j},\ j=1,\ldots,C \quad (9)$$

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

Figure 5. The bagging algorithm.

3.3. Boosting

In 1990, Schapire proved that *a weak learner*, an algorithm that generates classifiers that can merely do better than random guessing, can be turned into a *strong learner* that generates a classifier that can correctly classify all but an arbitrarily small fraction of the instances [3]. Formal definitions of weak and strong learner, as defined in the PAC learning frame work, can be found in [3], where Schapire also provides an elegant algorithm for boosting the performance of a weak learner to the level of a strong one. Hence called *boosting*, the algorithm is now considered as one of the most important developments in the recent history of machine learning.

Similar to bagging, boosting also creates an ensemble of classifiers by resampling the data, which are then combined by majority voting. However, similarities end there. In boosting, resampling is strategically geared to provide the most informative training data for each consecutive

Algorithm: Pasting Small Votes (Ivotes)

Input:

- Training data S with correct labels $\omega_i \in \Omega = \{\omega_1,..., \omega_C\}$ representing C classes;
- Weak learning algorithm **WeakLearn;**
- Integer T specifying number of iterations;
- *Bitesize M*, indicating the size of individual training subsets to be created.

Initialize

1. Choose a random subset S_0 of size M from S.
2. Call **WeakLearn** with S_0, and receive the hypothesis (classifier) h_0.
3. Evaluate h_0 on a validation dataset, and obtain error ε_0 of h_0.
4. If $\varepsilon_0 > 1/2$, return to step 1.

Do $t=1, \ldots, T$

1. Randomly draw an instance **x** from S according to uniform distribution.
2. Evaluate **x** using majority vote of out-of-bag classifiers in the current ensemble E_t.
3. If **x** is misclassified, place **x** in S_t. Otherwise, place **x** in S_t with probability p

$$p = \frac{\varepsilon_{t-1}}{(1-\varepsilon_{t-1})}. \qquad (10)$$

   Repeat Steps 1-3 until S_t has M such instances.
4. Call **WeakLearn** with S_t and receive the hypothesis h_t.
5. Evaluate h_t on a validation dataset, and obtain error ε_t of h_t. If $\varepsilon_t > 1/2$, return to step 4.
6. Add h_t to the ensemble to obtain E_t.

End

Test – Use simple majority voting on test data.

Figure 6. Pasting small votes (Ivotes) algorithm.

Algorithm: Boosting

Input:

- Training data S of size N with correct labels $\omega_i \in \Omega = \{\omega_1, \omega_2\}$;
- Weak learning algorithm **WeakLearn**.

Training

1. Select $N_1 < N$ 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 the third classifier C_3 with S_3.

Test – Given a test instance **x**

1. Classify **x** 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.

Figure 7. The boosting algorithm.

classifier. In essence, boosting creates three weak classifiers: the first classifier C_1 is trained with a random subset of the available training data. The training data subset for the second classifier C_2 is chosen as the most informative subset, given C_1. That is, C_2 is trained on a training data only half of which is correctly classified by C_1, and the other half is misclassified. The third classifier C_3 is trained with instances on which $C_1$and C_2 disagree. The three classifiers are combined through a three-way majority vote. The algorithm is shown in detail in Figure 7.

Schapire has shown that the error of this three-classifier ensemble is bounded above, and it is less than the error of the best classifier in the ensemble, provided that each classifier has an error rate that is less than 0.5. For a two-class problem, an error rate of 0.5 is the least we can expect from a classifier, as an error of 0.5 amounts to random guessing. Hence, a stronger classifier is generated from three weaker classifiers. A strong classifier in the strict PAC learning sense can then be created by recursive applications of boosting.

3.4. AdaBoost

In 1997, Freund and Schapire introduced AdaBoost [26], which has since enjoyed a remarkable attention, one that is rarely matched in computational intelligence. AdaBoost is a more general version of the original boosting algorithm. Among its many variations, AdaBoost.M1 and AdaBoost.R are more commonly used, as they are capable of handling multiclass and regression problems, respectively. In this paper, we discuss AdaBoost.M1 in detail, and provide a brief survey and references for other algorithms that are based on AdaBoost.

AdaBoost generates a set of hypotheses, and combines them through weighted majority voting of the classes predicted by the individual hypotheses. The hypotheses are generated by training a weak classifier, using instances drawn from an iteratively updated distribution of the training data. This distribution update ensures that instances misclassified by the previous classifier are more likely to be included in the training data of the next classifier. Hence, consecutive classifiers' training data are geared towards increasingly hard-to-classify instances.

The pseudocode of the algorithm is provided in Figure 8. Several interesting features of the algorithm are worth noting. The algorithm maintains a weight distribution $D_t(i)$ on training instances $\mathbf{x}_i$, $i = 1, \ldots, N$, from which training data subsets S_t are chosen for each consecutive classifier (hypothesis) h_t. The distribution is initialized to be uniform, so that all instances have equal likelihood to be selected into the first training dataset. The training error ε_t of classifier h_t is also weighted by this distribution, such that ε_t is the sum of distribution weights of the instances misclassified by h_t (Equation 12). As before, we require that this error be less than $1/2$. A normalized error is then obtained as β_t, such that for $< 0\, \varepsilon_t < 1/2$, we have $0 < \beta_t < 1$.

Equation 14 describes the distribution update rule: the distribution weights of those instances that are correctly classified by the current hypothesis are reduced by a factor of β_t, whereas the weights of the misclassified instances are unchanged. When the updated weights are renormalized, so that D_{t+1} is a proper distribution, the weights of the misclassified instances are effectively increased. Hence, iteration by iteration, AdaBoost focuses on increasingly difficult instances. Note that AdaBoost raises the weights of instanced misclassified by h_t so that they add up to $1/2$, and lowers the weights of correctly classified instances, so that they too add up to $1/2$. Since the base model learning algorithm **WeakLearn** is required to have an error less than $1/2$, it is guaranteed to correctly classify at least one previously misclassified training example. Once a preset T number of classifiers are generated, AdaBoost is ready for classifying unlabeled test instances. Unlike bagging or boosting, AdaBoost uses a rather undemocratic voting scheme, called the *weighted majority voting*. The idea is an intuitive one: those classifiers that have shown good performance during training are rewarded with higher voting weights than the others. Recall that a normalized error β_t was calculated in Equation 13. The reciprocal of this quantity, $1/\beta_t$ is therefore a measure of performance, and can be used to weight the classifiers. Furthermore, since β_t is training error, it is often close to zero and $1/\beta_t$ can therefore be a very large number. To avoid potential instability that can be caused by asymptotically large numbers, the logarithm of $1/\beta_t$ is usually used as the voting weight of h_t. At the end, the class that receives the highest total vote from all classifiers is the ensemble decision.

Algorithm AdaBoost.M1

Input:

- Sequence of N examples $S = [(\mathbf{x}_i, y_i)]$, $i = 1, \cdots, N$ with labels $y_i \in \Omega$, $\Omega = \{\omega_1, \ldots, \omega_C\}$;
- Weak learning algorithm **WeakLearn**;
- Integer T specifying number of iterations.

Initialize $D_1(i) = \frac{1}{N}., i = 1, \cdots, N$ (11)

Do for $t = 1, 2, \ldots, T$:

1. Select a training data subset S_t, drawn from the distribution D_t.
2. Train **WeakLearn** with S_t, receive hypothesis h_t.
3. Calculate the error of h_t: $\varepsilon_t = \sum_{i:h_t(\mathbf{x}_i)\neq y_i} D_t(i)$. (12)

   If $\varepsilon_t > 1/2$, **abort**.
4. Set $\beta_t = \varepsilon_t/(1-\varepsilon_t)$. (13)
5. Update distribution

$$D_t : D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(\mathbf{x}_i) = y_i \\ 1, & \text{otherwise} \end{cases} \quad (14)$$

   where $Z_t = \sum_i D_t(i)$ is a normalization constant chosen so that D_{t+1} becomes a proper distribution function.

Test – Weighted Majority Voting: Given an unlabeled instance $\mathbf{x}$,

1. Obtain total vote received by each class

$$V_j = \sum_{t:h_t(\mathbf{x})=\omega_j} \log \frac{1}{\beta_t}, j = 1, \ldots, C. \quad (15)$$

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

Figure 8. The AdaBoost.M1 algorithm.

A conceptual block diagram of the algorithm is provided in Figure 9. The diagram should be interpreted with the understanding that the algorithm is sequential: classifier C_K is created before classifier C_{K+1}, which in turn requires that β_K and the current distribution D_K be available.

Freund and Schapire also showed that the training error of AdaBoost.M1 is bounded above:

$$E < 2^T \prod_{t=1}^{T} \sqrt{\varepsilon_t (1-\varepsilon_t)} \quad (16)$$

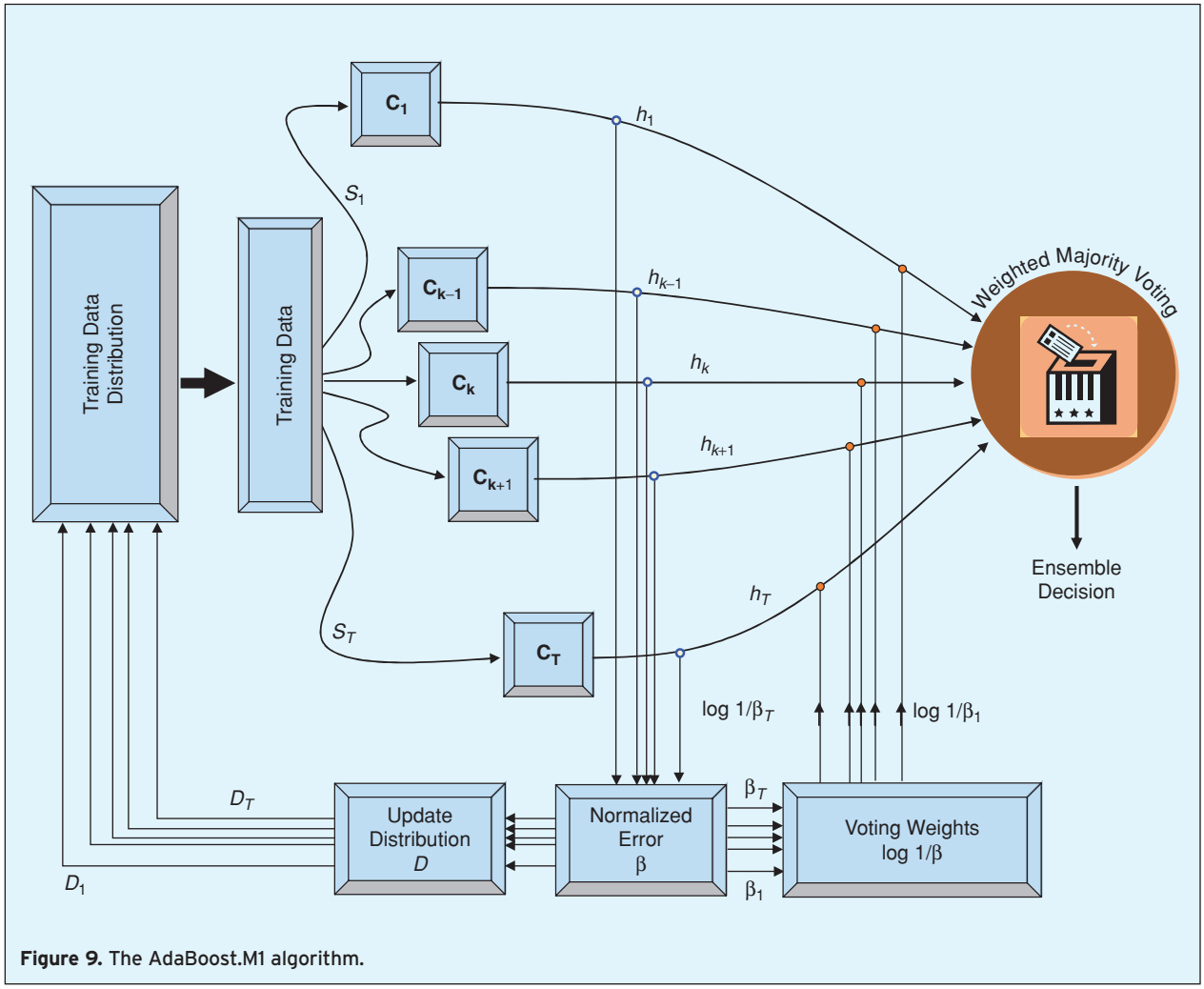


Figure 9. The AdaBoost.M1 algorithm.

where E is the ensemble error [26]. Since $\varepsilon_t < 1/2$, E is guaranteed to decrease with each new classifier. In most practical cases, the error decreases very rapidly in the first few iterations, and approaches zero as new classifiers are added. While this is remarkable on its own account, the surprising resistance of AdaBoost to overfitting is particularly noteworthy. Overfitting is a commonly observed phenomenon where the classifier performs poorly on test data, despite achieving a very small training error. Overfitting is usually attributed to memorizing the data, or learning the noise in the data. As a classifier's capacity increases (for example, with the complexity of its architecture), so does its tendency to memorize the training data and/or learn the noise in the data. Since the capacity of an ensemble increases with each added classifier, one would expect AdaBoost to suffer from overfitting, particularly if its complexity exceeds what is necessary to learn the underlying data distribution. Yet, AdaBoost performance usually levels off with increasing number of classifiers with no indication of overfitting.

Schapire *et al.* later provided an explanation to this phenomenon based on the so-called *margin theory* [52]. The details of margin theory are beyond the scope of this paper; however, the margin of an instance **x**, in loose terms, is its distance from the decision boundary. The further away an instance is from the boundary, the larger its margin, and hence the higher the confidence of the classifier in correctly classifying this instance. Margins are usually described within the context of support vector machine (SVM) type classifiers, which are based on the idea of maximizing the margins between the instances and the decision boundary. In fact, SVMs find those instances, called support vectors, that lie closest to the decision boundary. The support vectors are said to *define the margin* that separates the classes. The concept of a margin is conceptually illustrated in Figure 10.

By using a slightly different definition of a margin—suitably modified for AdaBoost—Schapire *et al.* show that AdaBoost also *boosts* the margins, that is, it finds the decision boundary that is further away from the instances of all classes. In the context of AdaBoost, the margin of an

instance is simply the difference between the total vote it receives from correctly identifying classifiers and the maximum vote received by any incorrect class. They show that the ensemble error is bounded with respect to the margin, but is independent of the number of classifiers [52]. While the bound is loose, the generalization error decreases with increasing margin, and AdaBoost maximizes these margins by increasing the confidence of the ensemble's decision.

Several variations of AdaBoost are available, a couple of which are proposed by Freund and Schapire themselves: AdaBoost.M2 removes the requirement that each individual hypothesis should maintain a weighted error less than $1/2$ (for multiclass problems, $\varepsilon_t < 1/2$ is sufficient, but not required), whereas AdaBoost.R extends the boosting approach to regression type problems [26]. There are also more heuristic variations that modify either the distribution update rule or the combination rule of the classifiers. For example, AveBoost averages the distribution weights to make the errors of each hypothesis as uncorrelated as possible with those of the previous ones (a key goal for achieving diversity) [53], [54], whereas Learn^{++} makes the distribution update rule contingent on the ensemble error (instead of the previous hypothesis' error) to allow for efficient incremental learning of new data that may introduce new classes [55].

3.5. Stacked Generalization

Certain instances may have a high likelihood of being misclassified because, for example, they are very close to the decision boundary, and hence often fall on the wrong side of the boundary approximated by the classifier. Conversely, certain instances may have a high likelihood of being correctly classified, because they are primarily far away from—and on the correct side of—their respective decision boundaries. A natural question arises: can we learn that certain classifiers consistently correctly classify, or consistently misclassify, certain instances? Or more specifically, given an ensemble of classifiers operating on a set of data drawn from an unknown but fixed distribution, can we map the outputs of these classifiers to their true classes?

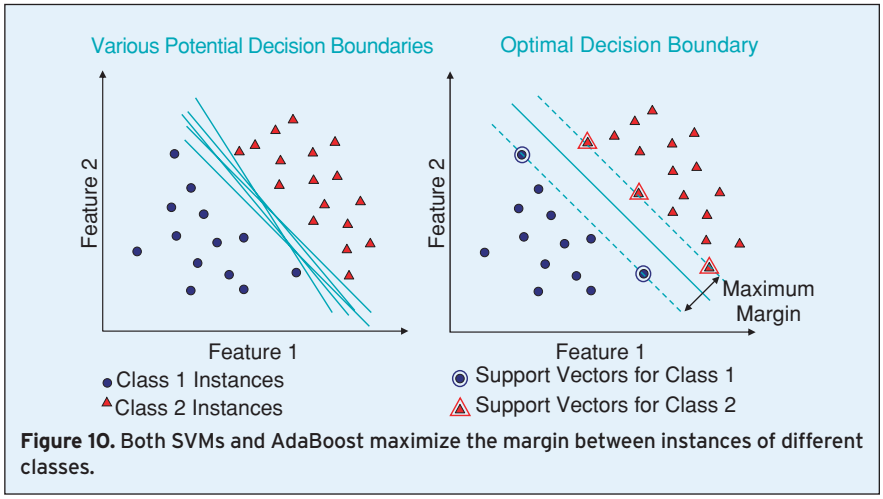


Figure 10. Both SVMs and AdaBoost maximize the margin between instances of different classes.

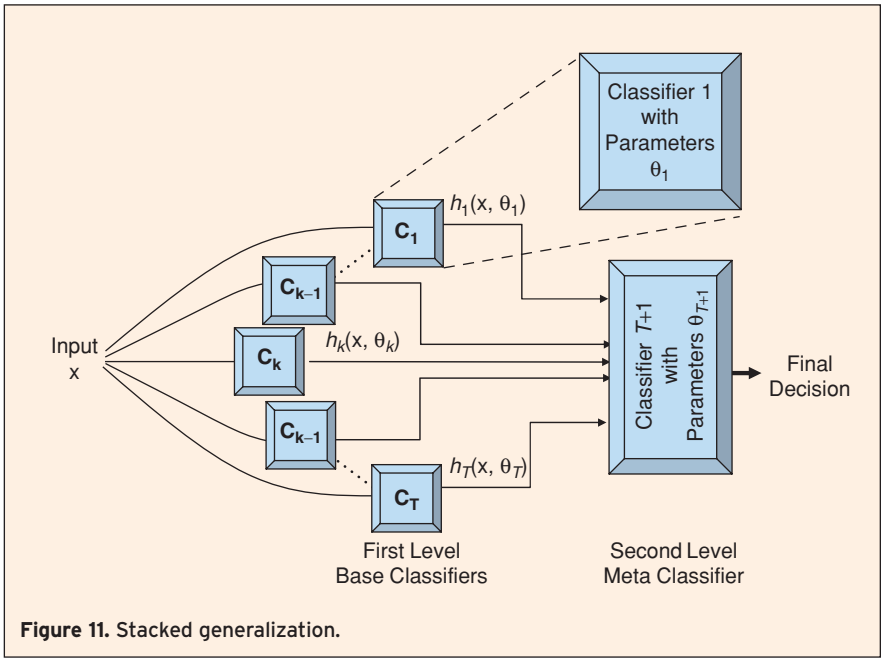


Figure 11. Stacked generalization.

In Wolpert's stacked generalization, an ensemble of classifiers are first created, whose outputs are used as inputs to a second level meta-classifier to learn the mapping between the ensemble outputs and the actual correct classes [6]. Figure 11 illustrates the stacked generalization approach, where classifiers $C_1, \ldots, C_T$ are trained using training parameters θ_1 through θ_T (where may include different training datasets, classifier architectural parameters, etc.) to output hypotheses h_1 through h_T. The outputs of these classifiers and the corresponding true classes are then used as input/output training

pairs for the second level classifier, C_{T+1}.Traditionally, the *k*-fold selection process described in Section 2 is used to obtain the training data for classifier C_{T+1}. Specifically, the entire training dataset is divided into T blocks, and each first-level classifier $C_1, \ldots, C_T$ is first trained on (a different set of) $T-1$ blocks of the training data. Therefore, there is one block of data not seen by each of the classifiers C_1through C_T. The outputs of each classifier for the block of instances on which it was not trained, along with the correct labels of those instances, constitute the training data for the second level meta-classifier C_{T+1}. Once C_{T+1} is trained, all data are pooled, and individual classifiers $C_1, \ldots C_T$ are retrained on the entire database, using a suitable resampling method.

3.6. Mixture-of-Experts

A conceptually similar technique is the *mixture-of-experts* model [4], where a set of classifiers $C_1, \ldots, C_T$ constitute the ensemble, followed by a second level classifier C_{T+1} used for assigning weights for the consecutive combiner. Figure 12 illustrates this model, where the combiner itself is usually not a classifier, but rather a simple combination rule, such as random selection (from a weight distribution), weighted majority, or weighted winner-takes-all. However, the weight distribution used for the combiner is determined by a second level classifier, usually a neural network, called the *gating network*. The gating network is trained either through standard gradient descent based backpropagation, or more commonly, through the expectation maximization (EM) algorithm [5], [56]. In either case, the inputs to the gating network are the actual training data instances themselves (unlike outputs of first level classifiers for stacked generalization), hence the weights used in combination rule are instance specific, creating a dynamic combination rule.

Mixture-of-experts can therefore be seen as a classifier selection algorithm. Individual classifiers are experts in some portion of the feature space, and the combination rule selects the most appropriate classifier, or classifiers weighted with respect to their expertise, for each instance **x**. The pooling system may use the weights in several different ways: it may choose a single classifier with the highest weight, or calculate a weighted sum of the classifier outputs for each class, and pick the class that receives the highest weighted sum. The latter approach requires that the classifier outputs be continuous-valued for each class.

4. Combining Classifiers

The second key component of any ensemble system is the strategy employed in combining classifiers. Combination rules are often grouped as (i) trainable vs. non-trainable combination rules, or (ii) combination rules that apply to class labels vs. to class-specific continuous outputs.

In trainable combination rules, the parameters of the combiner, usually called *weights*, are determined through a separate training algorithm. The EM algorithm used by the mixture-of-experts model is such an example. The combination parameters created by trainable rules are usually instance specific, and hence are also called dynamic combination rules. Conversely, there is no separate training involved in non-trainable rules beyond that used for generating the ensembles. Discussed below, weighted majority voting is an example of such non-trainable rules, since the parameters become immediately available as the classifiers are generated.

In the second taxonomy, combination rules that apply to class labels need the classification decision only (that is, one of ω_j, $j = 1, \ldots, C$), whereas others need the continuous-valued outputs of individual classifiers. These values often represent the degrees of support the classifiers give to each class, and they can estimate class

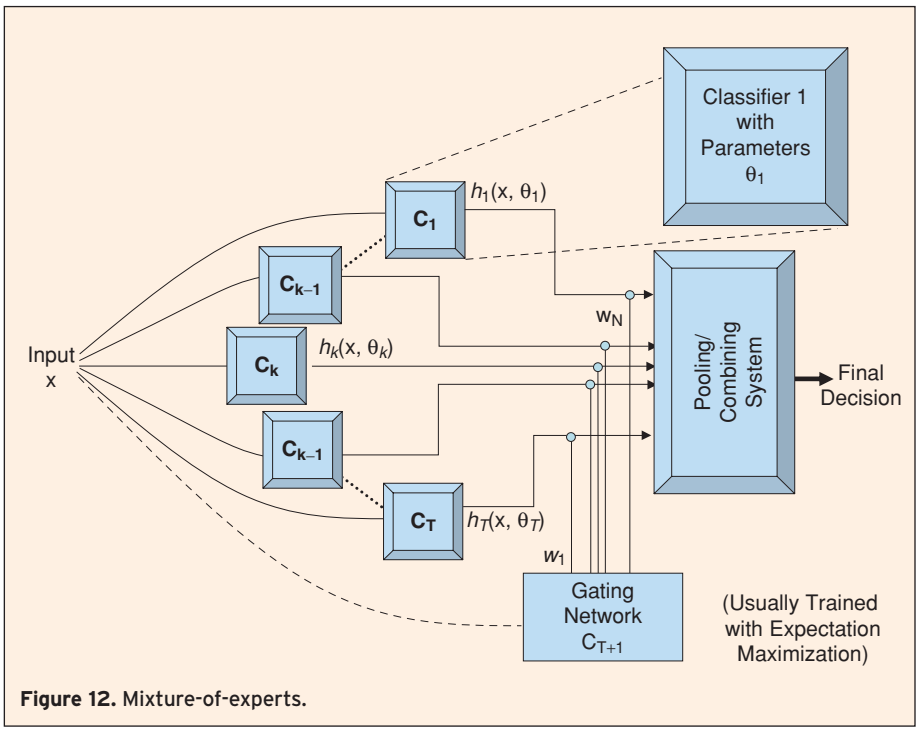


Figure 12. Mixture-of-experts.

conditional posterior probabilities $P(\omega_j|\mathbf{x})$ if (i) they are appropriately normalized to add up to 1 for all classes, and (ii) if the classifiers are trained with sufficiently dense data. Many classifier models, such as the MLP and the RBF networks, provide continuous-valued outputs, which are often interpreted as posterior probabilities—despite the sufficiently dense training data requirement rarely being met in practice.

In this paper, we follow the second grouping: we first review combination rules that apply to class labels, followed by those that combine class-specific continuous outputs.

4.1. Combining Class Labels

In the following discussion, we assume that only the class labels are available from the classifier outputs. Let us define the decision of the t^{th}classifier as $d_{t,j} \in \{0, 1\}$, $t = 1 \ldots, T$ and $j = 1, \ldots C$, where T is the number of classifiers and C is the number of classes. If t^{th} classifier chooses class ω_j, then $d_{t,j} = 1$, and 0, otherwise.

4.1.1. Majority Voting

There are three versions of majority voting, where the ensemble choose the class (i) on which all classifiers agree (*unanimous voting*); (ii) predicted by at least one more than half the number of classifiers (*simple majority*); or (iii) that receives the highest number of votes, whether or not the sum of those votes exceeds 50% (*plurality voting* or just *majority voting*). The ensemble decision for the plurality voting can be described as follows: choose class ω_J, if

$$\sum_{t=1}^{T} d_{t,J} = \max_{j=1}^{C} \sum_{t=1}^{T} d_{t,j}\,. \tag{17}$$

This is precisely the voting mechanism used by audience polling mentioned in the Introduction. It is not difficult to demonstrate that majority voting is an optimal combination rule under the minor assumptions of: (1) we have an odd number of classifiers for a two class problem; (2) the probability of each classifier choosing the correct class is p for any instance $\mathbf{x}$; and (3) the classifier outputs are independent. Then, plurality voting and simple majority voting are identical, and the ensemble makes the correct decision if at least $\lfloor T/2 \rfloor + 1$ classifiers choose the correct label, where the floor function $\lfloor \cdot \rfloor$ returns the largest integer less than or equal to its argument. The accuracy of the ensemble can be represented by the binomial distribution as the total probability of choosing $k \geq \lfloor T/2 \rfloor + 1$ successful one sout T of classifiers, where each classifier has the success rate of p. Hence, P_{ens}, the probability of ensemble success is

$$P_{ens} = \sum_{k=(T/2)+1}^{T} \binom{T}{k} p^k (1-p)^{T-k}\,. \tag{18}$$

This sum approaches 1 as $T \to \infty$, if $p < 0.5$; and it approaches 0 if $p < 0.5$. This result is known as the Condorcet Jury Theorem (1786), whose details can be found in [57], [58]. Herein lies the power of audience polling: if we can assume that the probability of each audience member giving the correct answer is better than $1/2$, then for a large enough audience (say, over 100), the probability of audience success approaches 1. Now, recall that we need to have $50\% + 1$ audience members to answer correctly for the majority voting to choose the correct answer for a two class problem; but the contestant has to choose one of four answers (a four class problem). It can be shown that, with more than two options, fewer classifiers could in fact be sufficient to obtain the correct class, since the incorrect votes will be distributed over a larger number of classes. Therefore, insisting that each classifier has a probability of success of $1/2$ or better, is in fact sufficient, but not necessary. It is actually a rather conservative requirement. An extensive and excellent analysis of the majority voting approach can be found in [21].

4.1.2. Weighted Majority Voting

If we have evidence that certain experts are more qualified than others, weighting the decisions of those qualified experts more heavily may further improve the overall performance than that can be obtained by the plurality voting. Again, let us denote the decision of hypothesis h_t on class ω_j as $d_{t,j}$, such that $d_{t,j}$ is 1, if h_t selects ω_j and 0, otherwise. Further assume that we have a way of estimating the future performance of each classifier, and we assign a weight w_t to classifier h_t in proportion to its estimated performance. According to this notation, the classifiers whose decisions are combined through weighted majority voting will chose class J, if

$$\sum_{t=1}^{T} w_t d_{t,J} = \max_{j=1}^{C} \sum_{t=1}^{T} w_t d_{t,j} \tag{19}$$

that is, if the total weighted vote received by ω_j is higher than the total vote received by any other class. For easier interpretation, we can normalize these weights so that they sum up to 1; however, normalization does not change the outcome of weighted majority voting.

A natural question is then "how do we assign the weights?" Clearly, had we known which classifiers would work better, we would give the highest weights to those classifiers, or perhaps, use only those classifiers. In the absence of this knowledge, a plausible strategy is to use the performance of a classifier on a separate validation dataset, or even its performance on the training dataset, as an estimate of that classifier's future performance. As indicated in Equations 13 and 15, AdaBoost follows the latter approach: AdaBoost assigns a voting weight of

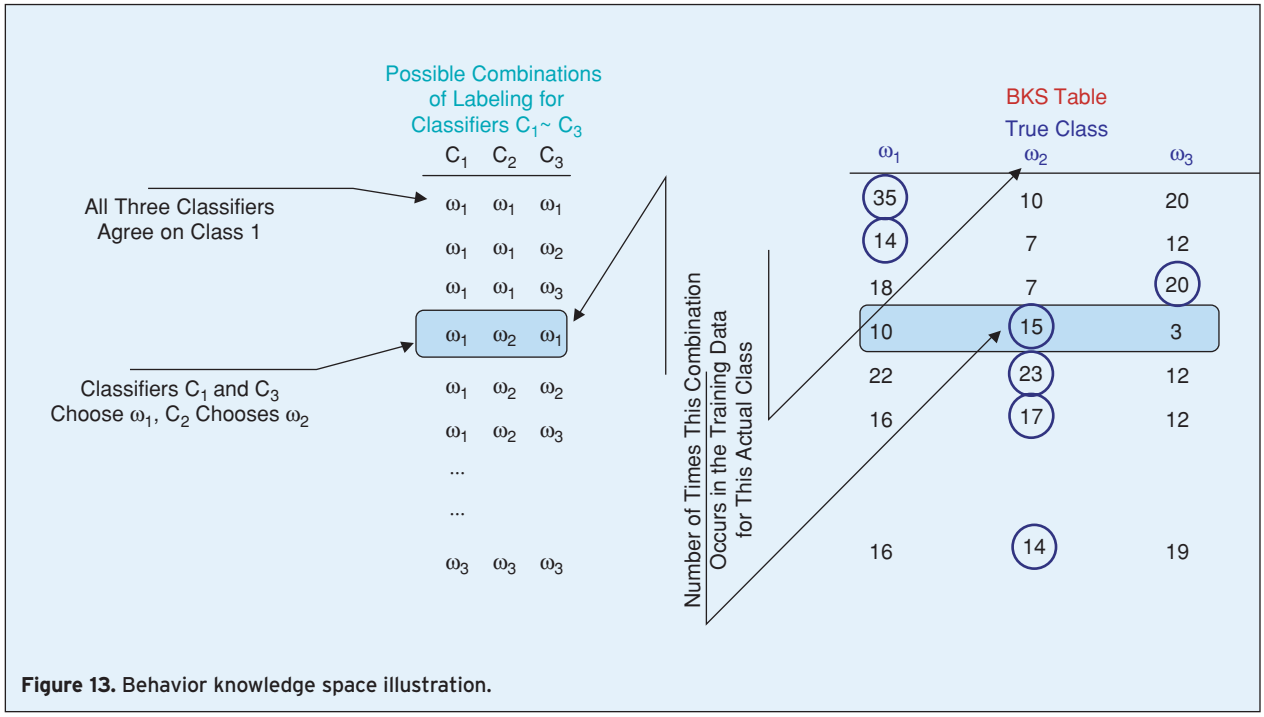


Figure 13. Behavior knowledge space illustration.

$\log(1/\beta_t)$ to h_t, where $\beta_t = \varepsilon_t/(1-\varepsilon_t)$, and ε_t is the weighted training error of hypothesis h_t. We have mentioned that the normalization of the error allows us to use an error measure β that is between 0 and 1 (as opposed to ε, which is between 0 and $1/2$); however, there is another strategic reason for this specific normalization. It can be shown that if we have T class-conditionally-independent classifiers with individual accuracies $p_1, \ldots, p_T$, the accuracy of the ensemble combined by weighted majority voting is maximized if the voting weights satisfy the proportionality

$$w_t \propto \log \frac{p_t}{1-p_t} \tag{20}$$

whose proof can be found in [21]. Representing the accuracies of individual classifiers as $p_t = 1-\varepsilon_t$, we see that the weight assignment used by AdaBoost is identical to the one in Equation 20. A detailed discussion on weighted majority voting can also be found in [59].

4.1.3. Behavior Knowledge Space (BKS)

Originally proposed by Huang and Suen, the behavioral knowledge space (BKS) uses a look up table, constructed based on the classification of the training data that keeps track of how often each labeling combination is produced by the classifiers [60], [61]. Then, the true class, for which a particular labeling combination is observed most often during training, is chosen every time that combination of class labels occurs during testing. The procedure is best described by an example, illustrated in Figure 13. Assume that we have three classifiers $C_1 \sim C_3$, for a three class problem. There are 27 possible combinations of labeling, listed from $\{\omega_1\,\omega_2\,\omega_3\}$ to [62], that can be chosen by the three classifiers. During training, we keep track of how often each combination occurs, and the corresponding correct class for each such occurrence. Sample numbers are provided for each combination and for each class, and the maxima are circled in Figure 13. For example, let us assume that the combination of $\{\omega_1\,\omega_1\,\omega_1\}$ occurs a total of 28 times, and in 10 of them the true class is ω_1, in 15 of them the true class is ω_2, and in 3 of them, the true class is ω_3. The winner of this combination is therefore ω_2, the most frequently observed true class for this combination of labels. Therefore, during testing, every time the combination of $\omega_1\,\omega_2\,\omega_1$ occurs, the ensemble chooses ω_2. Note that the choice of ω_2 is different than the class ω_1 that would have been chosen by the simple majority rule for this combination of labels. If trained with sufficiently dense training data, and if the table is appropriately normalized, the maximum vote obtained from the BKS table can estimate the class posterior probabilities with reasonable accuracy.

4.1.4. Borda Count

Borda count is different from rules listed previously in one important aspect: it does not throw away the support given to a non-winning class. Borda count is typically used if and when the classifiers can rank order the classes. This can be easily done if the classifiers provide continuous outputs, as the classes can then be rank ordered with respect to the support they receive from the

classifier. However, Borda count does not need the values of these continuous outputs, but just the rankings, hence it qualifies as a combination rule that apply to labels.

In standard Borda count, each voter (classifier) rank-orders the candidates (classes). If there are N candidates, the first-place candidate receives $N-1$ votes, the second-place candidate receives $N-2$, with the candidate in ith place receiving $N-i$ votes. The candidate ranked last receives 0 votes. The votes are added up across all classifiers, and the class with the most votes is chosen as the ensemble decision.

Originally devised in 1770 by Jean Charles de Borda, Borda count is used often in practice: selecting the most valuable player in U.S. baseball league, ranking universities in college sports, electing officers at certain university senate elections, and choosing the winning song in the annual European wide song contest Eurovision, all use some suitable variation of Borda count.

4.2. Combining Continuous Outputs

The continuous output provided by a classifier for a given class is interpreted as the degree of support given to that class, and it is usually accepted as an estimate of the posterior probability for that class. The posterior probability interpretation requires access to sufficiently dense training data, and that the outputs be appropriately normalized to add up to 1 over all classes. Usually, the *softmax* normalization is used for this purpose [63]. Representing the actual classifier output corresponding to the jth class as $g_j(\mathbf{x})$, and the normalized values as $g'_j(\mathbf{x})$, approximated posterior probabilities $P(\omega_j|\mathbf{x})$ can be obtained as

$$P(\omega_j|\mathbf{x}) \approx g'_j(\mathbf{x}) = \frac{e^{g_j(\mathbf{x})}}{\sum_{c=1}^{C} e^{g_c(\mathbf{x})}} \Rightarrow \sum_{j=1}^{C} g'_j(\mathbf{x}) = 1. \quad (21)$$

Kuncheva *et al.* define the so-called *decision profile* matrix in [16], which allows us to present all of the following combination rules from a unified perspective. The decision profile matrix $DP(\mathbf{x})$, for an instance $\mathbf{x}$, consists of elements $d_{t,j} \in [0, 1]$, which represent the support given by the tth classifier to class ω_j. The rows of $DP(\mathbf{x})$, therefore, represent the support given by individual classifiers to each of the classes, whereas the columns represent the support received by a particular class from all classifiers. The decision profile matrix is illustrated in Figure 14.

4.2.1. Algebraic combiners

Simple algebraic combiners are, in general, non-trainable combiners of continuous outputs. The total support for each class is obtained as a simple function of the supports received by individual classifiers. Following the same notation in [16], we represent the total support received by class ω_j, the jth column of the decision profile $DP(\mathbf{x})$, as

$$\mu_j(\mathbf{x}) = \Im[d_{1,j}(\mathbf{x}), \cdots, d_{T,j}(\mathbf{x})] \quad (22)$$

where $\Im(.)$ is the combination function, such as one of those listed below.

Mean Rule: The support for ω_j is obtained as the average of all classifiers' jth outputs,

$$\mu_j(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} d_{t,j}(\mathbf{x}) \quad (23)$$

that is, the function $\Im(\cdot)$ is the averaging function. The mean rule is equivalent to the sum rule (within a normalization factor of $1/T$), which also appears often in the literature. In either case, the ensemble decision is taken as the class ω_j for which the total support $\mu_j(\mathbf{x})$ is largest.

Weighted Average: This rule is a hybrid of the mean and the weighted majority voting, where the weights are applied not to class labels, but to the actual continuous outputs. This particular combination rule can qualify either as a trainable or non-trainable combination rule, depending on how the weights are obtained. If the weights are obtained during the ensemble generation as part of the regular training, as in AdaBoost, then it is a non-trainable combination rule. If a separate training is used to obtain the weights, such as in mixture of experts model, then it is a trainable combination rule. Usually, we have a weight for each classifier, or sometimes for each

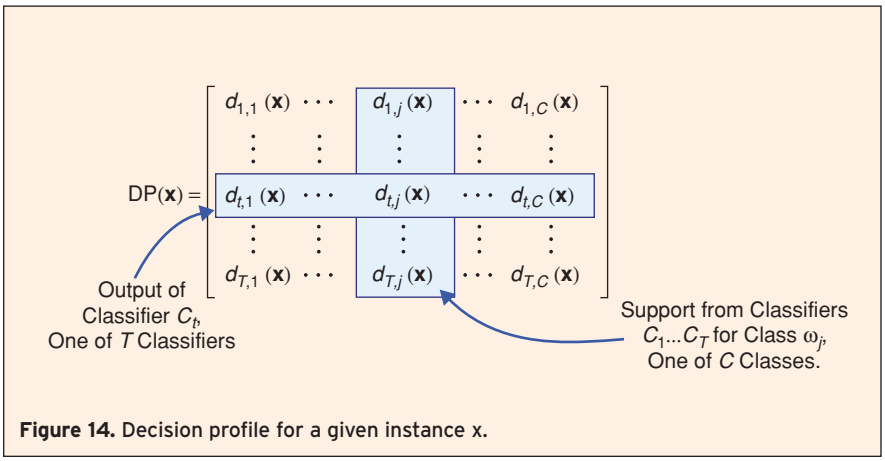
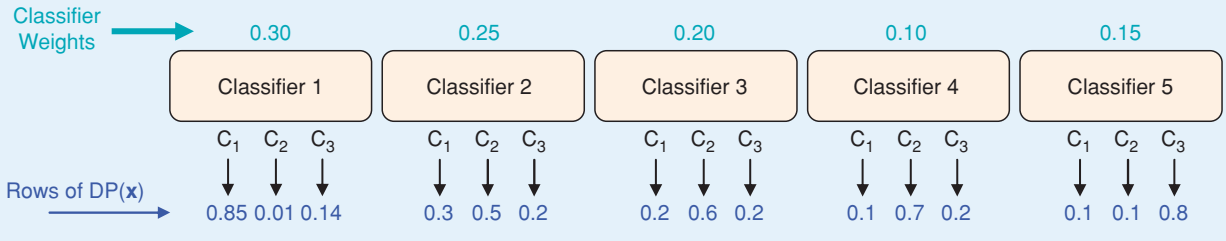


Figure 14. Decision profile for a given instance x.



Product Rule:	**C_1: 0.0179,**	C_2: 0.00021,	C_3: 0.0009
Sum Rule:	C_1: 1.55,	**C_2: 1.91,**	C_3: 1.54
Mean Rule:	C_1: 0.310,	**C_2: 0.382,**	C_3: 0.308
Max Rule:	**C_1: 0.85,**	C_2: 0.7,	C_3: 0.8
Median Rule:	C_1: 0.2,	**C_2: 0.5,**	C_3: 0.2
Minimum Rule:	C_1: 0.1,	C_2: 0.01,	**C_3: 0.14**
Weighted Average	**C_1: 0.395,**	C_2: 0.333,	C_3: 0.272
Majority Voting:	C_1: 1,	**C_2: 3,**	C_3: 1 Vote
Weighted Majority Voting:	C_1: 0.3,	**C_2: 0.55,**	C_3: 0.15 Votes
Borda Count:	C_1: 5	**C_2: 6,**	C_3: 4 Votes

Figure 15. Example on various combination rules.

class and for each classifier. In the former case, we have T weights, $w_1, \ldots, w_T$, usually obtained as estimated accuracies from training performances, and the total support for class ω_j is

$$\mu_j(\mathbf{x}) = \sum_{t=1}^{T} w_t d_{t,j}(\mathbf{x}). \tag{24}$$

In the latter case, we have a total of $T \times C$ weights that are class and classifier specific, hence a class-conscious combination [16]. Total support for class ω_j is given as

$$\mu_j(\mathbf{x}) = \sum_{t=1}^{T} w_{t,j} d_{t,j}(\mathbf{x}) \tag{25}$$

where $w_{t,j}$ is the weight of the t th classifier for classifying class ω_j instances.

Trimmed Mean: What if some of the classifiers give unwarranted and unusually low, or unusually high support to a particular class? This would adversely affect the mean combiner. To avoid this problem, the most optimistic and pessimistic classifiers are removed from the ensemble before calculating the mean, a procedure known as the trimmed mean. For a $P\%$ trimmed mean, we simply remove $P\%$ of the support from each end, and the mean of the remaining supports are calculated, avoiding extreme values of support.

Minimum/Maximum/Median Rule: As the names imply, these functions simply take the minimum, maximum or the median among the classifiers' individual outputs.

$$\begin{aligned}
\mu_j(\mathbf{x}) &= \max_{t=1\cdots T}\{d_{t,j}(\mathbf{x})\} \\
\mu_j(\mathbf{x}) &= \min_{t=1\cdots T}\{d_{t,j}(\mathbf{x})\} \\
\mu_j(\mathbf{x}) &= \operatorname{median}_{t=1\cdots T}\{d_{t,j}(\mathbf{x})\}
\end{aligned} \tag{26}$$

In any of these cases, the ensemble decision is again chosen as the class for which total support is largest. The minimum rule is the most conservative combination rule, as it chooses the class for which the *minimum support* among the classifiers is largest. Also worth mentioning is that the trimmed mean at limit 50% is equivalent to the median rule.

Product Rule: In product rule, supports provided by the classifiers are multiplied. This rule is very sensitive to the most pessimistic classifiers: a low support (close to 0) for a class from any of the classifiers can effectively remove any chance of that class being selected. However, if individual posterior probabilities are estimated correctly at the classifier outputs, then this rule provides the best estimate of the overall posterior probability of the class selected by the ensemble.

$$\mu_j(\mathbf{x}) = \frac{1}{T}\prod_{t=1}^{T} d_{t,j}(\mathbf{x}). \tag{27}$$

Generalized Mean: Many of the above rules are in fact special cases of the generalized mean,

$$\mu_j(\mathbf{x}, \alpha) = \left(\frac{1}{T}\sum_{t=1}^{T} d_{t,j}(\mathbf{x})^{\alpha}\right)^{1/\alpha} \tag{28}$$

where specific choices of α results in different combination rules discussed above. For example, for $\alpha \to -\infty$, we obtain the minimum rule, $\mu_j(\mathbf{x}, \alpha) = \min_t \{d_{t,j}](\mathbf{x})\}$; for $\alpha \to 0$, we obtain

$$\mu_j(\mathbf{x}, \alpha) = \left(\prod_{t=1}^{T} d_{t,j}(\mathbf{x})\right)^{1/T} \tag{29}$$

which is the geometric mean, a modified version of the product rule. For $\alpha \to 1$, we obtain the mean rule, and $\alpha \to \infty$ gives us the maximum rule.

As an example, consider the case in Figure 15, where we have a five classifier ensemble for a three class problem. Assume that we have obtained the outputs and weights shown in Figure 15 for a given input $\mathbf{x}$ from the ensemble. The corresponding decision profile for $\mathbf{x}$ is then

$$DP(\mathbf{x}) = \begin{bmatrix} 0.85 & 0.01 & 0.14 \\ 0.3 & 0.5 & 0.2 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.7 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}. \tag{30}$$

We make several observations from Figure 15, where the winning class for each combination rule is shown in bold. (i) Different classes may win for different combination rules. In this example, class 2 wins most of the combination rules, class 1 wins three and class 3 wins only one rule (the minimum rule). Had Classifier 5 output been [0.05 0.05 0.9], instead of [0.1 0.1 0.8], however, class 3 would have also won the maximum rule; (ii) the product rule and minimum rule severely punishes class 2, as it received very little support from classifier C_1; (iii) sum rule and mean rule provide the same outcome, as expected; and (iv) the weighted average chooses class 1 primarily because of the high support given to class 1 by the highly weighted Classifier 1. The three voting-based rules for combining label outputs are also provided, all of which have voted in favor of class 2 (for Borda Count, the ties in rankings were randomly broken in favor of class 1, and classifier weights given on top of classifiers were used for weighted majority voting).

4.2.2. Decision Templates

Kuncheva takes the idea of decision profiles one step further by defining *decision templates* as the average decision profile observed for each class throughout training [16]. Given a test instance $\mathbf{x}$, its decision profile is compared to the decision templates of each class, and the class whose decision template is closest, in some similarity measure, is chosen as the ensemble decision. More specifically, the decision template for ω_j, illustrated in Figure 16, is calculated as

$$DT_j = \frac{1}{N_j}\sum_{P\mathbf{X}_j \in \omega_j} DP(\mathbf{X}_j) \tag{31}$$

which is the average decision profile obtained from $\mathbf{X}_j$, the set (with cardinality N_j) of training in stances that belong to true class ω_j. Given an unlabeled test instance $\mathbf{x}$, we first construct its $DP(\mathbf{x})$ from the ensemble outputs (as shown in Figure 15), and calculate the similarity S between $DP(\mathbf{x})$ and the decision template DT_j for each class ω_j as the degree of support given to class ω_j.

$$\mu_j(\mathbf{x}) = S(DP(\mathbf{x}), DT_j), \quad j = 1, \cdots, C. \tag{32}$$

The similarity measure S is usually a squared Euclidean distance, obtained as

$$\mu_j(\mathbf{x}) = 1 - \frac{1}{T \times C}\sum_{t=1}^{T}\sum_{k=1}^{C}[DT_j(t,k) - d_{t,k}(\mathbf{x})]^2 \tag{33}$$

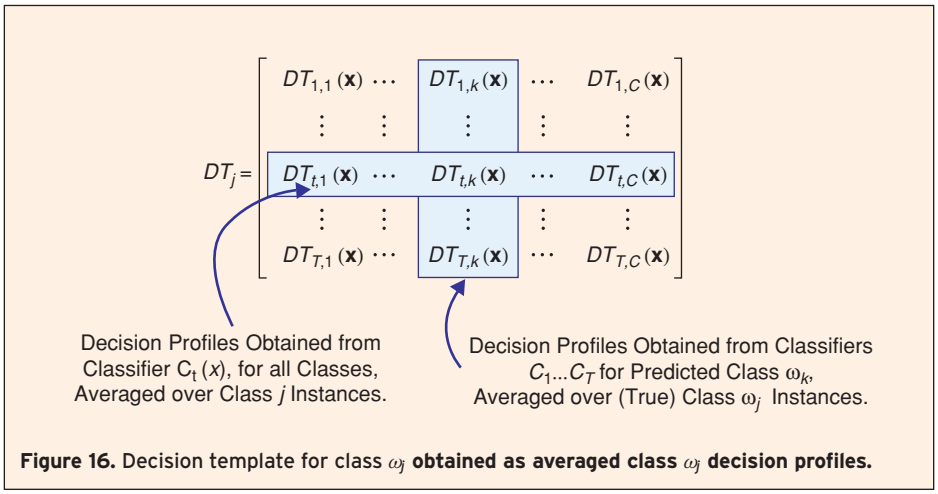


Figure 16. Decision template for class ω_j **obtained as averaged class** ω_j **decision profiles.**

where $DT_j(t,k)$ is the support given by the tth classifier to class ω_k by the decision template DT_j. In other words, $DT_j(t,k)$ is the support given by the tth classifier to class ω_k, averaged over class ω_j instances. This support should ideally be high when $k=j$, and low otherwise. The second term $d_{t,k}(\mathbf{x})$ is the support given by the tth classifier to class ω_k for the given instance $\mathbf{x}$. As usual, the class with the highest total support is finally chosen as the ensemble decision.

4.2.3. Dempster-Shafer Based Combination

This combination rule is borrowed from *data fusion*, a field of data analysis primarily involved in combining elements of evidence provided by different sources of data. Many data fusion techniques are based on the Dempster-Shafer (DS) theory of evidence, which uses belief functions (instead of probability) to quantify the evidence available from each data source, which are then combined using Dempster's rule of combination [64], [65]. DS theory has traditionally been used in military applications, such as sensor fusion for target tracking, or friend or foe identification. It can easily be applied, however, to any decision making problem, by interpreting the output of a classifier as a measure of evidence provided by the source that generated the training data [10], [34]. In such a setting, DS theory is not used for data fusion, that is, to combine data from different sources, but rather, to combine the evidence provided by ensemble classifiers trained on data coming from the same source. We describe the DS theory as an ensemble combination rule here, and discuss its feasibility of ensemble systems for combining data from different sources later under Current and Emerging Areas.

The decision template formulation becomes useful in describing DS theory as an ensemble combination rule: let DT_j^t denote the tth row of the decision template DT_j, and $C_t(\mathbf{x})$ denote the output of the tth classifier, that is, the tth row of the decision profile $DP(\mathbf{x})$: $C_t(\mathbf{x}) = [d_{t,1}(\mathbf{x}), \ldots, d_{t,C}(\mathbf{x})]$. Instead of similarity, we now calculate *proximity* $\Phi_{j,t}(\mathbf{x})$ of the tth classifier's class j decision template DT_j^t to this classifier's decision on instance $\mathbf{x}$, $C_t(\mathbf{x})$ [10], [21], [35]:

$$\Phi_{j,t}(\mathbf{x}) = \frac{\left(1 + \left\| DT_j^t - C_t(\mathbf{x}) \right\|^2\right)^{-1}}{\sum_{k=1}^{C} \left(1 + \left\| DT_k^t - C_t(\mathbf{x}) \right\|^2\right)^{-1}} \tag{34}$$

where the *differences* (calculated as distances in Euclidean norm) in both numerator and denominator are converted to *similarities*—representing proximities—using the reciprocal operation. The denominator is really a normalization term, representing the total proximity of tth classifier decision to the decision templates of all classes. Based on these proximities, we compute our *belief*, or evidence, that the tth classifier C_t is correctly identifying instance $\mathbf{x}$ into class ω_j,

$$b_j(C_t(\mathbf{x})) = \frac{\Phi_{j,t}(\mathbf{x}) \prod_{k \neq j} \left(1 - \Phi_{k,t}(\mathbf{x})\right)}{1 - \Phi_{j,t}(\mathbf{x}) \left[1 - \prod_{k \neq j} \left(1 - \Phi_{k,t}(\mathbf{x})\right)\right]} \tag{35}$$

Once the *belief* values are obtained for each source (classifier), they can be combined by Dempster's rule of combination, which simply states that the evidences (belief values) from each source should be multiplied to obtain the final support for each class:

$$\mu_j(\mathbf{x}) = K \prod_{t=1}^{T} b_j(C_t(\mathbf{x})) \tag{36}$$

where K is a normalization constant ensuring that the total support for ω_j from all classifiers is 1.

4.3. But, Which One Is Better?

When there are many competing approaches to a problem, an effort to determine a winning one is inevitable. Hence the question "which ensemble generation or combination rule is the best?" The *no-free-lunch* theorem has unarguably proven that there is in fact no such best classifier for all classification problems [47], and that the best algorithm depends on the structure of the available data and prior knowledge. Yet, many studies have compared various ensemble generation and combination rules under various scenarios. For example, Dietterich [66], [67], Breiman [68], Bauer *et al.* [69], Drucker [17] and Quinlan [70] all separately compared bagging, boosting and other ensemble based approaches. The typical consensus is that boosting usually achieves better generalization performances, but it is also more sensitive to noise and outliers.

Does the no-free-lunch theorem hold for combination rules as well? Many studies have been conducted to answer this question, including [18], [27], [28], [31], [36], [37], [42], [71], [72], among others. Their verdict? In general, the no-free-lunch theorem holds. The best combination method, just as for the best ensemble method, depends much on the particular problem. However, there is a growing consensus on using the mean rule due to its simplicity and consistent performance over a broad spectrum of applications. If the accuracies of the classifiers can be reliably estimated, then the weighted average and weighted majority approaches may be considered. If the classifier outputs correctly estimate the posterior probabilities, then the product rule should be considered.

5. Current & Emerging Areas

Despite two decades of intense research, and the widely held belief that our current understanding of ensemble based systems has matured [73], the field seems to be enjoying a growing attention. This may, in part, be due to emerging areas of applications that benefit from ensemble systems. The primary thrust of using ensemble systems has been to reduce the risk of choosing a single classifier with a poor performance, or to improve upon the performance of a single classifier by using an intelligently combined ensemble of classifiers. Many additional areas and applications have recently emerged, however, for which the ensemble systems are inherently appropriate. In this section, we discuss some of the more prominent examples of such emerging areas.

5.1. Incremental Learning

In certain applications, it is not uncommon for the entire dataset to gradually become available in small batches over a period of time. Furthermore, datasets acquired in subsequent batches may introduce instances of new classes that were not present in previous datasets. In such settings, it is necessary to learn the novel information content in the new data, without forgetting the previously acquired knowledge, and without requiring access to previously seen data. The ability of a classifier to learn under these circumstances is usually referred to as *incremental learning*.

A practical approach for learning from new data involves discarding the existing classifier, and retraining a new one using all data that have been accumulated thus far. This approach, however, results in loss of all previously acquired information, a phenomenon known as *catastrophic forgetting* (or *interfering*) [74]. Not conforming to the above stated definition of incremental learning aside, this approach is undesirable, if retraining is computationally or financially costly; but more importantly, it is unfeasible, if the original dataset is lost, corrupted, discarded, or otherwise unavailable. Such scenarios are not uncommon in databases of restricted or confidential access, such as in medical and military applications.

Ensemble systems have been successfully used to address this problem. The underlying idea is to generate additional ensembles of classifiers with each subsequent database that becomes available, and combine their outputs using one of the combination methods discussed above. The algorithm Learn^{++} and its recent variations have been shown to achieve incremental learning on a broad spectrum of applications, even when new data introduce instances of previously unseen classes [55], [75], [76]. Learn^{++} is inspired in part by AdaBoost; however it differs from AdaBoost in one important aspect: recall that AdaBoost updates its weight distribution based on the performance of hypothesis h_t generated in the previous iteration [26]. In contrast, Learn^{++} introduces the notion of composite hypothesis—the combined ensemble generated thus far at any given iteration—and updates its distribution based on the performance of the current *ensemble* through the use of this composite hypothesis. Learn^{++} then focuses on instances that have not been properly learned by the entire *ensemble*. During incremental learning, previously unseen instances, particularly those coming from a new class, are bound to be misclassified by the ensemble, forcing the algorithm to focus on learning such instances introduced by the new data. Furthermore, Learn^{++} creates an ensemble of ensemble classifiers, one ensemble for each database. The ensembles are then combined through a modified weighted majority voting algorithm. More recently, it was shown that the algorithm also works well, even when the data distribution is unbalanced, where the number of instances in each class or database vary significantly [77].

5.2. Data Fusion

The goal of data fusion is to extract complementary pieces of information from different data sources, allowing a more informed decision about the phenomenon generating the underlying data distributions. While ensemble systems are often used for fusing classifier outputs, such applications have traditionally used classifiers that are trained with different sampling of the data that comes from the same distribution. Therefore, all classifiers are trained on the same feature set. In data fusion applications, however, data obtained from different sources may use different feature sets that may be quite heterogeneous in their composition. An example would be combining medical test results including an MRI scan (an image), an electroencephalogram (a time series), and several blood tests (scalar numbers) for the diagnosis of a neurological disorder.

Even with heterogeneous feature sets, data fusion is a natural fit for ensemble systems, since different classifiers can be generated using data obtained from different sources, and subsequently combined to achieve the desired data fusion. Several classifier fusion approaches have been proposed for this purpose, including combining classifiers using Dempster-Shafer based combination [78]–[80], ARTMAP [81], Learn^{++} [82], genetic algorithms [83] and other combinations of boosting/voting methods [84]–[86].

5.3. Feature Selection

One way to improve diversity in the ensemble is to train the classifiers with data that consist of different feature subsets. Several approaches based on this idea have

Table 1.
Exhaustive ECOC for a five-class problem.

	Classifiers														
Class	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}
ω_1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ω_2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
ω_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
ω_4	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
ω_5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

known as the *random subspace method*, a term coined by Ho [44], who used it on constructing decision tree ensembles. Subset selection need not be random: Oza and Tumer propose *input decimation*, where the features are selected based on their correlation with the class labels [39]. In general, feature subsets should promote diversity, that is, identical subsets should not be repeated. While most approaches allow overlapping subsets of features, Gunter and Bunke [87] use mutually exclusive sets.

An interesting application of using feature subsets has been proposed by Krause and Polikar, who used an ensemble of classifiers trained with an iteratively updated feature distribution to address the missing feature problem [88]. The idea is to generate a large enough ensemble of classifiers using random subsets of features. Then, if certain features are missing from a given data instance, that instance can still be classified by a pool of classifiers that did not use the missing features during their training. Another interesting application of using an ensemble of classifiers with different feature subsets was proposed by Long and Vega, who have used the ensemble performance in determining the most informative feature subsets in gene expression data [89].

All of the above mentioned approaches implicitly assume that there is redundancy in the overall feature set, and that the features are not consecutive elements of a time-series data.

5.4. Error Correcting Output Codes

Error correcting output codes (ECOC) have originally been used in information theory for correcting bit reversals caused by noisy communication channels. More recently, they have also been used in converting binary classifiers, such as support vector machines, to multi-class classifiers by decomposing a multi-class problem into several two-class problems [90]. Inspired by these works, Dietterich and Bakiri introduced ECOC to be used within the ensemble setting [91]. The idea is to use a different class encoding for each member of the ensemble. where C is the number of classes and T is the number of classifiers, combined by the minimum Hamming distance rule. Table 1 shows a particular code matrix for a 5-class problem that uses 15 encodings. This encoding, suggested in [91], is known as the *exhaustive coding* because it includes all possible non-trivial and non-repeating codes. In this formulation, the individual classifiers are trained on several meta two-class problems, where individual meta-classes include some combination of the original classes. For example, C_5 recognizes two meta-classes: original classes ω_1 and ω_3 constitute one class, and the others constitute the second class. During testing, each classifier outputs a "0" or "1" creating a $2^{C-1}-1$ long output code vector. This vector is compared to each code word in the code matrix, and the class whose code word has the shortest Hamming distance to the output vector is chosen as the ensemble decision. More formally, the support for class ω_j is given as

$$\mu_j(\mathbf{x}) = -\sum_{t=1}^{T} |o_t - M_{j,t}| \tag{37}$$

where $o_t \in \{0, 1\}$ is the output of the tth binary classifier, and M is the code matrix. The negative sign converts the distance metric into a support value, whose largest value can be zero in case of a perfect match. For example, the output [0 1 1 1 0 1 0 1 0 1 0 1 0 1 0] is closest to ω_5 code word with a Hamming distance of 1 (support of -1), and hence ω_5 would be chosen for this output. Note that this output does not match any of the code words exactly, and therein lies the error correcting ability of the ECOC. In fact, the larger the minimum Hamming distance between code words, the more resistant the ECOC ensemble becomes to misclassifications of individual classifiers. More efficient and error-resistant coding (choosing appropriate codes) as well as decoding (combination methods other than Hamming distance) are topics of current research [92], [93].

5.5. Confidence Estimation

Using an ensemble based system also allows us to quantitatively assess the confidence of the decision even on an instance-by-instance basis. Loosely speaking, if a vast majority of classifiers in the ensemble agree on the decision of a given instance, we can interpret this outcome as the ensemble having a high confidence in its decision. Conversely, if the final decision is made based on a small difference in classifier opinions, then the ensemble would have less confidence in its decision. Muhlbaier *et al.* showed that continuous valued outputs of ensemble classifiers can also be used as an estimate of the posterior probability, which in turn can be interpreted as the confidence in the decision [29]. Specifically, they showed that the posterior probability of ω_j can be estimated through a softmax type normalization of classifier outputs:

$$P(\omega_j|\mathbf{x}) \approx X_j(\mathbf{x}) = \frac{e^{F_j(\mathbf{x})}}{\sum_{c=1}^{C} e^{F_c(\mathbf{x})}} \tag{38}$$

where $X_j(\mathbf{x})$ is the confidence of the ensemble in assigning instance $\mathbf{x}$ into class ω_j and

$$F_j(\boldsymbol{x}) = \sum_{t=1}^{N} \begin{pmatrix} \log(1/\beta_t) & h_t(\boldsymbol{x}) = \omega_j \\ 0 & \text{otherwise} \end{pmatrix} \tag{39}$$

is the total weighted sum of the weights for ω_j. More interestingly, on several benchmark datasets of known distributions, they showed that the posterior probability estimate of the ensemble approaches the true posterior probability as the number of classifiers in the ensemble increases.

5.6. Other Areas

There are other areas in which ensemble systems have been used and/or proposed, and the list is continuously growing at what appears to be a healthy rate. Some of the more promising ones include using ensemble systems in non-stationary environments [19], in clustering applications [94]–[96], and countless work on theoretical analyses of such systems, such as on incremental learning, improving diversity, bias-variance analysis, etc. Specific practical applications of ensemble systems—whether used for biomedical, financial, remote sensing, genomic, oceanographic, chemical or other types of data analysis problems are also rapidly growing.

6. Conclusions

Over the last decade, the ensemble based systems have enjoyed a growing attention and popularity due to their many desired properties, and the broad spectrum of applications that can benefit from them. In this paper, we discussed the fundamental aspects of these ensemble systems, including the need to ensure—and ways to measure—diversity in the ensemble; ensemble generation techniques such as bagging, AdaBoost, mixture of experts; and classifier combination strategies, such as algebraic combiners, voting methods, and decision templates. We have also reviewed some of the more popular areas where ensemble systems become naturally useful, such as incremental learning, data fusion, feature selection and error correcting output codes.

Whereas there is no single ensemble generation algorithm or combination rule that is universally better than others, all of the approaches discussed above have been shown to be effective on a wide range of real world and benchmark datasets, provided that the classifiers can be made as diverse as possible. In the absence of any other prior information, the best ones are usually (as the William of Ockham had said about 8 centuries ago) the simplest and least complicated ones that can learn the underlying data distribution.

As for the quiz show with the large payout, where you had to choose from the three lifelines, let's focus our attention once again to Equation 18:

- if the individual audience members vote independently (a reasonable assumption); and
- there is a reasonably large audience (another reasonable assumption); and
- each has a probability of 0.5 or higher for getting the correct answer (sufficient, but not necessary for more than two choices),

then the probability of correct answer of the majority voting of the audience (the ensemble) approaches 1. So, unless you know for a fact that your friend (for the "call a friend" option) has a very high probability of knowing the correct answer (prior information that warrants a change in your course of action), you are better off with the audience. Just in case you find yourself in that fortunate situation . . . .

Software

The following software tools include Matlab based functions that can be used for ensemble learning. Many other tools can also be found on the Internet.

- C. Merkwinh and J. Wichard, "ENTOOL—A Matlab Toolbox for Ensemble Modeling," 2002. Available online at: http://chopin.zet.agh. edu.pl/∼wichtel
- R. P.W. Duin, D. Ridder, P. Juszczak, P. Paclik, E. Pekalska and D. Tax, "PRTools," 2005. Available online at: http://www.prtools.org
- D. Stork and E. Yom-Tov, "Computer Manual in Matlab to Accompany Pattern Classification, 2nd Edition, New York: Wiley, 2004.

Acknowledgment

This material is in part based upon work supported by the National Science Foundation under Grant No: ECS 0239090.

References

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

[2] L.K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, 1990.

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

[4] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton, "Adaptive mixtures of local experts," *Neural Computation*, vol. 3, pp. 79–87, 1991.

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

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

[7] J.A. Benediktsson and P.H. Swain, "Consensus theoretic classification methods," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 22, no. 4, pp. 688–704, 1992.

[8] L. Xu, A. Krzyzak, and C.Y. Suen, "Methods of combining multiple classifiers and their applications to handwriting recognition," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, no. 3, pp. 418–435, 1992.

[9] T.K. Ho, J.J. Hull, and S.N. Srihari, "Decision combination in multiple classifier systems," *IEEE Trans. on Pattern Analy. Machine Intel.*, vol. 16, no. 1, pp. 66–75, 1994.

[10] G. Rogova, "Combining the results of several neural network classifiers," *Neural Networks*, vol. 7, no. 5, pp. 777–781, 1994.

[11] L. Lam and C.Y. Suen, "Optimal combinations of pattern classifiers," *Pattern Recognition Letters*, vol. 16, no. 9, pp. 945–954, 1995.

[12] K. Woods, W.P.J. Kegelmeyer, and K. Bowyer, "Combination of multiple classifiers using local accuracy estimates," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 4, pp. 405–410, 1997.

[13] L.I. Kuncheva, "Change-glasses' approach in pattern recognition," *Pattern Recognition Letters*, vol. 14, no. 8, pp. 619–623, 1993.

[14] I. Bloch, "Information combination operators for data fusion: A comparative review with classification," *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans.*, vol. 26, no. 1, pp. 52–67, 1996.

[15] S.B. Cho and J.H. Kim, "Combining multiple neural networks by fuzzy integral for robust classification," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 25, no. 2, pp. 380–384, 1995.

[16] L.I. Kuncheva, J.C. Bezdek, and R. Duin, "Decision templates for multiple classifier fusion: An experimental comparison," *Pattern Recognition*, vol. 34, no. 2, pp. 299–314, 2001.

[17] H. Drucker, C. Cortes, L.D. Jackel, Y. LeCun, and V. Vapnik, "Boosting and other ensemble methods," *Neural Computation*, vol. 6, no. 6, pp. 1289–1301, 1994.

[18] R. Battiti and A.M. Colla, "Democracy in neural nets: Voting schemes for classification," *Neural Networks*, vol. 7, no. 4, pp. 691–707, 1994.

[19] L.I. Kuncheva, "Classifier ensembles for changing environments," 5th Int. Workshop on Multiple Classifier Systems, *Lecture Notes in Computer Science,* F. Roli, J. Kittler, and T. Windeatt, Eds., vol. 3077, pp. 1–15, 2004.

[20] F. Smieja, "Pandemonium system of reflective agents," *IEEE Transactions on Neural Networks*, vol. 7, no. 1, pp. 97–106, 1996.

[21] L.I. Kuncheva, *Combining Pattern Classifiers, Methods and Algorithms*. New York, NY: Wiley Interscience, 2005.

[22] E. Alpaydin and M.I. Jordan, "Local linear perceptrons for classification," *IEEE Transactions on Neural Networks*, vol. 7, no. 3, pp. 788–792, 1996.

[23] F. Roli, G. Giacinto, and G. Vernazza, "Methods for designing multiple classifier systems," 2nd Int. Workshop on Multiple Classifier Systems, in *Lecture Notes in Computer Science,* J. Kittler and F. Roli, Eds., vol. 2096, pp. 78–87, 2001.

[24] G. Giacinto and F. Roli, "Approach to the automatic design of multiple classifier systems," *Pattern Recognition Letters*, vol. 22, no. 1, pp. 25–33, 2001.

[25] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[26] Y. Freund and R.E. Schapire, "Decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[27] F.M. Alkoot and J. Kittler, "Experimental evaluation of expert fusion strategies," *Pattern Recognition Letters*, vol. 20, no. 11–13, pp. 1361–1369, Nov. 1999.

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

[29] M. Muhlbaier, A. Topalis, and R. Polikar, "Ensemble confidence estimates posterior probability," 6th Int. Workshop on Multiple Classifier Sys., in *Lecture Notes in Comp. Science,* N. Oza, R. Polikar, J. Kittler, and F. Roli, Eds., vol. 3541, pp. 326–335, 2005.

[30] J. Grim, J. Kittler, P. Pudil, and P. Somol, "Information analysis of multiple classifier fusion," *2nd Int. Workshop on Multiple Classifier Systems*, in *Lecture Notes in Computer Science,* J. Kittler and F. Roli, Eds., vol. 2096, pp. 168–177, 2001.

[31] L.I. Kuncheva, "A theoretical study on six classifier fusion strategies," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 281–286, 2002.

[32] S.B. Cho and J.H. Kim, "Multiple network fusion using fuzzy logic," *IEEE Transactions on Neural Networks*, vol. 6, no. 2, pp. 497–501, 1995.

[33] M. Grabisch and J.M. Nicolas, "Classification by fuzzy integral: performance and tests," *Fuzzy Sets and Systems*, vol. 65, no. 2–3, pp. 255–271, 1994.

[34] Y. Lu, "Knowledge integration in a multiple classifier system," *Applied Intelligence*, vol. 6, no. 2, pp. 75–86, 1996.

[35] L.I. Kuncheva, "Using measures of similarity and inclusion for multiple classifier fusion by decision templates," *Fuzzy Sets and Systems*, vol. 122, no. 3, pp. 401–407, 2001.

[36] L.I. Kuncheva, "Switching between selection and fusion in combining classifiers: An experiment," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 32, no. 2, pp. 146–156, 2002.

[37] D. Tax, M. van Breukelen, R.P.W. Duin, and J. Kittler, "Combining multiple classifiers by averaging or by multiplying?," *Pattern Rec.*, vol. 33, no. 9, pp. 1475–1485, 2000.

[38] L.I. Kuncheva and C.J. Whitaker, "Feature Subsets for Classifier Combination: An Enumerative Experiment," 2nd Int. Workshop on Multiple Classifier Systems, in *Lecture Notes in Comp. Science,* J. Kittler and F. Roli, Eds., vol. 2096, pp. 228–237, 2001.

[39] N.C. Oza and K. Tumer, "Input decimation ensembles: Decorrelation through dimensionality reduction," 2nd Int. Workshop on Multiple Classifier Systems, in *Lecture Notes in Computer Science,* J. Kittler and F. Roli, Eds., vol. 2096, pp. 238–247, 2001.

[40] N.S.V. Rao, "On fusers that perform better than best sensor," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 8, pp. 904–909, 2001.

[41] K. Tumer and J. Ghosh, "Analysis of decision boundaries in linearly combined neural classifiers," *Pattern Recognition*, vol. 29, no. 2, pp. 341–348, 1996.

[42] G. Fumera and F. Roli, "A Theoretical and Experimental Analysis of Linear Combiners for Multiple Classifier Systems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 6, pp. 942–956, 2005.

[43] Various Authors, "Proceedings of International Workshop on Multiple Classifier Systems (2000–2005)," F. Roli, J. Kittler, T. Windeatt, N. C. Oza, and R. Polikar, Eds. Berlin, Germany: Springer, 2005.

[44] T.K. Ho, "Random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[45] L. Kuncheva and C. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine Learning*, vol. 51, no. 2, pp. 181–207, 2003.

[46] G. Brown, "Diversity in neural network ensembles," Ph.D. dissertation, University of Birmingham, 2004.

[47] D.H. Wolpert and W.G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[48] B. Rosen , "Ensemble learning using decorrelated neural networks," *Connection Science*, vol. 8, no. 3, pp. 373–384, 1996.

[49] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[50] L. Breiman, "Pasting small votes for classifcation in large databases and on-line," *Machine Learning*, vol. 36, pp. 85–103, 1999.

[51] N.V. Chawla, L.O. Hall, K.W. Bowyer, J. Moore, and W.P. Kegelmeyer, "Distributed pasting of small votes," 3rd Int. Workshop on Multiple Classifier Systems, in *Lecture Notes in Computer Science,* F. Roli and J. Kittler, Eds., vol. 2364, p. 52–61, Springer, 2002.

[52] R.E. Schapire, Y. Freund, P. Bartlett, and W.S. Lee, "Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods," *Annals of Statistics*, vol. 26, no. 5, pp. 1651–1686, 1998.

[53] N.C. Oza, "Boosting with Averaged Weight Vectors," 4th Int. Workshop on Multiple Classifier Systems, in *Lecture Notes in Computer Science,* T. Windeatt and F. Roli, Eds., vol. 2709, pp. 15–24, Springer, 2003.

[54] N.C. Oza, "AveBoost2: Boosting for Noisy Data," 5th Int. Workshop on Multiple Classifier Systems, in *Lecture Notes in Computer Science,* vol. 3077, pp. 31–40, F. Roli, J. Kittler, and T. Windeatt, Eds. Springer, 2004.

[55] R.Polikar, L. Udpa, S.S. Udpa, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 31, no. 4, pp. 497–508, 2001.

[56] M.I. Jordan and L. Xu, "Convergence results for the EM approach to mixtures of experts architectures," *Neural Networks*, vol. 8, no. 9, pp. 1409–1431, 1995.

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

[58] L. Shapley and B. Grofman, "Optimizing group judgmental accuracy in the presence of interdependencies," *Public Choice (Historical Archive)*, vol. 43, no. 3, pp. 329–343, 1984.

[59] N. Littlestone and M. Warmuth, "Weighted majority algorithm," *Information and Computation*, vol. 108, pp. 212–261, 1994.

[60] Y.S. Huang and C.Y. Suen, "Behavior-knowledge space method for combination of multiple classifiers," *Proc. of IEEE Computer Vision and Pattern Recog.*, pp. 347–352, 1993.

[61] Y.S. Huang and C.Y. Suen, "A method of combining multiple experts for the recognition of unconstrained handwritten numerals," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 17, no. 1, pp. 90–94, 1995.

[62] H. Ichihashi, T. Shirai, K. Nagasaka, and T. Miyoshi, "Neuro-fuzzy ID3: a method of inducing fuzzy decision trees with linear programming for maximizing entropy and an algebraic method for incremental learning," *Fuzzy Sets and Systems*, vol. 81, no. 1, pp. 157–167, 1996.

[63] R.O. Duda, P.E. Hart, and D. Stork, "Algorithm Independent Techniques," in *Pattern Classification,* 2 ed New York: Wiley, pp. 453–516, 2001.

[64] A.P. Dempster, "Upper and lower probabilities induced by multivalued mappings," *Annals of Mathematical Statistics*, vol. 38, no. 2, pp. 325–339, 1967.

[65] G. Shafer, *A Mathematical Theory of Evidence*. Princeton, NJ: Princeton Univ. Press, 1976.

[66] T.G. Dietterich, "Ensemble methods in machine learning," 1st Int. Workshop on Multiple Classifier Systems, in *Lecture Notes in Computer Sceince*, J. Kitler and F. Roli, Eds., vol. 1857, pp. 1–15, 2000.

[67] T.G. Dietterich, "Experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization," *Machine Learning*, vol. 40, no. 2, pp. 139–157, 2000.

[68] L. Breiman, "Arcing classifiers," *Annals of Statistics*, vol. 26, no. 3, pp. 801–849, 1998.

[69] E. Bauer and R. Kohavi, "An empiricial comparison of voting classification algorithms: bagging, boosting and variants," *Machine Learning*, vol. 36, pp. 105–142, 1999.

[70] J.R. Quinlan, "Bagging, boosting and C4.5, *13th Int. Conf. on Artificial Intelligence*, pp. 725–730, 1996.

[71] R.A. Jacobs, "Methods for combining experts' probability assessments," *Neural Computation*, vol. 7, no. 5, p. 867, 1995.

[72] R. Duin and D. Tax, "Experiments with Classifier Combining Rules," 1st Int. Workshop on Multiple Classifier Systems, in *Lecture Notes in Computer Science*, J. Kittler and F. Roli, Eds., vol. 1857, pp. 16–29, Springer, 2000.

[73] J. Ghosh, "Multiclassifier Systems: Back to the Future," 3rd Int. Workshop on Multiple Classifier Systems, in *Lecture Notes in Computer Science*, F. Roli and J. Kittler, Eds., vol. 2364, pp. 1–15, 2002.

[74] R. French, "Catastrophic forgetting in connectionist networks: causes, consequences and solutions," *Trends in Cognitive Sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[75] M. Muhlbaier, A. Topalis, and R. Polikar, "Learn++.MT: A New Approach to Incremental Learning," 5th Int. Workshop on Multiple Classifier Systems, in *Lecture Notes in Computer Sceince*, F. Roli, J. Kittler, and T. Windeatt, Eds., vol. 3077, pp. 52–61, Springer, 2004.

[76] R. Polikar, L. Udpa, S. Udpa, and V. Honavar, "An incremental learning algorithm with confidence estimation for automated identification of NDE signals," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 51, no. 8, pp. 990–1001, 2004.

[77] M. Muhlbaier, A. Topalis, and R. Polikar, "Incremental learning from unbalanced data," *IEEE Int. Joint Conf. on Neural Networks* vol. 2, pp. 1057–1062, Budapest, Hungary, 2004.

[78] H. Altincay and M. Demirekler, "Speaker identification by combining multiple classifiers using Dempster-Shafer theory of evidence," *Speech Communication*, vol. 41, no. 4, pp. 531–547, 2003.

[79] Y. Bi, D. Bell, H. Wang, G. Guo, and K. Greer, "Combining multiple classifiers using dempster's rule of combination for text categorization," in *Lecture Notes in Artificial Intelligence*, vol. 3131, pp. 127–138, Barcelona, Spain: Springer, 2004.

[80] T. Denoeux, "Neural network classifier based on Dempster-Shafer theory," *IEEE Trans. on Sys, Man, and Cyber. A: Sys. and Humans.*, vol. 30, no. 2, pp. 131–150, 2000.

[81] G.A. Carpenter, S. Martens, and O.J. Ogas, "Self-organizing information fusion and hierarchical knowledge discovery: a new framework using ARTMAP neural networks," *Neural Networks*, vol. 18, no. 3, pp. 287–295, Apr. 2005.

[82] D. Parikh, M.T. Kim, J. Oagaro, S. Mandayam, and R. Polikar, "Combining classifiers for multisensor data fusion," *IEEE Int. Conf. on Systems, Man and Cybernetics*, vol. 2, pp. 1232–1237, The Hague, Netherlands, 2004.

[83] B.F. Buxton, W.B. Langdon, and S.J. Barrett, "Data fusion by intelligent classifier combination," *Measurement and Control*, vol. 34, no. 8, pp. 229–234, 2001.

[84] G.J. Briem, J.A. Benediktsson, and J.R. Sveinsson, "Use of multiple classifiers in classification of data from multiple data sources," in *International Geoscience and Remote Sensing Symposium (IGARSS),* vol. 2, pp. 882–884, Sydney, 2001.

[85] W. Fan, M. Gordon, and P. Pathak, "On linear mixture of expert approaches to information retrieval," *Decision Support Systems*, vol. In Press, Corrected Proof.

[86] S. Jianbo, W. Jun, and X. Yugeng, "Incremental learning with balanced update on receptive fields for multi-sensor data fusion," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 34, no. 1, pp. 659–665, 2004.

[87] S. Gunter and H. Bunke, "An evaluation of ensemble methods in handwritten word recognition based on feature selection," *Int. Conf. on Pattern Recog.,* vol. 1, pp. 388–392, 2004.

[88] S. Krause and R. Polikar, "An Ensemble of Classifiers Approach for the Missing Feature Problem," in *Int. Joint Conf on Neural Networks,* vol. 1, pp. 553–556, Portland, OR, 2003.

[89] P.M. Long and V.B. Vega, "Boosting and Microarray Data," *Machine Learning*, vol. 52, no. 1–2, pp. 31–44, July 2003.

[90] E. Allwein, R.E. Schapire, and Y. Singer, "Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers," *Journal of Machine Learning Research*, vol. 1, pp. 113–141, 2000.

[91] T.G. Dietterich and G. Bakiri, "Solving Multiclass Learning Problems via Error-Correcting Output Codes," *Journal of Artificial Intel. Research*, vol. 2, pp. 263–286, 1995.

[92] S. Rajan and J. Ghosh, "An empirical comparison of hierarchical vs. two-level approaches to multiclass problems," in *Lecture Notes in Computer Science,* vol. 3077, pp. 283–292, F. Roli, J. Kittler, and T. Windeatt, Eds. Springer, 2004.

[93] R.S. Smith and T. Windeatt, "Decoding Rules for Error Correcting Output Code Ensembles," 6th Int. Workshop on Multiple Classifier Sys., *Lecture Notes in Computer Science,* N. C. Oza, R. Polikar, F. Roli, and J. Kittler, Eds., vol. 3541, pp. 53–63, Springer, 2005.

[94] H.G. Ayad and M.S. Kamel, "Cluster-Based Cumulative Ensembles," 6th Int. Workshop on Multiple Classifier Sys., *Lecture Notes in Computer Science,* N. C. Oza, R. Polikar, F. Roli, and J. Kittler, Eds., vol. 3541, pp. 236–245, Springer, 2005.

[95] S. Monti, P. Tamayo, J. Mesirov, and T. Golub, "Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data," *Machine Learning*, vol. 52, no. 1–2, pp. 91–118, July 2003.

[96] A. Strehl and J. Ghosh, "Cluster Ensembles-A Knowledge Reuse Framework for Combining Multiple Partitions," *Journal of Machine Learning Research*, vol. 3, pp. 583–617, 2002.

Robi Polikar received his B.S. degree in electronics and communications engineering from Istanbul Technical University, Istanbul, Turkey, in 1993, and his M.S. and Ph.D. degrees, both co-majors in biomedical engineering and electrical engineering, from Iowa State University, Ames, Iowa, in 1995 and in 2000, respectively. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering at Rowan University, Glassboro, NJ. His current research interests include signal processing, pattern recognition, neural systems, machine learning and computational models of learning, with applications to biomedical engineering and imaging; chemical sensing, nondestructive evaluation and testing. He also teaches upper level undergraduate and graduate courses in wavelet theory, pattern recognition, neural networks and biomedical systems and devices at Rowan University. He is a member of IEEE, ASEE, Tau Beta Pi and Eta Kappa Nu. His current work is funded primarily through NSF's CAREER program and NIH's Collaborative Research in Computational Neuroscience program.