Pattern Recognition 43 (2010) 3817–3832

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Learn++.MF: A random subspace approach for the missing feature problem

Robi Polikar [a,*], Joseph DePasquale [a], Hussein Syed Mohammed [a], Gavin Brown [b], Ludmilla I. Kuncheva [c]

[a] Electrical and Computer Eng., Rowan University, 201 Mullica Hill Road, Glassboro, NJ 08028, USA
[b] University of Manchester, Manchester, England, UK
[c] University of Bangor, Bangor, Wales, UK

ARTICLE INFO

Article history:
Received 9 November 2009
Received in revised form
16 April 2010
Accepted 21 May 2010

Keywords:
Missing data
Missing features
Ensemble of classifiers
Random subspace method

ABSTRACT

We introduce Learn++.MF, an ensemble-of-classifiers based algorithm that employs random subspace selection to address the missing feature problem in supervised classification. Unlike most established approaches, Learn++.MF does not replace missing values with estimated ones, and hence does not need specific assumptions on the underlying data distribution. Instead, it trains an ensemble of classifiers, each on a random subset of the available features. Instances with missing values are classified by the majority voting of those classifiers whose training data did not include the missing features. We show that Learn++.MF can accommodate substantial amount of missing data, and with only gradual decline in performance as the amount of missing data increases. We also analyze the effect of the cardinality of the random feature subsets, and the ensemble size on algorithm performance. Finally, we discuss the conditions under which the proposed approach is most effective.

1. Introduction

1.1. The missing feature problem

The integrity and completeness of data are essential for any classification algorithm. After all, a trained classifier – unless specifically designed to address this issue – cannot process instances with missing features, as the missing number(s) in the input vectors would make the matrix operations involved in data processing impossible. To obtain a valid classification, the data to be classified should be complete with no missing features (henceforth, we use *missing data* and *missing features* interchangeably). Missing data in real world applications is not uncommon: bad sensors, failed pixels, unanswered questions in surveys, malfunctioning equipment, medical tests that cannot be administered under certain conditions, etc. are all familiar scenarios in practice that can result in missing features. Feature values that are beyond the expected dynamic range of the data due to extreme noise, signal saturation, data corruption, etc. can also be treated as missing data. Furthermore, if the entire data are not acquired under identical conditions (time/location, using the same equipment, etc.), different data instances may be missing different features.

Fig. 1 illustrates such a scenario for a handwritten character recognition application: characters are digitized on an 8×8 grid, creating 64 features, f_1–f_{64}, a random subset (of about 20–30%) of which – indicated in orange (light shading) – are missing in each case. Having such a large proportion of randomly varying features may be viewed as an extreme and unlikely scenario, warranting reacquisition of the entire dataset. However, data reacquisition is often expensive, impractical, or sometimes even impossible, justifying the need for an alternative practical solution.

The classification algorithm described in this paper is designed to provide such a practical solution, accommodating missing features subject to the condition of *distributed redundancy* (discussed in Section 3), which is satisfied surprisingly often in practice.

1.2. Current techniques for accommodating missing data

The simplest approach for dealing with missing data is to ignore those instances with missing attributes. Commonly referred to as *filtering* or *list wise deletion* approaches, such techniques are clearly suboptimal when a large portion of the data have missing attributes [1], and of course are infeasible, if each instance is missing at least one or more features. A more pragmatic approach commonly used to accommodate missing data is *imputation* [2–5]: substitute the missing value with a meaningful estimate. Traditional examples of this approach include replacing the missing value with one of the existing data points (most similar in some measure) as in *hot – deck* imputation

* Corresponding author: Tel: +1 856 256 5372; fax: +1 856 256 5241.
E-mail address: polikar@rowan.edu (R. Polikar).

0031-3203/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2010.05.028

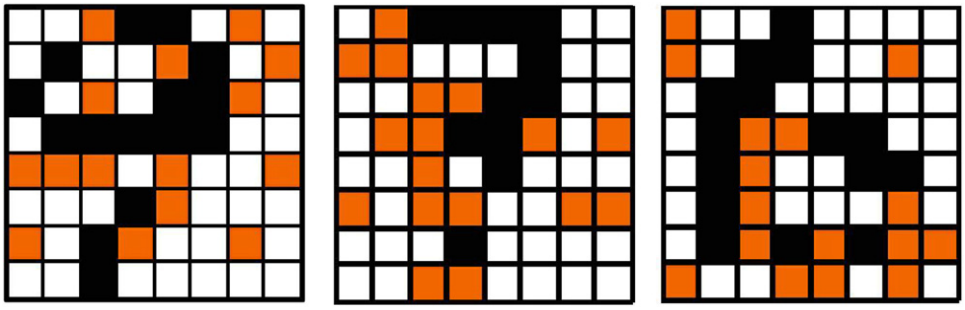

Fig. 1. Handwritten character recognition as an illustration of the missing feature problem addressed in this study: a large proportion of feature values are missing (orange/shaded), and the missing features vary randomly from one instance to another (the actual characters are the numbers 9, 7 and 6). For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.

[2,6,7], the mean of that attribute across the data, or the mean of its *k*-nearest neighbors [4]. In order for the estimate to be a faithful representative of the missing value, however, the training data must be sufficiently dense, a requirement rarely satisfied for datasets with even modest number of features. Furthermore, imputation methods are known to produce biased estimates as the proportion of missing data increases. A related, but perhaps a more rigorous approach is to use polynomial regression to estimate substitutes for missing values [8]. However, regression techniques – besides being difficult to implement in high dimensions – assume that the data can reasonably fit to a particular polynomial, which may not hold for many applications.

Theoretically rigorous methods with provable performance guarantees have also been developed. Many of these methods rely on *model based* estimation, such as Bayesian estimation [9–11], which calculates posterior probabilities by integrating over the missing portions of the feature space. Such an approach also requires a sufficiently dense data distribution; but more importantly, a prior distribution for all unknown parameters must be specified, which requires prior knowledge. Such knowledge is typically vague or non-existent, and inaccurate choices often lead to inferior performance.

An alternative strategy in model based methods is the Expectation Maximization (EM) algorithm [12–14,10], justly admired for its theoretical elegance. EM consists of two steps, the expectation (E) and the maximization (M) step, which iteratively maximize the expectation of the log-likelihood of the complete data, conditioned on observed data. Conceptually, these steps are easy to construct, and the range of problems that can be handled by EM is quite broad. However, there are two potential drawbacks: (i) convergence can be slow if large portions of data are missing; and (ii) the M step may be quite difficult if a closed form of the distribution is unknown, or if different instances are missing different features. In such cases, the theoretical simplicity of EM does not translate into practice [2]. EM also requires prior knowledge that is often unavailable, or estimation of an unknown underlying distribution, which may be computationally prohibitive for large dimensional datasets. Incorrect distribution estimation often leads to inconsistent results, whereas lack of sufficiently dense data typically causes loss of accuracy. Several variations have been proposed, such as using Gaussian mixture models [15,16]; or Expected Conditional Maximization, to mitigate some of these difficulties [2].

Neural network based approaches have also been proposed. Gupta and Lam [17] looked at weight decay on datasets with missing values, whereas Yoon and Lee [18] proposed the Training-EStimation-Training (TEST) algorithm, which predicts the actual target value from a reasonably well estimated imputed one. Other approaches use neuro-fuzzy algorithms [19], where unknown values of the data are either estimated (or a classification is made using the existing features) by calculating the fuzzy membership of the data point to its nearest neighbors, clusters or hyperboxes. The parameters of the clusters and hyperboxes are determined from the existing data. Algorithms based on the general fuzzy min–max neural networks [20] or ARTMAP and fuzzy c-means clustering [21] are examples of this approach.

More recently, ensemble based approaches have also been proposed. For example, Melville et al. [22] showed that the algorithm DECORATE, which generates artificial data (with no missing values) from existing data (with missing values) is quite robust to missing data. On the other hand, Juszczak and Duin [23] proposed combining an ensemble of one-class classifiers, each trained on a single feature. This approach is capable of handling any combination of missing features, with the fewest number of classifiers possible. The approach can be very effective as long as single features can reasonably estimate the underlying decision boundaries, which is not always plausible.

In this contribution, we propose an alternative strategy, called Learn^{++}.MF. It is inspired in part by our previously introduced ensemble-of-classifiers based incremental learning algorithm, Learn^{++}, and in part by the random subspace method (RSM). In essence, Learn^{++}.MF generates a sufficient number of classifiers, each trained on a random subset of features. An instance with one or more missing attributes is then classified by majority voting of those classifiers that did not use the missing features during their training. Hence, this approach differs in one fundamental aspect from the other techniques: Learn^{++}.MF does not try to estimate the values of the missing data; instead it tries to extract the most discriminatory *classification* information provided by the *existing* data, by taking advantage of a presumed redundancy in the feature set. Hence, Learn^{++}.MF avoids many of the pitfalls of estimation and imputation based techniques.

As in most missing feature approaches, we assume that the probability of a feature being missing is independent of the value of that or any other feature in the dataset. This model is referred to as *missing completely at random* (MCAR) [2]. Hence, given the dataset $\boldsymbol{X}=(x_{ij})$ where x_{ij} represents the jth feature of instance $\boldsymbol{x}_i$, and a missing data indicator matrix $\boldsymbol{M}=(M_{ij})$, where M_{ij} is 1 if x_{ij} is missing and 0 otherwise, we assume that $p(\boldsymbol{M}|\boldsymbol{X})=p(\boldsymbol{M})$. This is the most restrictive mechanism that generates missing data, the one that provides us with no additional information. However, this is also the only case where list-wise deletion or most imputation approaches lead to no bias.

In the rest of this paper, we first provide a review of ensemble systems, followed by the description of the Learn^{++}.MF algorithm, and provide a theoretical analysis to guide the choice of its free parameters: number of features used to train each classifier, and the ensemble size. We then present results, analyze algorithm performance with respect to its free parameters, and compare performance metrics with theoretically expected values. We also

compare the performance of Learn^{++}.MF to that of a single classifier using mean imputation, to naïve Bayes that can naturally handle missing features, and an intermediate approach that combines RSM with mean imputation, as well as to that of one-class ensemble approach of [23]. We conclude with discussions and remarks.

2. Background: ensemble and random subspace methods

An ensemble-based system is obtained by combining *diverse* classifiers. The diversity in the classifiers, typically achieved by using different training parameters for each classifier, allows individual classifiers to generate different decision boundaries [24–27]. If proper diversity is achieved, a different error is made by each classifier, strategic combination of which can then reduce the total error. Starting in early 1990s, with such seminal works as [28–30,13,31–33], ensemble systems have since become an important research area in machine learning, whose recent reviews can be found in [34–36].

The diversity requirement can be achieved in several ways [24–27]: training individual classifiers using different (i) training data (sub)sets; (ii) parameters of a given classifier architecture; (iii) classifier models; or (iv) subsets of features. The last one, also known as the *random subspace method* (RSM), was originally proposed by Ho [37] for constructing an ensemble of decision trees (decision forests). In RSM, classifiers are trained using different random subsets of the features, allowing classifiers to err in different sub-domains of the feature space. Skurichina points out that RSM works particularly well when the database provides redundant information that is "dispersed" across all features [38].

The RSM has several desirable attributes: (i) working in a reduced dimensional space also reduces the adverse consequences of the curse of dimensionality; (ii) RSM based ensemble classifiers may provide improved classification performance, because the synergistic classification capacity of a diverse set of classifiers compensates for any discriminatory information lost by choosing a smaller feature subset; (iii) implementing RSM is quite straightforward; and (iv) it provides a stochastic and faster alternative to the optimal-feature-subset search algorithms. RSM approaches have been well-researched for improving diversity, with well-established benefits for classification applications [39], regression problems [40] and optimal feature selection applications [38,41]. However, the feasibility of an RSM based approach has not been explored for the missing feature problem, and hence constitutes the focus of this paper.

Finally, a word on the algorithm name: Learn^{++} was developed by reconfiguring AdaBoost [33] to incrementally learn from new data that may later become available. Learn^{++} generates an ensemble of classifiers for each new database, and combines them through weighted majority voting. Learn^{++} draws its training data from a distribution, iteratively updated to force the algorithm to learn novel information not yet learned by the ensemble [42,43]. Learn^{++}.MF, combines the distribution update concepts of Learn^{++} with the random feature selection of RSM, to provide a novel solution to the *M*issing *F*eature problem.

3. Approach

3.1. Assumptions and targeted applications of the proposed approach

The essence of the proposed approach is to generate a sufficiently large number of classifiers, each trained with a random subset of features. When an instance $\mathbf{x}$ with missing feature(s) needs to be classified, only those classifiers *trained with the features that are presently available in* $\mathbf{x}$ are used for the classification. As such, Learn^{++}.MF follows an alternative paradigm for the solution of the missing feature problem: the algorithm tries to extract most of the information from the available features, instead of trying to estimate, or impute, the values of the missing ones. Hence, Learn^{++}.MF does not require a very dense training data (though, such data are certainly useful); it does not need to know or estimate the underlying distributions; and hence does not suffer from the adverse consequences of a potential incorrect estimate.

Learn^{++}.MF makes two assumptions. First, the feature set must be redundant, that is, the classification problem is solvable with unknown subset(s) of the available features (of course, if the identities of the redundant features were known, they would have already been excluded from the feature set). Second, the redundancy is distributed randomly over the feature set (hence, time series data would not be well suited for this approach). These assumptions are primarily due to the random nature of feature selection, and are shared with all RSM based approaches. We combine these two assumptions under the name *distributed redundancy*. Such datasets are not uncommon in practice (the scenario in Fig. 1, sensor applications, etc), and it is these applications for which Learn^{++}.MF is designed, and expected to be most effective.

3.2. Algorithm Learn^{++}.MF

A trivial approach is to create a classifier for each of the 2^f-1 possible non-empty subsets of the f features. Then, for any instance with missing features, use those classifiers whose training feature set did not include the missing ones. Such an approach, perfectly suitable for datasets with few features, has in fact been previously proposed [44]. For large feature sets, this exhaustive approach is infeasible. Besides, the probability of a particular feature combination being missing also diminishes exponentially as the number of features increase. Therefore, trying to account for every possible combination is inefficient. At the other end of the spectrum, is Juszczak and Duin's approach [23], using *fxc* one-class classifiers, one for each feature and each of the c classes. This approach requires fewest number of classifiers that can handle any feature combination, but comes at a cost of potential performance drop due to disregarding any existing relationship between the features, as well as the information provided by other existing features.

Learn^{++}.MF offers a strategic compromise: it trains an ensemble of classifiers with a subset of the features, randomly drawn from a feature distribution, which is iteratively updated to favor selection of those features that were previously undersampled. This allows Learn^{++}.MF to achieve classification performances with little or no loss (compared to a fully intact data) even when large portions of data are missing. Without any loss of generality, we assume that the training dataset is complete. Otherwise, a sentinel can be used to flag the missing features, and the classifiers are then trained on random selection of existing features. For brevity, we focus on the more critical case of field data containing missing features.

The pseudocode of Learn^{++}.MF is provided in Fig. 2. The inputs to the algorithm are the training data $S=\{(\mathbf{x}_i,y_i),i=1,\ldots,N\}$; feature subset cardinality; i.e., number of features, *nof*, used to train each classifier; a supervised classification algorithm (**BaseClassifier**), the number of classifiers to be created T; and a sentinel value *sen* to designate a missing feature. At each iteration t, the algorithm creates one additional classifier C_t. The features used for training C_t are drawn from an iteratively updated

Inputs:

- Sentinel value *sen*, BaseClassifier model and the number of classifiers, T.
- Training data set $\boldsymbol{S}=\{(\mathbf{x}_i, y_i)\,|\,i=1,\cdots,N\}$, with N instances of f features, each, from c classes.
- Number of features used to train each classifier, *nof*;

Initialize $D_1(j)=1/f, \forall j, j=1,\cdots,f$; $\beta=nof/f$ or $\beta=1/f$ or $\beta=1$

Do for t = 1, 2,..., T:

1. Normalize $\boldsymbol{D}_t$ so that it is a proper distribution;
2. Draw *nof* features for $\boldsymbol{F}_{selection}(t)$ from $\boldsymbol{D}_t$.
3. Call **BaseClassifier** to generate classifier C_t using features in $\boldsymbol{F}_{selection}(t)$.
4. Obtain $Perf_t$, the performance of C_t on $\boldsymbol{S}$. If $Perf_t < \theta$ (θ=0, ½ or 1/c), discard C_t and go to step 2.
5. Update feature distribution $\boldsymbol{D}_{t+1}\left(\boldsymbol{F}_{selection}(t)\right)=\beta * \boldsymbol{D}_t\left(\boldsymbol{F}_{selection}(t)\right)$, $0<\beta\le 1$. **end Do loop**

Using Trained Ensemble

Given test / field data **z**,

1. Determine missing features $\boldsymbol{M}(\mathbf{z})=\arg(\mathbf{z}(j)==sen)$, $\forall j, j=1,\cdots,f$.
2. Obtain ensemble decision as the class with the most votes among the outputs of classifiers $\boldsymbol{C}_t^*$ trained on the non-missing features: $\mathcal{E}(\mathbf{z})=\arg\max_{y\in Y}\sum_{t:C_t^*(\mathbf{z})=y}[\![\boldsymbol{M}(\mathbf{z})\cap \boldsymbol{F}_{selection}(t)=\varnothing]\!]$.

Fig. 2. Pseudocode of algorithm Learn^{++}.MF.

distribution $\boldsymbol{D}_t$ that promotes further diversity with respect to the feature combinations. $\boldsymbol{D}_1$ is initialized to be uniform, hence each feature is equally likely to be used in training classifier C_1 (step 1).

$$D_1(j)=1/f, \quad j=1,\cdots,f \tag{1}$$

where f is the total number of features. For each iteration t=1,...,T, the distribution $\boldsymbol{D}_t\in\mathbb{R}^f$ that was updated in the previous iteration is first normalized to ensure that $\boldsymbol{D}_t$ stays as a proper distribution

$$\boldsymbol{D}_t=\boldsymbol{D}_t\Big/\sum_{j=1}^{f}D_t(j) \tag{2}$$

A random sample of *nof* features is drawn, without replacement, according to $\boldsymbol{D}_t$, and the indices of these features are placed in a list, called $\boldsymbol{F}_{\text{selection}}(t)\in\mathbb{R}^{nof}$ (step 2). This list keeps track of which features have been used for each classifier, so that appropriate classifiers can be called during testing. Classifier C_t, is trained (step 3) using the features in $\boldsymbol{F}_{\text{selection}}(t)$, and evaluated on $\boldsymbol{S}$ (step 4) to obtain

$$Perf_t=\frac{1}{N}\sum_{i=1}^{N}[\![C_t(\boldsymbol{x}_i)=y_i]\!] \tag{3}$$

where $[\![\cdot]\!]$ evaluates to 1 if the predicate is true. We require C_t to perform better than some minimum threshold, such as 1/2 as in AdaBoost, or 1/c (better than the random guess for a reasonably well-balanced c-class data), on its training data to ensure that it has a meaningful classification capacity. The distribution $\boldsymbol{D}_t$ is then updated (step 5) according to

$$\mathbf{D}_{t+1}(\mathbf{F}_{selection}(t))=\beta * \boldsymbol{D}_t(\mathbf{F}_{selection}(t)), \quad 0<\beta\le 1 \tag{4}$$

such that the weights of those features in current $\boldsymbol{F}_{selection}(t)$ are reduced by a factor of β. Then, features not in $\boldsymbol{F}_{selection}(t)$ effectively receive higher weights when $\boldsymbol{D}_t$ is normalized in the next iteration. This gives previously undersampled features a better chance to be selected into the next feature subset. Choosing $\beta=1$ results in a feature distribution that remains uniform throughout the experiment. We recommend $\beta=nof/f$, which reduces the likelihood of previously selected features being selected in the next iteration. In our preliminary efforts, we obtained better performances by using $\beta=nof/f$ over other choices such as $\beta=1$ or $\beta=1/f$, though not always with statistically significant margins [45].

During classification of a new instance **z**, missing (or known to be corrupt) values are first replaced by a sentinel *sen* (chosen as a value not expected to occur in the data). Then, the features with the value *sen* are identified and placed in $\boldsymbol{M}(\mathbf{z})$, the set of missing features.

$$\mathbf{M}(\mathbf{z})=\arg(\mathbf{z}(j)==sen), \quad \forall j, j=1,\ldots,f, \tag{5}$$

Finally, all classifiers C_t^* whose feature list $\boldsymbol{F}_{selection}(t)$ does not include the features listed in $\boldsymbol{M}(\mathbf{z})$ are combined through majority voting to obtain the ensemble classification of **z**.

$$\mathbf{E}(\mathbf{z})=\arg\max_{y\in Y}\sum_{t:C_t^*(\mathbf{z})=y}[\![(\mathbf{M}(\mathbf{z})\cap\mathbf{F}_{selection}(t))=\emptyset]\!] \tag{6}$$

3.3. Choosing algorithm parameters

Considering that feature subsets for each classifier are chosen randomly, and that we wish to use far fewer than 2^f-1 classifiers necessary to accommodate every feature subset combination, it is possible that a particular feature combination required to accommodate a specific set of missing features might not have been sampled by the algorithm. Such an instance cannot be processed by Learn^{++}.MF. It is then fair to ask how often Learn^{++}.MF will not be able to classify a given instance. Formally, given that Learn^{++}.MF needs at least one useable classifier that can handle the current missing feature combination, what is the probability that there will be at least one classifier – in an ensemble of T classifiers – that will be able to classify an instance with m of its f features missing, if each classifier is trained on $nof<f$ features? A formal analysis to answer this question also allows us to determine the relationship among these parameters, and hence can guide us in appropriately selecting the algorithm's free parameters: *nof* and T.

A classifier C_t is only useable if none of the *nof* features selected for its training data matches one of those missing in **x**. Then, what is the probability of finding exactly such a classifier? Without loss of any generality, assume that m missing features are fixed, and that we are choosing *nof* features to train C_t. Then, the probability of choosing the first feature – among f features – such that it does not coincide with any of the m missing ones is $(f-m)/f$. The probability of choosing the second such feature is $(f-m-1)/(f-1)$, and similarly the probability of choosing the $(nof)^{\text{th}}$ such feature is $(f-m-(nof-1))/(f-(nof-1))$. Then the probability of

finding a useable classifier C_t for $\mathbf{x}$ is

$$p=\frac{f-m}{f}\cdot\frac{f-m-1}{f-1}\cdot\frac{f-m-2}{f-2}\cdot\cdots\cdot\frac{f-m-(nof-1)}{f-(nof-1)}$$
$$=\left(1-\frac{m}{f}\right)\cdot\left(1-\frac{m}{f-1}\right)\cdot\left(1-\frac{m}{f-2}\right)\cdot\cdots\cdot\left(1-\frac{m}{f-(nof-1)}\right)=\prod_{i=0}^{nof-1}\left(1-\frac{m}{f-i}\right) \tag{7}$$

Note that this is exactly the scenario described by the hypergeometric (HG) distribution: we have f features, m of which are missing; we then randomly choose nof of these f features. The probability that exactly k of these selected nof features will be one of the m missing features is

$$p\sim HG(k;f,m,nof)=\binom{m}{k}\binom{f-m}{nof-k}\Big/\binom{f}{nof} \tag{8}$$

We need $k=0$, i.e., none of the features selected is one of the missing ones. Then, the desired probability is

$$p\sim HG(k=0;f,m,nof)=\frac{\binom{m}{0}\binom{f-m}{nof}}{\binom{f}{nof}}=\frac{(f-m)!}{(f-m-nof)!}\cdot\frac{(f-nof)!}{f!} \tag{9}$$

Eqs. (7) and (9) are equivalent. However, Eq. (7) is preferable, since it does not use factorials, and hence will not result in numerical instabilities for large values of f. If we have T such classifiers, each trained on a randomly selected nof features, what is the probability that at least one will be useable, i.e., at least one of them was trained with a subset of nof features that do not include one of the m missing features. The probability that a random single classifier is not useable is $(1-p)$. Since the feature selections are independent (specifically for $\beta=1$), the probability of not finding any single useable classifier among all T classifiers is $(1-p)^T$. Hence, the probability of finding at least one useable classifier is

$$P=1-(1-p)^T=1-\left(1-\prod_{i=0}^{nof-1}\left(1-\frac{m}{f-i}\right)\right)^T \tag{10}$$

If we know how many features are missing, Eq. (10) is exact. However, we usually do not know exactly how many features will be missing. At best, we may know the probability ρ that a given feature may be missing. We then have a binomial distribution: naming the event " a given feature is missing" as success, then the probability p_m of having m of f features missing in any given instance is the probability of having m successes in f trials of a Bernoulli experiment, characterized by the binomial distribution

$$p_m=\binom{f}{m}\rho^m(1-\rho)^{f-m} \tag{11}$$

The actual probability of finding a useable classifier is then a weighted sum of the probabilities $(1-p)^T$, where the weights are the probability of having exactly m features missing, with m varying from 0 (no missing features) to maximum allowable number of $f-nof$ missing features.

$$P=1-\sum_{m=0}^{f-nof}\left[\binom{f}{m}\rho^m(1-\rho)^{f-m}\cdot\left(1-\prod_{i=0}^{nof-1}\left(1-\frac{m}{f-i}\right)\right)^T\right] \tag{12}$$

Eq. (12) computes the probability of finding at least one useable classifier by subtracting the probability of finding no useable classifier (in an ensemble of T classifiers) from one. A more informative way, however, is to compute the probability of finding t useable classifiers (out of T). If the probability of finding a single useable classifier is a success in a Bernoulli experiment with probability of success p, then the probability of finding at least t useable classifiers is

$$P(>t \text{ useable classifiers when } m \text{ features are missing})=p_m^{\geq t}$$
$$=\sum_{\tau=t}^{T}\binom{T}{\tau}(p)^{\tau}(1-p)^{T-\tau} \tag{13}$$

Note that Eq. (13) still needs to be weighted with the (binomial) probabilities of "having exactly m missing features", and then summed over all values of m to yield the desired probability

$$P=\sum_{m=0}^{f-nof}\left[\binom{f}{m}\rho^m(1-\rho)^{f-m}\cdot(p_m^{\geq t})\right] \tag{14}$$

By using the appropriate expressions from Eqs. (7) and (13) in Eq. (14), we obtain

$$P=\sum_{m=0}^{f-nof}\left[\binom{f}{m}\rho^m(1-\rho)^{f-m}\cdot\left(\sum_{\tau=t}^{T}\binom{T}{\tau}\left(\prod_{i=0}^{nof-1}\left(1-\frac{m}{f-i}\right)\right)^{\tau}\times\left(1-\prod_{i=0}^{nof-1}\left(1-\frac{m}{f-i}\right)\right)^{T-\tau}\right)\right] \tag{15}$$

Eq. (15), when evaluated for $t=1$, is the probability of finding at least one useable classifier trained on nof features – from a pool of T classifiers – when each feature has a probability ρ of being missing. For $t=1$, Eqs. (10) and (13) are identical. However, Eq. (13) allows computation of the probability of finding *any* number of useable classifiers (say, $t_{desired}$, not just one) simply by starting the summation from $\tau=t_{desired}$. These equations can help us choose the algorithm's free parameters, nof and T, to obtain a reasonable confidence that we can process most instances with missing features.

As an example, consider a 30-feature dataset. First, let us fix probability that a feature is missing at $\rho=0.05$. Then, on average, 1.5 features are missing from each instance. Fig. 3(a) shows the probability of finding at least one classifier for various values of nof as a function of T. Fig. 3(b) provides similar plots for a higher rate of $\rho=0.20$ (20% missing features). The solid curve in each case is the curve obtained by Eq. (15), whereas the various indicators represent the actual probabilities obtained as a result of 1000 simulations. We observe that (i) the experimental data fits the theoretical curve very well; (ii) the probability of finding a useable classifier increases with the ensemble size; and (iii) for a fixed ρ and a fixed ensemble size, the probability of finding a useable classifier increases as nof decreases. This latter observation makes sense, as fewer the number of features used in training, the larger the number of missing features that can be accommodated. In fact, if a classifier is trained with nof features out of a total of f features, then that classifier can accommodate up to $f-nof$ missing features.

In Fig. 3(c), we now fix $nof=9$ (30% of the total feature size), and calculate the probability of finding at least one useable classifier as a function of ensemble size, for various values of ρ. As expected, for a fixed value of nof and ensemble size, this probability decreases as ρ increases.

In order to see how these results scale with larger feature sets, Fig. 4 shows the theoretical plots for the 216—feature multiple features (MFEAT) dataset [46]. All trends mentioned above can be observed in this case as well, however, the required ensemble size is now in thousands. An ensemble with a few thousand classifiers is well within today's computational resources, and in fact not uncommon in many multiple classifier system applications. It is clear, however, that as the feature size grows beyond the

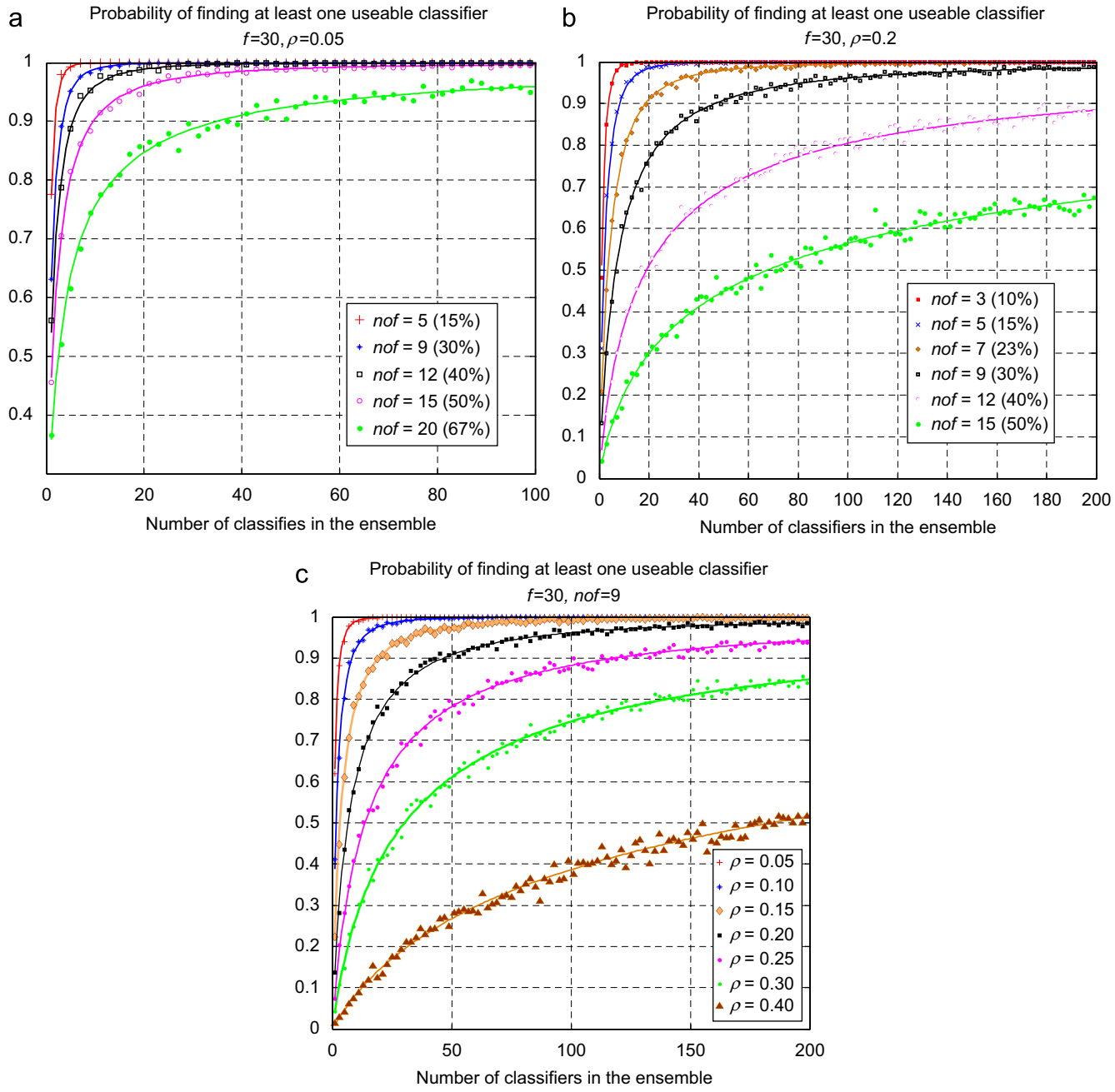

Fig. 3. Probability of finding at least one useable classifier for $f=30$ (a) $\rho=0.05$, variable *nof*, (b) $\rho=0.2$ variable *nof* and (c) *nof*=9, variable ρ.

thousands range, the computational burden of this approach greatly increases. Therefore, Learn^{++}.MF is best suited for applications with moderately high dimensionality (fewer than 1000). This is not an overly restrictive requirement, since many applications have typically much smaller dimensionalities.

4. Simulation results

4.1. Experimental setup and overview of results

Learn^{++}.MF was evaluated on several databases with various number of features. The algorithm resulted in similar trends in all cases, which we discuss in detail below. Due to the amount of detail involved with each dataset, as well as for brevity and to avoid repetition of similar outcomes, we present representative full results on four datasets here, and provide results for many additional real world and UCI benchmark datasets online [48], a summary of which is provided at the end of this section. The four datasets featured here encompass a broad range of feature and database sizes. These datasets are the Wine (13 features), Wisconsin Breast Cancer (WBC—30 features), Optical Character Recognition (OCR—62 features) and Multiple Features (MFEAT—216 features) datasets obtained from the UCI repository [46]. In all cases, multilayer perceptron type neural networks were used as the base classifier, though any supervised classifier can also be used. The training parameters of the base classifiers were not fine tuned, and selected as fixed reasonable values (error goal = 0.001–0.01, number of hidden layer nodes = 20–30 based on the dataset). Our goal is not to show the absolute best classification

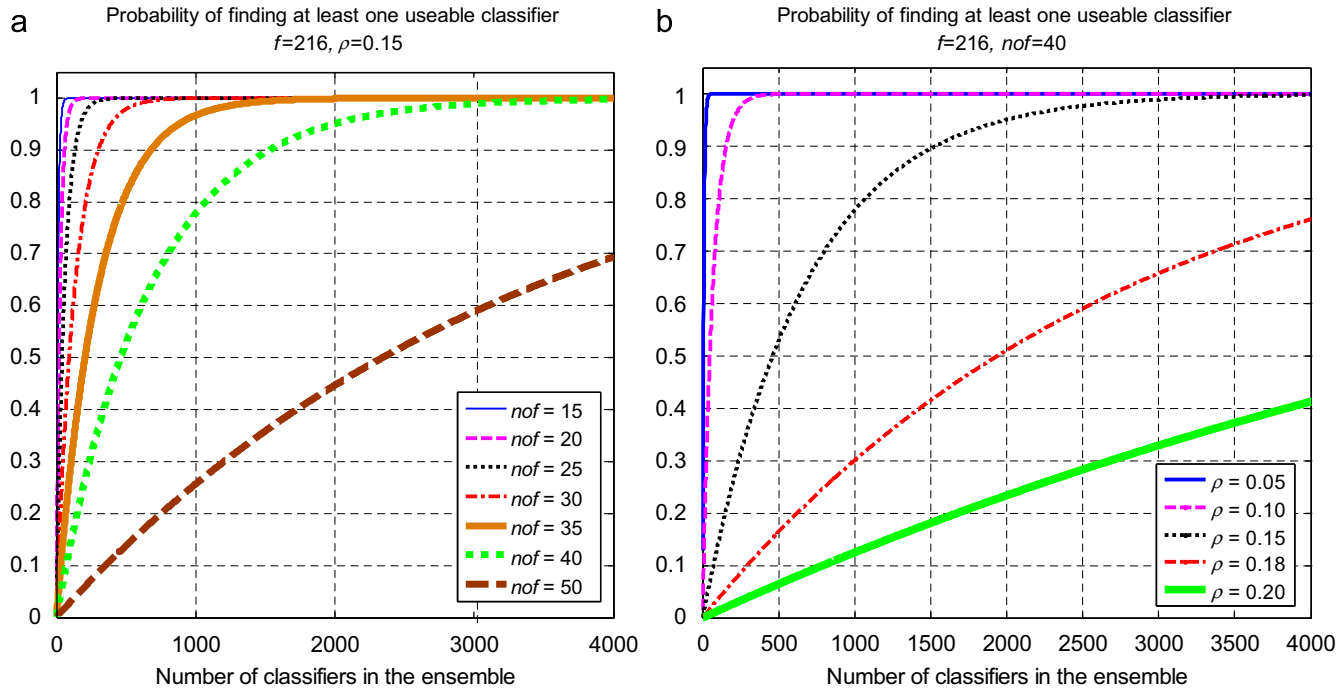

Fig. 4. Probability of finding at least one classifier, $f=216$ (a) $\rho=0.15$, variable *nof* and (b) *nof*$=40$, variable ρ.

Table 1
Datasets used in this study.

Dataset ↓	Dataset size	Number of classes	Number of features	nof_1	nof_2	nof_3	nof_4	nof_5	nof_6	T
WINE	178	3	13	3	4	5	6	7	variable	200
WBC	400	2	30	10	12	14	16	–	variable	1000
OCR	5620	10	62	16	20	24	–	–	variable	1000
MFEAT	2000	10	216	10	20	30	40	60	variable	2000
VOC-I*	384	6	6	2	3	–	–	–	variable	100
VOC-II*	84	12	12	3	4	5	6	–	variable	200
ION*	270	2	34	8	10	12	14	–	variable	1000
WATER*	374	4	38	12	14	16	18	–	variable	1000
ECOLI*	347	5	5	2	3	–	–	–	variable	1000
DERMA*	366	6	34	8	10	12	14	–	variable	1000
PEN*	10,992	10	16	6	7	8	9	–	variable	250

* The full results for these datasets are provided online due to space considerations and may be found at [48].

performances (which are quite good), but rather the effect of the missing features and strategic choice of the algorithm parameters.

We define the *global number of features* in a dataset as the product of the number of features f, and the number of instances N; and a single *test trial* as evaluating the algorithm with 0–30% (in steps of 2.5%), of the global number of features missing from the test data (typically half of the total data size). We evaluate the algorithm with different number of features, *nof*, used in training individual classifiers, including a variable *nof*, where the number of features used to train each classifier is determined by randomly drawing an integer in the 15–50% range of f. All results are averages of 100 trials on test data, reported with 95% confidence intervals, where training and test sets were randomly reshuffled for each trial. Missing features were simulated by randomly replacing actual values with sentinels.

Table 1 summarizes datasets and parameters used in the experiments: the cardinality of the dataset, the number of classes, total number of features f, the specific values of *nof* used in simulations, and the total number of classifiers T generated.

The behavior of the algorithm with respect to its parameters was very consistent across the databases. The two primary observations, discussed below in detail, were as follows: (i) the algorithm can handle data with missing values, 20% or more, with little or no performance drop, compared to classifying data with all values intact; and (ii) the choice of *nof* presents a critical trade-off: higher performance can be obtained using a larger *nof*, when fewer features are missing, yet smaller *nof* yields higher and more stable performance, and leaves fewer instances unprocessed, when larger portions of the data are missing.

4.2. Wine database

Wine database comes from the chemical analysis of 13 constituents in wines obtained from three cultivars. Previous experiments using optimized classifiers trained with *all* 13 features, and evaluated on a test dataset with no missing features, had zero classification error, setting the benchmark target performance for this database. Six values of *nof* were considered: 3, 4, 5, 6, 7 and *variable*, where each classifier was trained on – not with a fixed number of features, but a random selection of one of these values.

Table 2
Performance on the WINE database.

nof=3 (out of 13)			*nof*=4 (out of 13)			*nof*=5 (out of 13)		
PMF	**Ensemble Perf**	**PIP**	**PMF**	**Ensemble Perf**	**PIP**	**PMF**	**Ensemble Perf**	**PIP**
0.00	100.00 ± 0.00	100	0.00	100.00 ± 0.00	100	0.00	100.00 ± 0.00	100
2.50	100.00 ± 0.00	100	2.50	99.67 ± 0.71	100	2.50	100.00 ± 0.00	100
5.00	100.00 ± 0.00	100	5.00	100.00 ± 0.00	100	5.00	99.67 ± 0.71	100
7.50	99.67 ± 0.71	100	7.50	99.67 ± 0.71	100	7.50	99.33 ± 1.43	100
10.00	99.33 ± 0.95	100	10.00	99.67 ± 0.71	100	10.00	99.00 ± 1.09	100
15.00	99.00 ± 1.09	100	15.00	98.00 ± 1.58	100	15.00	98.33 ± 1.60	100
20.00	99.00 ± 1.09	100	20.00	98.00 ± 1.91	100	20.00	96.67 ± 1.85	100
25.00	99.00 ± 1.09	100	25.00	97.66 ± 1.53	100	25.00	96.29 ± 1.31	99
30.00	96.33 ± 1.98	100	30.00	94.29 ± 1.54	99	30.00	95.22 ± 2.26	98
nof=6 (out of 13)			*nof*=7 (out of 13)			*nof*=variable (3–7 out of 13)		
PMF	**Ensemble Perf**	**PIP**	**PMF**	**Ensemble Perf**	**PIP**	**PMF**	**Ensemble Perf**	**PIP**
0.00	100.00 ± 0.00	100	0.00	100.00 ± 0.00	100	0.00	100.00 ± 0.00	100
2.50	99.67 ± 0.71	100	2.50	99.33 ± 0.95	100	2.50	99.33 ± 0.95	100
5.00	99.33 ± 0.95	100	5.00	99.33 ± 0.95	100	5.00	100.00 ± 0.00	100
7.50	98.67 ± 1.58	100	7.50	98.67 ± 1.17	100	7.50	99.67 ± 0.71	100
10.00	98.33 ± 2.20	100	10.00	98.32 ± 1.60	100	10.00	99.67 ± 0.74	100
15.00	96.98 ± 2.00	99	15.00	96.94 ± 2.30	99	15.00	97.67 ± 1.53	100
20.00	96.98 ± 2.27	99	20.00	91.27 ± 3.55	96	20.00	98.00 ± 1.17	100
25.00	93.72 ± 2.66	95	25.00	93.03 ± 4.35	86	25.00	97.33 ± 1.78	100
30.00	95.29 ± 2.63	93	30.00	91.34 ± 4.20	78	30.00	94.67 ± 2.18	100

Table 2 summarizes the test performances using all values of *nof*, and the percentage of instances that could be processed – correctly or otherwise – with the existing ensemble (described below). The first row with 0.0% missing features is the algorithm's performance when classifiers were trained on *nof* features, but evaluated on a fully intact test data. The algorithm is able to obtain 100% correct classification with any of the *nof* values used, indicating that this dataset does in fact include redundant features.

Now, since the feature subsets are selected at random, it is possible that the particular set of features available for a given instance do not match the feature combinations selected by any of the classifiers. Such an instance cannot be processed, as there would be no classifier trained with the unique combination of the available features. The column with the heading PIP (percentage of instances processed) indicates the percentage of those instances that can be processed by the generated ensemble. As expected, PIP decreases as the "% missing features (PMF)" increases. We note that with 30% of the features missing from the test data, 100% of this test dataset can be processed (at a performance of 96.3%) when *nof*=3, whereas only 78% can be processed (at a performance of 91.3%) when *nof*=7 features are used for training the classifiers. A few additional observations: first, we note that there is no (statistically significant) performance drop with up to 25% of features missing when *nof*=3, or up to 10% when *nof*=6, and only a few percentage points (3.5% for *nof*=3, 4.7% for *nof*=6) when as many as 30% of the features are missing—hence the algorithm can handle large amounts of missing data with relatively little drop in classification performance. Second, virtually all test data can be processed—even when as much as 30% of the features are missing, if we use relatively few features for training the classifiers.

For a more in depth analysis of the algorithm's behavior, consider the plots in Fig. 5. Fig. 5(a) illustrates the overall performance of the Learn^{++}.MF ensemble. All performances in the 0–30% interval of PMF with increments of 2.5% are provided (while the tables skip some intermediate values for brevity). As expected, the performance declines as PMF increases – albeit only gradually. The decline is much milder for *nof*=3 than it is for *nof*=7, the reasons of which are discussed below. Also included in Fig. 5(a) for comparison, is the performance of a single classifier (of the same architecture as ensemble members) that employs the commonly used *mean imputation* (M.I.) to replace the missing data, as well as that of the E.M. algorithm [47]. We observe that Learn^{++}.MF ensemble (significantly) outperforms both E.M. and mean imputation for all values of PMF, an observation that was consistent for all databases we have evaluated (we compare Learn^{++}.MF to *naïve Bayes* and *RSM with mean imputation* later in the paper). Fig. 5(b) shows the performance of a single usable classifier, averaged over all T classifiers, as a function of PMF. The average performance of any fixed single classifier is, as expected, independent of PMF; however, using higher *nof* values are generally associated with better individual classifier performance. This makes sense, since there is more information available to each classifier with a larger *nof*. However, this comes at the cost of smaller PIP. In Fig. 5(b) and (c) we compare the performance and the PIP with a single vs. ensemble classifier. For example, consider 20% PMF, for which the corresponding PIP values for single classifiers are shown in Fig. 5(b). A single usable classifier trained with *nof*=3 features is able to classify, on average, 51% of instances with a performance of 79%, whereas the ensemble has a 100% PIP with a performance of 99%. Fig. 5(c) shows PIP as a function of PMF, both for single classifiers (lower curves) and the ensembles (upper curves). As expected, PIP decreases with increasing PMF for both the ensemble and a single usable classifier. However, the decline in PIP is much steeper for single classifiers. In fact, there is virtually no drop (PIP=100%) up until 20% PMF when the ensemble is used (regardless of *nof*). Hence the impact of using an ensemble is two-folds: the ensemble can process a larger percentage of instances with missing features, and it also provides a higher classification performance on those instances, compared to a single classifier.

Also note in Fig. 5(c) that the decline in PIP is much steeper with *nof*=7 than it is for *nof*=3, similar to decline in classification performance shown in Fig. 5(a). Hence, our main observation regarding the trade-off one must accept by choosing a specific *nof* value is as follows: classifiers trained with a larger *nof* may achieve a higher generalization performance individually, or initially when fewer features are missing, but are only able to classify fewer instances as the PMF increases. This observation can be explained as follows: using large *nof* for training

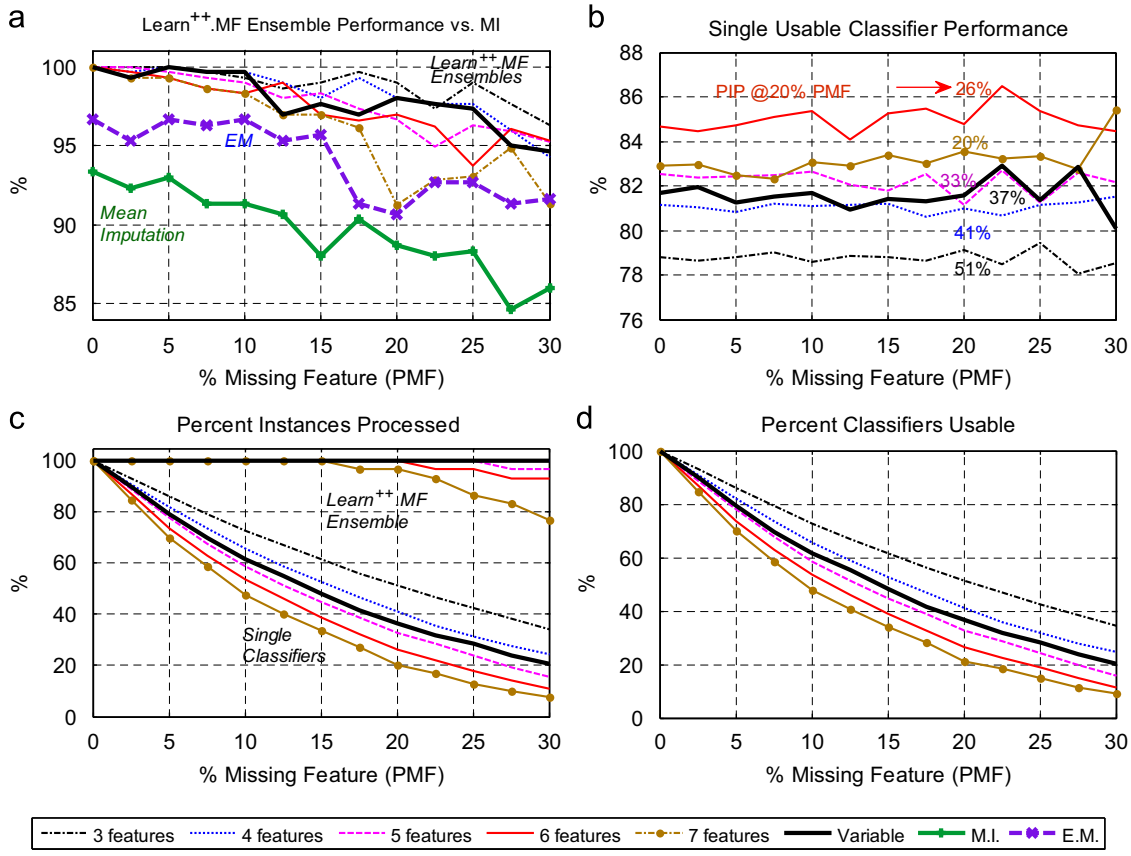

Fig. 5. Detailed analysis of the algorithm on the WINE database with respect to PMF and *nof*: (a) Learn++.MF ensemble performances compared to that of mean imputation, (b) single classifier performance, (c) single classifier and ensemble PIP and (d) percentage of useable classifiers in the ensemble.

(e.g., 10 out of 13) means that the same large number of features (10) are required for testing. Therefore, fewer number of missing features (only 3, in this case) can be accommodated. In fact, the probability of no classifier being available for a given combination of missing features increases with *nof* . Of course, if all features are used for training, then no classifier will be able to process any instance with even a single missing feature, hence the original motivation of this study.

Finally, Fig. 5(d) analyzes the average percentage of usable classifiers (PUC) per given instance, with respect to PMF. The decline in PUC also makes sense: there are fewer classifiers available to classify instances that have a higher PMF. Furthermore, this decline is also steeper for higher *nof*.

In summary, then, a larger *nof* usually provides a better individual and/or initial performance than a smaller *nof*, but the ensemble is more resistant to unusable feature combinations when trained with a smaller *nof*. How do we, then, choose the *nof* value, since we cannot know PMF in future datasets? A simple solution is to use a *variable* number of features—typically in the range of 15–50% of the total number of features. We observe from Table 2 and Fig. 5, that such a variable selection provides surprisingly good performance: a good compromise with little or no loss of unprocessed instances.

As for the other parameter, *T*—the number of classifiers, the trade off is simply a matter of computational resources. A larger PIP and PUC will be obtained, if larger number of classifiers is trained. Fig. 5(d) shows PUC for various values of PMF, indicating the actual numbers obtained for this database *after* the classifiers were trained. Family of curves generated using Eq. (15), such as those in Fig. 3, as well as Monte Carlo simulations can provide us with these numbers, *before* training the classifiers. Fig. 6 shows the results of averaging 1000 such Monte Carlo simulations for $f=13$, $nof=3$–7, PMF [0–100]%, which agrees with the actual results of Fig. 5(d) (up to 30% PMF is shown in Fig. 7(d)).

4.3. Wisconsin breast cancer (WBC) database

The WBC database consists of 569 instances of cell-biopsy classified as malignant or benign based on 30 features obtained from the individual cells. The expected best performance from this dataset (when no features are missing) is 97% [46]. Five *nof* values 10, 12, 14, 16 and *variable*, were tested.

Table 3 summarizes the test performances using all values of *nof*, and the PIP values, similar to that of Wine database in Table 2. The results with this database of larger number of features have much of the same trends and characteristics mentioned for the Wine database. Specifically, the proximity of the ensemble performance when used with $nof=10$, 12, 14 and 16 features with no missing features to the 97% figure reported in the literature indicates that this database also has a redundant feature set. Also, as in the Wine dataset, the algorithm is able to handle up to 20% missing features with little or no (statistically) significant performance drop, and only a gradual decline thereafter. As expected, the PIP drops with increasing PMF, and the drop is steeper for larger *nof* values compared to smaller ones. Using variable number of features provides a good trade off. These characteristics can also be seen in Fig. 7, where we plot the ensemble performance, and PIP for single and ensemble classifiers as in Fig. 5 for Wine database. Fig. 7(a) also shows that the ensemble outperforms a single classifier (of identical architecture) that uses mean imputation to replace the missing features. The

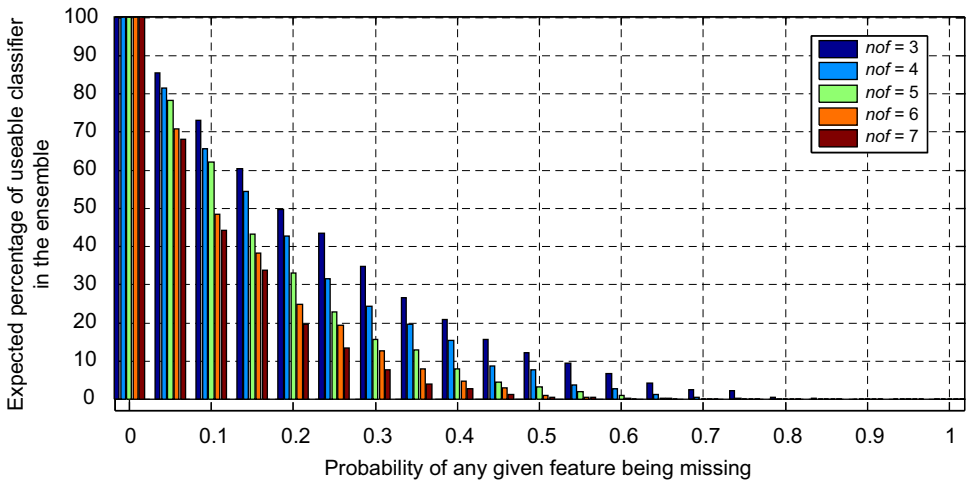

Fig. 6. Estimating the required ensemble size ahead of time.

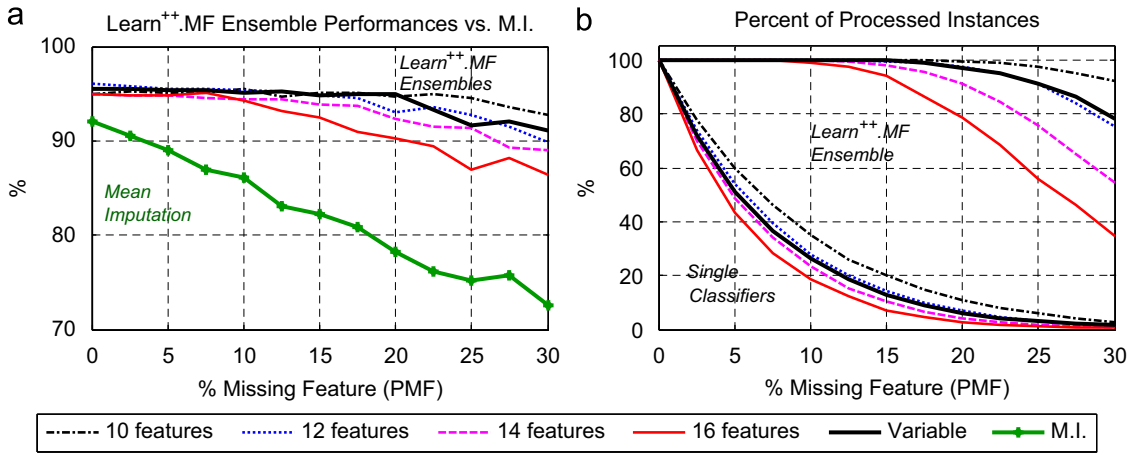

Fig. 7. Analysis of the algorithm on the WBC database with respect to PMF and *nof*: (a) Learn^{++}.MF ensemble performances compared to that of mean imputation and (b) single classifier and ensemble PIP.

Table 3
Performance on the WBC database.

nof=10 (out of 30)			*nof*=12 (out of 30)			*nof*=14 (out of 30)		
PMF	**Ensemble Perf**	**PIP**	**PMF**	**Ensemble Perf**	**PIP**	**PMF**	**Ensemble Perf**	**PIP**
0.00	95.00 ± 0.00	100	0.00	96.00 ± 0.00	100	0.00	95.00 ± 0.00	100
2.50	95.20 ± 0.24	100	2.50	95.80 ± 0.18	100	2.50	94.75 ± 0.24	100
5.00	95.05 ± 0.30	100	5.00	95.55 ± 0.41	100	5.00	94.80 ± 0.33	100
7.50	95.15 ± 0.42	100	7.50	95.55 ± 0.11	100	7.50	94.55 ± 0.56	100
10.00	95.55 ± 0.30	100	10.00	95.34 ± 0.40	100	10.00	94.39 ± 0.31	100
15.00	95.10 ± 0.69	100	15.00	94.88 ± 0.50	100	15.00	93.84 ± 0.95	98
20.00	94.63 ± 0.77	100	20.00	93.08 ± 1.10	97	20.00	92.35 ± 1.08	91
25.00	94.53 ± 0.81	98	25.00	92.78 ± 0.74	91	25.00	91.42 ± 1.20	76
30.00	92.78 ± 1.39	92	30.00	89.87 ± 2.02	75	30.00	89.01 ± 1.73	55

nof=16 (out of 30)			*nof*=var. (10–16 out of 30)		
PMF	**Ensemble Perf**	**PIP**	**PMF**	**Ensemble Perf**	**PIP**
0.00	95.00 ± 0.00	100	0.00	95.50 ± 0.00	100
2.50	94.80 ± 0.29	100	2.50	95.45 ± 0.19	100
5.00	94.75 ± 0.37	100	5.00	95.30 ± 0.43	100
7.50	95.04 ± 0.52	100	7.50	95.30 ± 0.46	100
10.00	94.24 ± 0.48	99	10.00	95.15 ± 0.39	100
15.00	92.50 ± 0.98	94	15.00	94.79 ± 0.85	100
20.00	90.20 ± 1.10	79	20.00	94.90 ± 0.79	97
25.00	86.98 ± 2.45	56	25.00	91.61 ± 1.23	91
30.00	86.34 ± 3.17	34	30.00	91.13 ± 1.32	78

margin of improvement of Learn^{++}.MF over mean imputation also widens as PMF increases. Due to their consistent behavior and for brevity, single classifier performances and PUC plots for this and other datasets are provided in [48].

4.4. Optical character recognition (OCR) database

The 10-class OCR database consists of 5620 instances of handwritten numerical characters 0–9 digitized on an 8×8 matrix to provide 64 total features. Two features had consistent values of zero for all instances, and hence were removed. The reported best performances on this dataset are around 97% when no features are missing. Four *nof* values of 16, 20, 24 and *variable* were used. The generalization performances as well as the PIP values for 0–30% PMF can be seen in Table 4 and Fig. 8. Single classifier performances were in the 80–86% range (based on *nof*) [48]. Once again, we make similar observations that the algorithm can accommodate up to 20% missing data with little or no loss of performance, or unprocessed instances, particularly for smaller *nof* values with a gradual decline in performance for larger PMF values. As in previous datasets, the ensemble based Learn^{++}.MF algorithm also performs significantly better than the commonly used mean imputation.

4.5. Multiple features (MFEAT) database

This database is similar to the OCR database and contains digitized handwritten numerical characters. However, this dataset consists of different sets of features (hence the name), a total of 649 features including 76 Fourier coefficients, 216 profile correlations, 64 PCA coefficients, etc. We choose the largest subset, 216 feature set for this experiment, making this the largest one in terms of feature size. Six different *nof* values were used, 10, 20, 30, 40, 50 and *variable*. Generalization performances as well as the PIP values for 0–30% PMF can be seen in Table 5 and Fig. 9. Single classifier performances were in the 81–91% range based on the *nof* values [48]. Duin (and Jain et al.) who contributed this dataset to UCI report error rates in the 4–40% range depending on the classification algorithm, using the entire feature set. Training with as few as randomly selected 10 of 216 features and obtaining a 95–96% generalization accuracy indicates that this dataset also includes a very redundant set of features. Using *nof*=10, Learn^{++}.MF was able to achieve 94.4% classification accuracy (less than 1% drop), while processing 100% of the instances. As expected, the classification performance declines gradually with increasing *nof* as well as increasing PMF, whereas PIP drops much faster with both parameters. For example, when using *nof*=50, the algorithm cannot find any useable classifiers once PMF reaches 25%. Interestingly, however, using a variable number of features—drawn randomly from uniform distribution in the 10–50—range can process 100% of the instances with 93% performance. Also, since this is the dataset with the largest number of features, we also compare the results to both mean imputation and EM. We observe that Learn^{++}.MF outperforms both of these "estimation" based approaches, the difference being substantially more significant with respect to mean imputation.

Table 4
Performance on the OCR database.

nof=16 (out of 62)			*nof*=20 (out of 62)			*nof*=24 (out of 20)			*nof*=var. (16–24 of 62)		
PMF	**Ensemble Perf.**	**PIP**	**PMF**	**Ensemble Perf.**	**PIP**	**PMF**	**Ensemble Perf.**	**PIP**	**PMF**	**Ensemble Perf.**	**PIP**
0.00	96.7 ± 0.0	100	0.00	97.1 ± 0.0	100	0.00	97.5 ± 0.0	100	0.0	97.1 ± 0.0	100
2.50	96.8 ± 0.1	100	2.50	97.0 ± 0.1	100	2.50	97.3 ± 0.1	100	2.5	97.1 ± 0.1	100
5.00	96.6 ± 0.1	100	5.00	97.0 ± 0.1	100	5.00	97.2 ± 0.1	100	5.0	96.9 ± 0.1	100
7.50	96.6 ± 0.1	100	7.50	96.9 ± 0.1	100	7.50	96.8 ± 0.2	100	7.5	96.7 ± 0.1	100
10.0	96.4 ± 0.1	100	10.0	96.5 ± 0.1	100	10.0	96.4 ± 0.3	99	10.0	96.3 ± 0.1	100
15.0	95.7 ± 0.3	100	15.0	94.9 ± 0.2	97	15.0	94.1 ± 0.3	87	15.0	94.7 ± 0.2	99
20.0	93.2 ± 0.4	97	20.0	91.8 ± 0.4	83	20.0	91.4 ± 0.3	59	20.0	91.7 ± 0.1	89
25.0	89.6 ± 0.5	86	25.0	88.6 ± 0.5	56	25.0	89.1 ± 0.6	27	25.0	87.7 ± 0.6	67
30.0	85.9 ± 0.5	63	30.0	86.5 ± 1.1	27	30.0	88.7 ± 1.5	9	30.0	84.8 ± 1.2	37

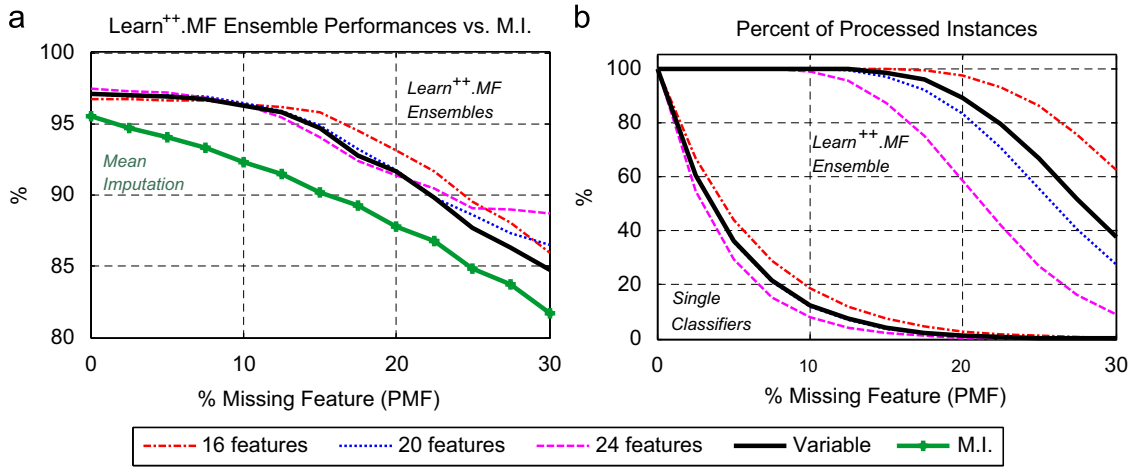

Fig. 8. Analysis of the algorithm on the OCR database with respect to PMF and *nof*: (a) Learn^{++}.MF ensemble performances compared to that of mean imputation and (b) single classifier and ensemble PIP.

Table 5
Performance on the MFEAT database.

nof=10 (out of 216)			*nof*=20 (out of 216)			*nof*=30 (out of 216)		
PMF	**Ensemble Perf**	**PIP**	**PMF**	**Ensemble Perf**	**PIP**	**PMF**	**Ensemble Perf**	**PIP**
0.00	94.90 ± 0.00	100	0.00	95.50 ± 0.00	100	0.00	95.40 ± 0.00	100
5.00	94.82 ± 0.10	100	5.00	95.42 ± 0.12	100	5.00	95.28 ± 0.10	100
10.00	94.82 ± 0.10	100	10.00	95.46 ± 0.16	100	10.00	95.04 ± 0.14	100
15.00	94.82 ± 0.10	100	15.00	95.24 ± 0.19	100	15.00	94.28 ± 0.39	100
20.00	94.68 ± 0.10	100	20.00	95.18 ± 0.32	100	20.00	92.45 ± 0.96	83
25.00	94.52 ± 0.08	100	25.00	93.65 ± 0.30	98	25.00	90.54 ± 1.70	37
30.00	94.36 ± 0.21	100	30.00	90.83 ± 0.57	79	30.00	91.42 ± 3.21	8
nof=40 (out of 216)			*nof*=50 (out of 216)			*nof*=var. (10–50 out of 216)		
PMF	**Ensemble Perf**	**PIP**	**PMF**	**Ensemble Perf**	**PIP**	**PMF**	**Ensemble Perf**	**PIP**
0.00	95.20 ± 0.00	100	0.00	95.20 ± 0.00	100	0.00	95.60 ± 0.00	100
5.00	95.12 ± 0.17	100	5.00	94.86 ± 0.10	100	5.00	95.60 ± 0.06	100
10.00	94.91 ± 0.19	100	10.00	93.38 ± 0.08	95	10.00	95.42 ± 0.19	100
15.00	92.89 ± 0.60	82	15.00	92.06 ± 0.81	40	15.00	95.20 ± 0.17	100
20.00	91.71 ± 0.91	28	20.00	92.56 ± 2.87	4	20.00	94.98 ± 0.15	100
25.00	89.53 ± 4.70	4	25.00	n/a	0	25.00	94.98 ± 0.50	100
30.00	n/a	0	30.00	n/a	0	30.00	93.06 ± 0.53	100

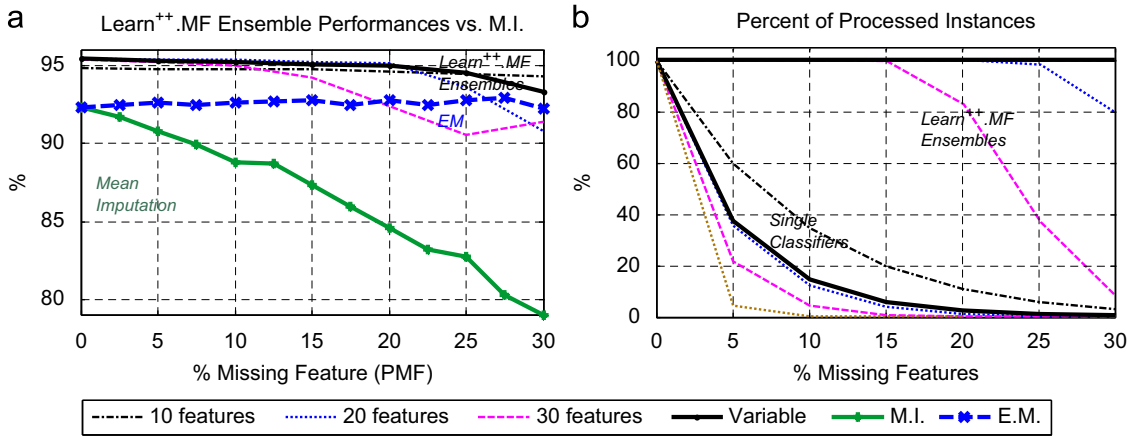

Fig. 9. Analysis of the algorithm on the MFEAT database with respect to PMF and *nof*: (a) Learn^{++}.MF ensemble performances compared to that of mean imputation and (b) single classifier and ensemble PIP.

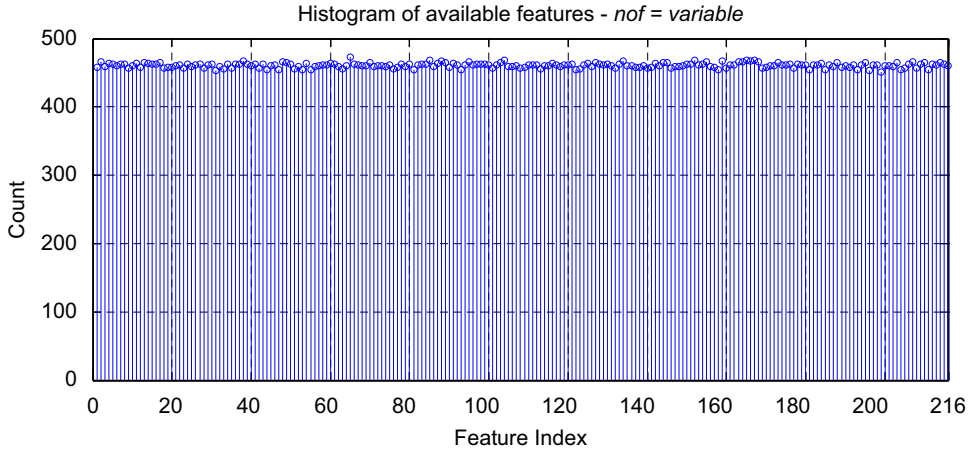

Fig. 10. Histogram of the available features from useable classifiers (*nof*=variable).

Finally, with this many features, a potential concern is whether every feature is adequately represented by the *useable* classifiers. The histogram of the available features in Fig. 10 indicates that the union of the features chosen by algorithm encompasses the entire feature set adequately and uniformly.

Table 6
Best *nof* performances on all databases.

	PMF	0%		5%		10%		15%		20%		25%		30%		TargetPerf.	
Dataset	*nof* ↓	**PIP**	**PGP**	**PIP**	**PGP**	**PIP**	**PGP**	**PIP**	**PGP**	**PIP**	**PGP**	**PIP**	**PGP**	**PIP**	**PGP**	**MI**	% ↓
WINE	**3/13**	*100*	**100**	*100*	**100**	*100*	**99**	*100*	**99**	*100*	**99**	*100*	**99**	*100*	**96**	**86**	**100**
WBC	**10/30**	*100*	**95**	*100*	**95**	*100*	**96**	*100*	**95**	*100*	**95**	*98*	**94**	*92*	**93**	**73**	**97**
OCR	**16/62**	*100*	**97**	*100*	**97**	*100*	**96**	*100*	**96**	*97*	**93**	*86*	**90**	*63*	**86**	**82**	**97**
MFEAT	**10/216**	*100*	**95**	*100*	**95**	*100*	**95**	*100*	**95**	*100*	**95**	*100*	**95**	*100*	**94**	**79**	**96**
WATER (38)	**12/38**	*100*	**77**	*100*	**78**	*100*	**77**	*100*	**76**	*99*	**77**	*95*	**76**	*85*	**75**	**69**	**80**
VOC-I	**3/6**	*100*	**93**	*100*	**92**	*100*	**91**	*100*	**91**	*99*	**90**	*99*	**90**	*99*	**89**	**54**	**92**
ECOLI	**3/5**	*100*	**89**	*100*	**87**	*100*	**86**	*100*	**84**	*100*	**83**	*100*	**80**	*100*	**81**	**65**	**89**
ION	**8/34**	*100*	**95**	*100*	**95**	*100*	**96**	*100*	**96**	*100*	**96**	*100*	**94**	*100*	**94**	**76**	**95**
DERMA (34)	**8/34**	*100*	**98**	*100*	**98**	*100*	**98**	*100*	**97**	*100*	**96**	*100*	**96**	*99*	**93**	**92**	**100**
PEN	**7/16**	*100*	**91**	*100*	**90**	*100*	**90**	*100*	**88**	*98*	**87**	*95*	**85**	*89*	**81**	**51**	**97**
VOC-II (12)	**4/12**	*100*	**96**	*100*	**96**	*100*	**96**	*100*	**96**	*100*	**96**	*100*	**96**	*98*	**97**	**57**	**98**

4.6. Summary results on other datasets

The algorithm was evaluated on several additional real world and benchmark datasets with various number of features, whose full set of results are available online [48]; however, summary results are provided in Table 6. For each dataset in Table 1, Table 6 provides the percent generalization performance (PGP) as well as PIP as obtained by the best overall selection of *nof* for all PMF values in the 0–30% range. Also included in Table 6 is the target performance, which is the best average performance reported for these datasets in the literature when used with the full feature set. Finally, the MI column indicates the performance of mean imputation at 30% PMF, which is always poorer, typically with very wide margins than that of Learn^{++}.MF. Despite these encouraging absolute performance levels, we again note that the objective in these experiments was not to obtain the best possible generalization/classification performance, but rather to illustrate the behavior of the algorithm. Hence, base classifiers were never optimized, yet the algorithm was able to meet target performances by using only a fraction of the available features.

4.7. Comparisons to other approaches

Comparison of Learn^{++}.MF to standard mean imputation was shown in the above results. An intermediate approach would be to combine mean imputation with standard RSM, i.e., training individual classifiers with feature subsets, using all classifiers for final decision with mean imputation used for the missing features. Such an approach is attractive since all classifiers are utilized, including those that would otherwise be discarded by Learn^{++}.MF, and that all instances can be processed. Using the exact same experimental setups described above, such an approach was also implemented, whose results are shown in Fig. 11 for all *nof* values used for each database. We observe that the generalization performances obtained by Learn^{++}.MF were in general higher for the Wine, WBC and OCR datasets (and with statistical significance for certain *nof* values—see [48] for confidence intervals), and comparable for the MFEAT database. Considering that Learn^{++}.MF uses substantially fewer classifiers (as classifiers using missing features are discarded) this outcome is favorable, not only for computational reasons, but also because such an outcome indicates that the algorithm can extract all the useful information from the available classifiers.

Also included in Fig. 11 are the performances obtained by a naïve Bayes classifier on these datasets. The naïve Bayes classifier (NBC) is also attractive, as it can naturally accommodate missing data without any preprocessing (except computing the required class conditional probabilities). The NBC performances, however, paint a mixed picture: the performances are comparable to those of RSM for some datasets, whereas clearly inferior for others; though they were always inferior to those of Learn^{++}.MF. Note that NBC requires features to be class conditionally independent (often not the case), and accurate computation of class conditional probabilities requires a sufficiently dense training dataset (also often not the case), which may in part explain its somewhat inferior performance.

Finally, we also compared the Learn^{++}.MF performances to those obtained by one-class one-feature approach described in [23]. As mentioned earlier, this approach trains one classifier using each feature on each class, and therefore can accommodate any number of missing features with the fewest classifiers possible (number of features times number of classifiers). The classifiers are then combined using a non-trainable combiner, such as mean, product rule, etc. In our implementation, we used the mean combiner along with the same base MLP classifier and architecture as used in the Learn^{++}.MF simulations. The results (averages of 10 trials) are tabulated in Table 7 for each of the four databases discussed above.

Of the four datasets listed in Table 7, WBC, and a smaller version of MFEAT (a subset of the original feature space using only 10 features) were also used in [23]. Our results on these two datasets are comparable to those presented in [23]. Specifically, the authors reported around 10% error on WBC and 20–30% error on the reduced MFEAT dataset (using the mean combiner, their results using other combiners were generally even poorer). The performances of this approach on all datasets fall far short of those of Learn^{++}.MF, except perhaps on WBC where the performances were comparable or better for Learn^{++}.MF up to 20% PMF, and comparable or better for one-class combiners for 25% and 30% PMF. Finally, the authors of [23] also present results for the dermatology dataset, with the mean combiner providing 35–45% error, also far poorer than that of Learn^{++}.MF (compare results at Table 6, or in [48]). We also note that the performances of the one-class combiners do not change too much for different PMF values, an outcome of using a single feature for training the classifiers.

5. Discussions and conclusions

The RSM in general, and Learn^{++}.MF algorithm as a specific implementation of RSM in particular, are presented as alternative solutions to the missing feature problem. Learn^{++}.MF creates an ensemble of classifiers, each trained with a random subset of the features, so that an instance with missing features can still be

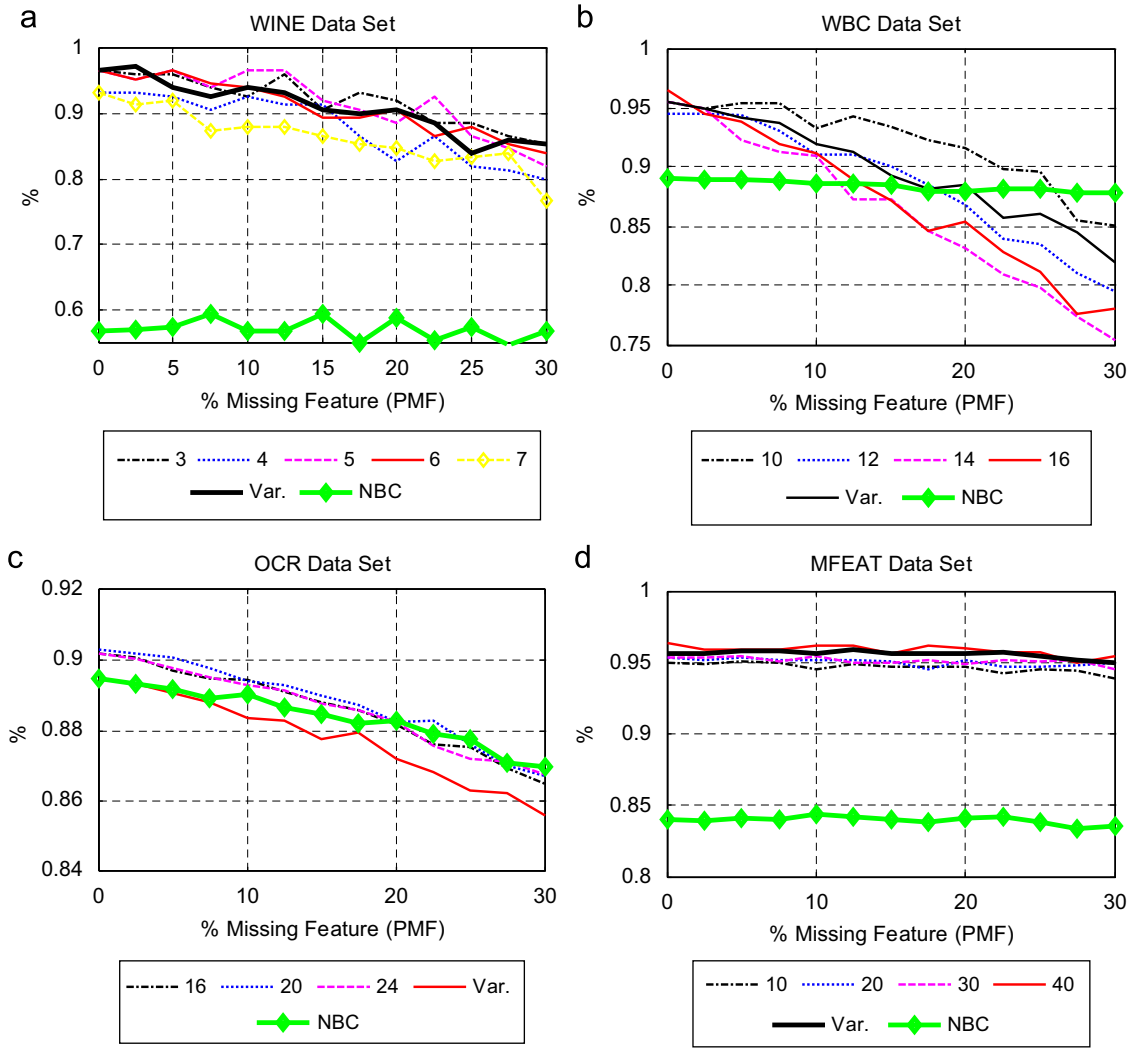

Fig. 11. Performances obtained by (i) using all classifiers trained according to standard RSM, complemented with mean imputation for missing features and (ii) naïve Bayes on the four features datasets.

Table 7
Performances on combining one-class classifiers trained on single features.

PMF	WINE	WBC	OCR	MFEAT
	PERFORMANCE			
0.00	86.67	93.50	24.90	70.33
5.00	88.70	93.75	24.70	70.27
10.00	89.67	93.55	24.70	69.80
15.00	91.00	93.40	24.10	70.60
20.00	88.70	93.50	23.90	70.00
25.00	90.33	92.90	23.80	68.87
30.00	89.33	93.55	23.31	68.67

classified using classifiers whose training data did not include those attributes.

There are two important parameters of the algorithm: *nof*, the number of features used to train individual classifiers, and *T*, the total number of classifiers to be generated. The effect of *T* on the performance is strictly an issue of computational resources. In general, a relatively large number of classifiers should be generated to ensure that a sufficient number of classifiers are available for as many possible combinations of missing features as possible. The choice of *T* directly impacts the PIP (% instances processed) and PUC (% usable classifiers). With more classifiers, more missing feature combinations can be accommodated, and hence more instances can be processed. Note that the computational burden of the approach is not as excessive as it might first appear, however, since individual classifiers are relatively weak, obtained by using small network architectures and high error goal, allowing quick training. Also, using a subset of features further reduces computational burden. In fact, we can argue that – due to the assumed distributed redundancy in feature space – random subset selection is quite efficient: for example, an exhaustive run on training with every possible combination of, say, 24 out of 62 features of the OCR data would have required 9.7×10^{16} classifiers, though the algorithm performed quite well with only 1000. Furthermore, as customary in most batch processing applications, the training is done off-line and once complete, the testing part takes much less time even with thousands of classifiers.

As described earlier, there are algorithms that use far fewer classifiers than Learn^{++}.MF. For example, combining one-class classifiers trained on single feature at a time can handle any combination of missing features using the fewest possible classifiers (number of features times number of classifiers); however, as reported in [23] and verified in our analyses, the generalization performance may suffer due to insufficient distinguishing

ability of single features. Conversely, Learn^{++}.MF does not claim capability of handling all possible combinations of missing features. However, it can typically process – and usually correctly – virtually all or a substantially large portion of the data, provided that a reasonably sufficient number of classifiers are trained, and of course that the main redundancy assumption of the algorithm is met.

A slight variation of Learn^{++}.MF is using the standard random subspace method, generating an ensemble of classifiers on feature subsets, and then employing all classifiers with mean imputation used for the missing features. Such an approach has the advantage of leaving no instances unprocessed, albeit at a cost of some performance drop compared to Learn^{++}.MF. Depending on the dataset, such a cost may be reasonable.

The second free parameter of the algorithm is the number of features (*nof*) used to train individual classifiers. In order to obtain a general rule of thumb on proper selection of this parameter, we have analyzed the impact of this parameter on the overall performance, as well as on the percent of instances (PIP) that can be processed. As described in the results section, using a larger number of features for training typically provides better performance when the PMF is less than 10%. However, as PMF increases, the overall performance and the PIP drop rapidly. Using fewer features for training, on the other hand, provides a more stable performance with a more gradual drop both in performance and PIP for increasing PMF. Not surprisingly, these results suggest that the smallest *nof* should be chosen that provides a satisfactory performance. While the specific value depends on the feature redundancy in the dataset, the number of features we have used was typically in the 15–50% range of the total feature pool. We found that using a variable number of features, selected at random for each classifier in the above mentioned range is a good trade-off, as it usually achieves similar performance with little or no drop in PIP.

In recognition of the no-free-lunch theorem, we acknowledge that any algorithm is effective only to the extent its characteristics match those of the data. In the case of well-established techniques, such as Bayesian estimation and expectation maximization, this translates into restrictions on dimensionality, prior knowledge of underlying distributions, and/or availability of sufficiently dense training data. For Learn^{++}.MF, which tries to make the most of the available data instead of trying to estimate the missing values, two restrictions apply: first, the dataset must include redundant features (the number and identities of which are unknown to us, since they would not have been part of the data otherwise). Second, the redundancy in the features must be distributed randomly over the feature set. Therefore, time series such as raw electrocardiogram (ECG) cannot be used with this approach, though, specific features extracted from the ECG, such as maximum amplitude, area under the QRS complex, rise time, etc. can be used.

Fortunately, applications that meet these criteria are abundant in real world. For example, applications that use a set of different sensors (e.g., temperature, speed, force, acceleration, humidity, load, temporal parameters, etc.) to monitor a physical condition, typically meet these conditions. In such cases, the algorithm would be particularly useful when one or more of the sensors malfunction.

We should add that, as with most ensemble based approaches, Learn^{++}.MF is independent of the base (component) classifier used to build the ensemble, allowing the flexibility to use the model (classifier type) identified to be most appropriate for the particular application.

Finally, we should reemphasize that we assumed the features are missing completely at random (MCAR), and that the probability of all features being missing is the same. In certain applications, this may not be the case: for example, in a medical diagnosis problem, certain test results (features) corresponding to particularly expensive or risky procedures may be missing more often than simpler and less expensive tests. If the probability of any given feature being missing is known ahead of time, the proposed general framework can still be used, however, alternative feature subspaces should be explored that take this knowledge into consideration. Our future work will explore this direction.

Acknowledgements

This material is based upon the work supported by the National Science Foundation under Grant nos. ECCS-0239090 and ECCS 0926159. The authors acknowledge S. Krause for his contributions during the earliest stages of this work. The authors also wish to acknowledge the anonymous reviewers, as this paper has benefited greatly from addressing their comments and suggestions.

References

[1] H..Schöner, Working with real-world datasets: preprocessing and prediction with large incomplete and heterogeneous datasets, Ph.D. Dissertation, Technical University of Berlin, 2004.
[2] J.J.A. Little, D.B. Rubin, in: Statistical Analysis with Missing Data, second ed., Wiley, New York, 2002.
[3] K.L..Wagstaff, V.G..Laidler, Making the most of missing values: object clustering with partial data in astronomy, Astronomical Data Analysis Software and Systems XIV, ASP Conference Series, vol. 30, 2005, P2.1.25.
[4] R.L. Morin, D.E. Raeside, A reappraisal of distance-weighted K-nearest neighbor classification for pattern recognition with missing data, IEEE Transactions on Systems, Man and Cybernetics 11 (1981) 241–243.
[5] A. Farhangfar, L. Kurgan, J. Dy, Impact of imputation of missing values on classification error for discrete data, Pattern Recognition 41 (2008) 3692–3705.
[6] D. Howell, The treatment of missing data, in The SAGE Handbook of Social Science Methodology, in: William, Stephen P. Turner (Eds.), OuthwaiteSage, London, UK, 2007, pp. 208–224.
[7] Y. Qin, S. Zhang, Empirical likelihood confidence intervals for differences between two datasets with missing data, Pattern Recognition Letters 29 (2008) 803–812.
[8] R.J.A. Little, Consistent regression methods for discriminant analysis with incomplete data, Journal of the American Statistical Association 73 (1978) 319–322.
[9] A.C. Morris, M.P. Cooke, P.D. Green, Some solution to the missing feature problem in data classification, with application to noise robust ASR, IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 98) 2 (1998) 737–740.
[10] V. Tresp, R. Neuneier, S. Ahmad, Efficient methods for dealing with missing data in supervised learning, Neural Information Processing Systems 7 (1995) 689–696.
[11] M. Ramoni, P. Sebastiani, Robust learning with missing data, Machine Learning V45 (2001) 147–170.
[12] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society 39 (1977) 1–38.
[13] M.I. Jordan, R.A. Jacobs, Hierarchical mixtures of experts and the EM algorithm, Neural Computation 6 (1994) 181–214.
[14] G.J. Mclachlan, T. Krishnan, in: The EM Algorithm and ExtensionsWiley, New York, NY, 1992.
[15] W. David, et al., On classification with incomplete data, IEEE Transactions on Pattern Analysis and Machine Intelligence 29 (2007) 427–436.
[16] M. Di Zio, U. Guarnera, O. Luzi, Imputation through finite Gaussian mixture models, Computational Statistics and Data Analysis 51 (2007) 5305–5316.
[17] A. Gupta, M. Lam, The weight decay backpropagation for generalizations with missing values, Annals of Operations Research 78 (1998) 165–187.
[18] S.Y. Yoon, S.Y. Lee, Training algorithm with incomplete data for feed-forward neural networks, Neural Processing Letters 10 (1999) 171–179.
[19] R. Nowicki, Rough neuro-fuzzy structures for classification with missing data, IEEE Transactions on Systems, Man, and Cybernetics, Part B 39 (2009) 1334–1347.
[20] G. Bogdan, Neuro-fuzzy approach to processing inputs with missing values in pattern recognition problems, International Journal of Approximate Reasoning 30 (2002) 149–179.
[21] C.-P. Lim, J.-H. Leong, M.-M. Kuan, A hybrid neural network system for pattern classification tasks with missing features, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (2005) 648–653.

[22] P. Melville, et al., Experiments on ensembles with missing and noisy data, International Workshop on Multiple Classifier Systems (MCS 2004), Lecture Notes in Computer Science 3077 (2004) 293–302.
[23] P. Juszczak, R.P.W. Duin, Combining one-class classifiers to classify missing data, In: Proceedings of the International Workshop on Multiple Classifier Systems (MCS 2004) 3077 (2004) 92–101.
[24] M. Aksela, J. Laaksonen, Using diversity of errors for selecting members of a committee classifier, Pattern Recognition 39 (2006) 608–623.
[25] R.E. Banfield, et al., Ensemble diversity measures and their application to thinning, Information Fusion 6 (2005) 49–62.
[26] G. Brown, et al., Diversity creation methods: a survey and categorisation, Information Fusion 6 (2005) 5–20.
[27] S.T. Hadjitodorov, L.I. Kuncheva, L.P. Todorova, Moderate diversity for better cluster ensembles, Information Fusion 7 (2006) 264–275.
[28] L.K. Hansen, P. Salamon, Neural network ensembles, IEEE Transactions on Pattern Analysis and Machine Intelligence 12 (1990) 993–1001.
[29] R.E. Schapire, The strength of weak learnability, Machine Learning 5 (1990) 197–227.
[30] D.H. Wolpert, Stacked generalization, Neural Networks 5 (1992) 241–259.
[31] T.K. Ho, J.J. Hull, S.N. Srihari, Decision combination in multiple classifier systems, IEEE Transactions on Pattern Analysis and Machine Intelligence 16 (1994) 66–75.
[32] L. Breiman, Bagging predictors, Machine Learning 24 (1996) 123–140.
[33] Y. Freund, R.E. Schapire, Decision-theoretic generalization of on-line learning and an application to boosting, Journal of Computer and System Sciences 55 (1997) 119–139.
[34] L.I. Kuncheva, Combining pattern classifiers, Methods and AlgorithmsWiley Interscience, New York, NY, 2005.
[35] R. Polikar, Ensemble based systems in decision making, IEEE Circuits and Systems Magazine 6 (2006) 21–45.
[36] R. Polikar, Bootstrap—inspired techniques in computational intelligence, IEEE Signal Processing Magazine 24 (2007) 59–72.
[37] T.K. Ho, Random subspace method for constructing decision forests, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (1998) 832–844.
[38] M. Skurichina, R. Duin, Combining Feature Subsets in Feature Selection (2005) 165–175.
[39] M. Skurichina, R. Duin, Bagging and the random subspace method for redundant feature spaces, multiple classifier systems (MCS 2001), Lecture Notes in Computer Science 2096 (2001) 1–10.
[40] N. Rooney, others, Technical report: random subspacing for regression ensembles, 2004.
[41] A. Tsymbal, M. Pechenizkiy, P. Cunningham, Diversity in search strategies for ensemble feature selection, Information Fusion 6 (2005) 83–98.
[42] R. Polikar, et al., Learn^{++}: an incremental learning algorithm for supervised neural networks, IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews 31 (2001) 497–508.
[43] M.D. Muhlbaier, A. Topalis, R. Polikar, Learn^{++}.NC: combining ensemble of classifiers with dynamically weighted consult-and-vote for efficient incremental learning of new classes, IEEE Transactions on Neural Networks 20 (2009) 152–168.
[44] P.K. Sharpe, R.J. Solly, Dealing with missing values in neural network-based diagnostic system, Neural Computing and Applications 3 (1995) 73–77.
[45] J. DePasquale, R. Polikar, Random feature subset selection for ensemble based classification of data with missing features, in: Proceedings of the 7th International workshop on Multiple Classifier Systems, Lecture Notes in Computer Science 4472 (2007) 251–260.
[46] A. Asuncion, D.J. Newman, in: UCI Repository of machine learning database at Irvine CA, University of California, School of Information and Computer Science, Irvine, available online: < http://archive.ics.uci.edu/ml/index.html >, 2007.
[47] T. Schneider, Analysis of incomplete climate data: estimation of mean values and covariance matrices and imputation of missing values, Journal of Climate 14 (2001) 853–871.
[48] R. Polikar, J. Depasquale, Full Learn^{++}.MF results on additional datasets, Electrical and Computer Engineering, Rowan University, available online: < http://users.rowan.edu/~polikar/RESEARCH/Learn++.MF.html >, 2009.

Dr. Robi Polikar is an Associate Professor of Electrical and Computer Engineering at Rowan University, in Glassboro, New Jersey where he leads the Signal Processing and Pattern Recognition Laboratory. He has received his co-major Ph.D. degree in Electrical Engineering and Biomedical Engineering, from Iowa State University, Ames, Iowa in 2000. His current research interests within computational intelligence include ensemble systems, incremental and nonstationary learning, and various applications of pattern recognition in biomedical engineering, such as brain-computer interface and early diagnosis of Alzheimer's disease based on EEG analysis. He is a senior member of IEEE, and member of ASEE, Tau Beta Pi and Eta Kappa Nu. His current and recent work has been funded primarily by NSF and NIH.

Joseph DePasquale is a Master of Science student at Rowan University. His areas of interest include applications of ensemble systems to various areas including bioinformatics.

Hussein Syed-Mohamed has recently completed his Master of Science degree at Rowan University and is currently a System Engineer at Honeywell.

Dr. Gavin Brown is a Lecturer at the University of Manchester, UK, in the Machine Learning and Optimization Group. He received his Ph.D. from the University of Birmingham in 2004, on the topic of neural network ensembles, for which he also received the Distinguished Dissertation Award from the British Computer Society. His current areas of interest are feature selection and extraction with information theoretic methods, Markov Blanket algorithms, and online learning with particular application to systems biology and adaptive compiler optimization.

Dr. Ludmilla Kuncheva received her M.Sc. degree from the Technical University of Sofia, Bulgaria, in 1982, and her Ph.D. degree from the Bulgarian Academy of Sciences in 1987. Until 1997 she worked at the Central Laboratory of Biomedical Engineering at the Bulgarian Academy of Sciences. Dr. Kuncheva is currently a Reader at the School of Computer Science, Bangor University, UK. Her interests include pattern recognition and classification, machine learning, classifier combination and fMRI data analysis. She has published two books and above 150 scientific papers.