Proceedings of International Joint Conference on Neural Networks, Orlando, Florida, USA, August 12-17, 2007

Random Feature Subset Selection for Analysis of Data with Missing Features

Joseph DePasquale, *Student Member IEEE* and Robi Polikar, *Member IEEE*

Abstract — **We discuss an ensemble-of-classifiers based algorithm for the missing feature problem. The proposed approach is inspired in part by the random subspace method, and in part by the incremental learning algorithm, Learn^{++}. The premise is to generate an adequately large number of classifiers, each trained on a different and random combination of features, drawn from an iteratively updated distribution. To classify an instance with missing features, only those classifiers whose training data did not include the currently missing feature are used. These classifiers are combined by using a majority voting combination rule to obtain the final classification of the given instance. We had previously presented preliminary results on a similar approach, which could handle up to 10% missing data. In this study, we expand our work to include different types of rules to update the distribution, and also examine the effect of the algorithm's primary free parameter (the number of features used to train the ensemble of classifiers) on the overall classification performance. We show that this algorithm can now accommodate up to 30% of features missing without a significant drop in performance.**

I. INTRODUCTION

One of the most commonly encountered problems in the real-world implementation of automated decision making systems is the issue of missing data. Bad sensors, corrupted pixels, malfunctioning equipment, human error in data collection, and data corruption are all common scenarios in real-world applications, resulting in partial-loss of data during acquisition. Sometimes such loss of data is merely a nuisance, when, for example, a few data points here and there may be missing. Such cases are often addressed by ignoring the data points with missing components. In other cases, however, the nature of data loss may render the subsequent data analysis impossible. Automated classification applications are prime examples of such cases, particularly if the classifier (or model used to analyze the data) is not designed to handle data with missing features. An extreme, but not unlikely scenario is when all instances in a given database are missing a single (or more) feature. In such cases, instances with missing features cannot be ignored, as this would amount to discarding the entire data. For most commonly used classifiers, including feed forward neural networks this is a potentially fatal weakness.

Extensive research has already been performed to develop robust approaches to handle the missing data problem. Data imputation, such as replacing the missing value with the average of its *k*-nearest neighbors is a commonly used practical approach that can provide meaningful estimates [1,2]. However, this and similar imputation techniques require training data to be sufficiently dense, so that a true representation of the missing value can be obtained. This requirement is rarely satisfied in real-world applications even for problems with small dimensionality. There are, of course, several theoretically rigorous approaches that provide the best estimate of the missing value under certain conditions. These techniques are often probabilistic in nature, and they require some prior knowledge of the data distributions and/or a sufficiently dense training dataset. These techniques are in fact optimal if prior distribution information is available; however, they can lead to grossly inaccurate estimates of the missing value if the distribution information is not available. Then, the distribution itself must be estimated from the existing training data, which in turn requires that the training dataset is sufficiently dense. Techniques based on Bayesian estimation or expectation maximization are well established examples of such approaches [3,4].

An alternative, and perhaps a more intuitive approach for dealing with missing features is provided by neuro-fuzzy algorithms, where missing data values are either estimated, or the classification is done based on the fuzzy membership of the data point with respect to its nearest neighbors, clusters, or hyperboxes. The parameters of the clusters and hyperboxes are determined from the existing data points. Algorithms based on general fuzzy min-max neural networks [5], ARTMAP or fuzzy c-means clustering [6] are examples of this approach.

More recently, ensemble-based classification algorithms have also been proposed for the missing feature problem. In [7], Juszczack and Duin describe using an ensemble of one-class classifiers, each trained on a single feature. Such an ensemble is capable of handling any combination of missing features in the data using the least number of classifiers. This approach has been found to be quite effective as long as each feature used to train individual classifiers is reasonably sufficient to approximate the necessary decision boundary. Melville et al. propose a different ensemble based approach through their algorithm DECORATE, which creates artificial data with no missing features from the original data that has missing features [8].

Manuscript received January 31, 2007. This material is based upon work supported by the National Science Foundation under Grant No ECS-0239090.

Joseph DePasquale (depasq63@students.rowan.edu) and Robi Polikar (polikar@rowan.edu) are with the Signal Processing and Pattern Recognition Laboratory, Electrical and Computer Engineering, Rowan University, 201 Mullica Hill Rd, Glassboro, NJ 08028 USA.

Contact author: R. Polikar, phone: (856) 256-5372; fax: (856) 256-5241.

The algorithm proposed here, called Learn^{++}.MF, is an alternative approach based on generating an ensemble of classifiers, each trained on a random subset of the available features. An instance **x** with missing features is then classified by majority vote decision of those classifiers whose training data did not include the particular features missing in **x**. In this approach, the subset of features used to train each classifier is drawn from an iteratively updated distribution, which ensures that the combinations of features that have not previously been used have a higher likelihood of being selected. As such, the algorithm is inspired in part by the feature subset selection ideas introduced by the Ho's random subspace method (RSM) [9], and in part by the distribution update rule mechanism used in AdaBoost [10] and Learn^{++} type algorithms [11]; hence the name Learn^{++}.MF for missing feature problems.

In our preliminary work, we described a similar approach using a specific distribution update rule, and showed that the proposed approach could handle the missing data that comprised as much as 10% of the entire data size [12]. In this contribution, we extend our work by comparing different feature distribution update rules, and analyzing the effect of primary free parameter of the algorithm on classification performance. We show that by judiciously selecting this parameter – the number (or percentage) of features to be used in the feature subsets – we can now handle as much as 30% missing data.

II. LEARN^{++}.MF

An intuitive ensemble based approach would be to train one or more classifiers for every possible combination of features subsets. Such an approach, perfectly suitable for datasets with few features, has in fact been previously proposed [13]. As the number of features increases, however, training classifiers with such an exhaustive selection of feature sets becomes computationally prohibitive: for a database with *n* features, the total number of feature subsets is

$$K = \sum_{k=1}^{n} \frac{n!}{(n-k)!k!} = 2^n - 1 \qquad (1)$$

On the other hand, while the number of feature subsets increase exponentially with the total number of features, the probability of any given feature subset being missing also decreases exponentially. Hence, trying to accommodate every possible combination of features is not only prohibitively expensive, but it is also unnecessary (and is an inefficient use of computer resources). If we have any prior information on the expected ratio of features missing (a very rough approximation is usually good enough), a random selection of a relatively small number of combinations can be used to address most combinations of missing features. Learn^{++}.MF follows such an approach: it trains an ensemble of classifiers with a random subset of the features, where the number of features used to train each classifier is a free parameter of the algorithm. The global number of features available for training is defined as *f*. We define *nof* < *f*, as the number of features used to train each classifier in the ensemble. Learn^{++}.MF also employs an iterative distribution update rule so that the feature combinations not previously accounted for are more likely to be selected for future classifiers. We show that Learn++.MF can then classify the data with missing features with little or no performance loss, compared to classifying fully intact data, even when large portions of data may be missing.

The algorithm is described in detail below, and its pseudocode is provided in Figure 1.

Inputs:

- Sentinel value, *sen*, denoting missing features.
- A supervised algorithm, BaseClassifier
- The number of classifiers to be generated, T
- Training data, $S = \{(x_i, y_i) \mid i = 1, \ldots, N\}$.
- Number of features used to train each classifier, *nof*.

Training

Initialize $D_1(j) = 1/f, \forall j, j = 1, \cdots, f$

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

1. Normalize the distribution D_t
2. Draw *nof* features from distribution D_t
3. If S includes instances with missing features, choose those instances that are *not* missing the features selected in step 2.
4. Train classifier C_t with **BaseClassifier** using the training data selected in step 3.
5. Update the distribution D_t

End Loop

Validation / Testing

Given test data $Z = \{z_i\}, \quad i = 1, \ldots, I$

Do for $i = 1, \ldots, I$

1. Determine the missing features in z_i which have previously been flagged using the sentinel *sen*. Let $M(i)$ be the array holding indices of the missing features.
2. Choose classifiers whose training data did not include the missing features indicated in $M(i)$.
3. Combine the classifiers chosen in step 2 using simple majority voting.
4. Decide on the class that receives the highest vote from the ensemble.

End Loop

Fig. 1. The pseudocode of algorithm Learn^{++}.MF

The inputs to Learn^{++}.MF are the training data S, a supervised classification algorithm, BaseClassifier, the number of features, *nof*, used to train each classifier, the number of classifiers, T, to be generated, and a sentinel value, *sen*, a placeholder for missing features. The numerical value of *sen* should be kept well outside the range of data values being processed. A distribution D, which will be used to determine feature subsets, is initialized to be uniform. Such initialization ensures that all features are equally likely to be drawn into the first feature subset. The distribution is then iteratively updated. During the t^{th} iteration, Learn^{++}.MF draws a random bootstrap sample of *nof* features from the distribution D_t. Let $F_{selection}(t)$ represent those features selected to train the tth classifier C_t. This allows us to keep track of which features have previously been used on which classifiers. If the training dataset is complete, then the entire training dataset is used to train classifier C_t, but only using the selected subset of the features. If the training dataset includes missing features as well, then those instances that are not missing the features in $F_{selection}(t)$ are used to train C_t.

The distribution is then updated such that the features that have been used to train the current classifier are less likely to be selected again in the following iteration. Equation (2) represents this distribution update. The value of β is a parameter which we vary to investigate the effect of different distribution update rules, and determine whether a particular selection significantly affects the overall classification performance. In actual real-world use, the distribution update would normally be a pre-determined expression. In this study β was set to three different database specific values: (i) $\beta = 1/f$, (ii) $\beta = nof/f$, and $\beta = 1/nof$. In our preliminary work, only the first β value was used.

$$D_{t+1}(F_{selection}(t)) = \beta * D_t(F_{selection}(t)) \qquad (2)$$

We have also tried a fourth mechanism, where no distribution update rule was used, by drawing each feature subset from a uniform distribution.

During field use and/or before validation of the algorithm, the field / test data is first scanned for missing or corrupt values, which are replaced by the sentinel value *sen*. For any given test instance z_i the algorithm first determines which features, if any, of z_i are missing by looking for the sentinel values in z_i. The indices of the missing features (if any) are then stored in an array $M(i)$.

$$M(i) = \{\arg(z_i(j) == sen\}, \quad \forall j, j = 1, \cdots, f \qquad (3)$$

Comparing this set of (missing) features to each of the $F_{selection}(t)$, Learn^{++}.MF determines which classifiers C_t were trained on feature set that did not include the features in $M(i)$. The decisions of those classifiers are then combined by majority voting to determine the final classification of z_i.

$$\varepsilon(z_i) = \arg\max_{y \in Y} \sum_{t:C_t(z_i)=y} [|M(i) \bigcap F_{selection}(t) = \varnothing|] \qquad (4)$$

where $[|\bullet|]$ evaluates to 1, if the predicate holds true, and zero otherwise; and $\varepsilon(z_i)$ is the ensemble decision of z_i.

While the algorithm has several free parameters, the one that matters the most is the number of features (*nof*) used to train the individual classifiers. This is because a judicious selection of this parameter can not only affect the final classification performance, but also the portion of the data that can be analyzed by the algorithm. The number of classifiers to be trained, T, has relatively less impact, as long as a sufficiently large number of classifiers are generated. The results obtained by evaluating this algorithm on two benchmark and one real-world dataset are provided below, along with the analysis of the impact of the *nof* parameter.

III. Results

We present the results of the implementation of the algorithm on two datasets obtained from the UCI repository [14], and one real-world database. The benchmark datasets were the 16-feature, 10-class Pen-Digits database for recognition of hand written numerical characters, and the 34-feature, 2-class Ionosphere database for classifying radar returns from the ionosphere. The real world application was 12-feature and 12-class gas identification database for identifying one of 12 volatile organic compounds (VOCs) from responses of quartz crystal microbalance gas sensors.

For each database, missing values were simulated by randomly removing the entries from the data matrix. The percentage of missing features (PMF) was varied from 0% (no data missing) up to 30% in steps of 2.5%. Ten independent trials were performed for each experiment, where the training and data sets were randomly shuffled. All results are averages of such 10 independent trials.

Table 1 shows the four *nof* values, and the number of classifiers T used for each experiment. Total number of features in each dataset is also indicated in parenthesis in the first column. As we discuss below, the range of *nof* values chosen for these experiments roughly represents 25-50% of the total number of features. Note that the value of *nof* is inversely proportional to the number of missing features that can be accommodated by the algorithm. Specifically the algorithm can handle at most *f-nof* missing features for any test instance. For example, if a classifier was trained using 3 out of 12 features, the algorithm can then handle any combination of features missing within the range 1 ~ 9. This is the manner in which the algorithm is able to avoid using a very large number of classifiers.

TABLE 1:
NUMBER OF FEATURES (*nof*) AND ENSEMBLE SIZE (T) USED FOR EACH DATASET

Dataset (f)	nof_1	nof_2	nof_3	nof_4	T
VOC (12)	3	4	5	6	200
PEN (16)	6	7	8	9	250
ION (34)	8	10	12	14	1000

A. VOC Database

This database consists of the responses of 12 quartz crystal microbalance sensors to 12 different volatile organic compounds (VOC), including toluene, xylene, hexane, octane, methanol, and trichloroethylene, among others. Figure 2 summarizes the performance of the algorithm for various values of *nof* (indicated with different line styles), as well as four different distribution update rules (one for each row of plots).

The plots on the left indicate the generalization performance of the algorithm with respect to *p*ercentage of *m*issing *f*eatures (PMF) in the 0 to 30% range.

As discussed in the introduction, Learn^{++}.MF does not guarantee that all possible combinations of features can be accommodated as missing data. Since features are selected at random, and since an exhaustive search is not used, there may be certain feature combinations not represented by any of the classifiers trained in the ensemble. Instances with those exact feature combinations cannot be classified by any of the classifiers in the ensemble, and hence cannot be processed. The plots on the right show the average percentage of instances that can be processed by the algorithm. Two families of curves are given in each plot: those on the center of the plot show the average *P*ercentage of *I*nstances *P*rocessed (PIP) that can be achieved by a single classifier. The family of curves in the upper portion of these plots show the average PIP achieved by the Learn^{++}.MF ensemble.

Several interesting observations can be made by analyzing different families of curves in Figure 2. The first and foremost observation is that the algorithm's generalization performance exhibits very little or no performance drop for even large PMF values. A decline in the performance is certainly expected as the percent of missing features increase, and such a decline is seen for certain *nof* values, however the decline is very minor. In the worst scenario obtained for *nof*=3, the overall performance decline is merely 4% with as much as 30% of the data being missing. Second, the performances appear to be slightly higher for larger *nof* values. This makes sense, as the more features are used, the more information is provided to the classifier (assuming that all features carry relevant information – which they do in this application). Third, while the distribution update rule with β = *nof/f* performs better than other update rules, the differences in the performances are not statistically significant.

Based on the performance plots (on the left), one may be inclined to think that the higher the *nof* the better the performance, and hence higher *nof* values should be used whenever possible. Analyzing the PIP plots however, draws a completely different picture: the smaller the *nof* the larger the amount of data that can be processed.

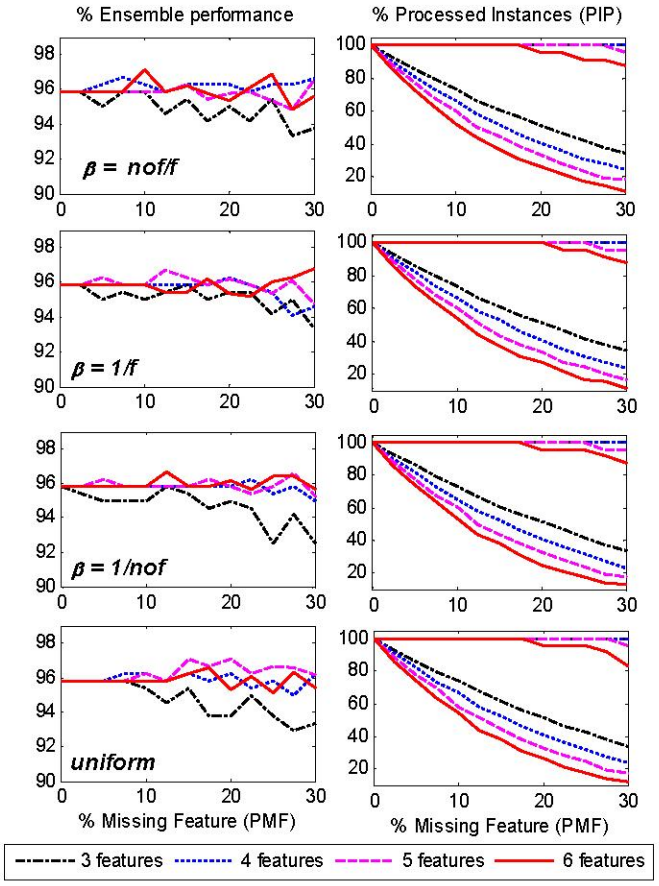

Fig 2. Learn^{++}.MF performance on VOC database

This makes sense: if we use a large number of features to train each classifier (say 10 out of 12), then we need the same large number of features to be available (10 in this case) in the test instance. Then, we can accommodate fewer number of missing features (at most 2, in this example) for any given instance. Hence a larger number of feature combinations will be left unprocessed by the classifier. Conversely, if we use fewer number of features for training the classifiers (say, 3 out of 12), then fewer number of features will be necessary to classify any given instance. Hence, a larger number of missing features (up to 9), and therefore, a larger number of combinations of missing features can be accommodated by the algorithm.

The plots also indicate that using an ensemble approach helps regardless what *nof* is used. Note that for any of the four distribution update rules, the average PIP drops rapidly to around 40% when PMF reaches 30%, if we use a single classifier to classify data with missing features. However, when we use an ensemble of classifiers, the PIP is still 100% for *nof*=3, and about 80-90% for *nof*=6.

B. *Pen-Digits Database*

The Pen-Digits database consists of handwritten numerical values zero through nine. Each of the handwritten characters is digitized to obtain sixteen attributes. Fig 3 shows the generalization performance of the algorithm (on the left) and the PIP values (on the right) for four different *nof* values and four different distribution update mechanisms. As in the previous set of experiments, and as expected, there is a slight and steady decline in the performance as PMF increases. The generalization performance still reaches 85% when the PMF at 30%, indicating the ability of the algorithm to handle missing data. Performance wise, there is little to no difference among different update rules (though, *nof*/*f* and 1/*f* being the better ones), as well as among different *nof* values.

On the other hand, there is substantial difference in the PIP values for different selection of *nof*. As in the VOC database, a higher PIP can be obtained when fewer features (6 out of 16) are used, compared to that when larger number of features are used (9 out of 16). Also, as in the previous case, using an ensemble substantially increases the amount of instances that can be processed, as this adds considerable redundancy and robustness to the algorithm. Using Learn^{++}.MF with *nof*=6, 98% of instances could be processed when 30% of the features were missing, whereas for *nof*=9, PIP drops to about 60%. If a single classifier were used instead of Learn^{++}.MF, PIP then drops to 15% for *nof*=6 and to a practically useless 5% when *nof* = 9.

C. *Ionosphere Database*

The 34-feature ionosphere database is a two class problem for classifying radar returns as "good" or "bad." If the radar signal bounces of the ionosphere and returns, it is defined as "good," if it passes through the ionosphere it is considered "bad." Fig 4 shows the similar generalization performance and PIP plots for this database. Similar trends are exhibited here as in the previous two applications, except that using fewer *nof* provides not only higher PIP (expected), but also higher performance (not usually expected, perhaps due to this database including irrelevant features).

In general, the *nof*/*f* update rule performed better (though the difference from 1/*f* was not statistically significant), fewer *nof* values provided better PIP (100% even for 30% PMF), and of course an ensemble combination provides dramatically more resistance to potential combinations of features that cannot be processed.

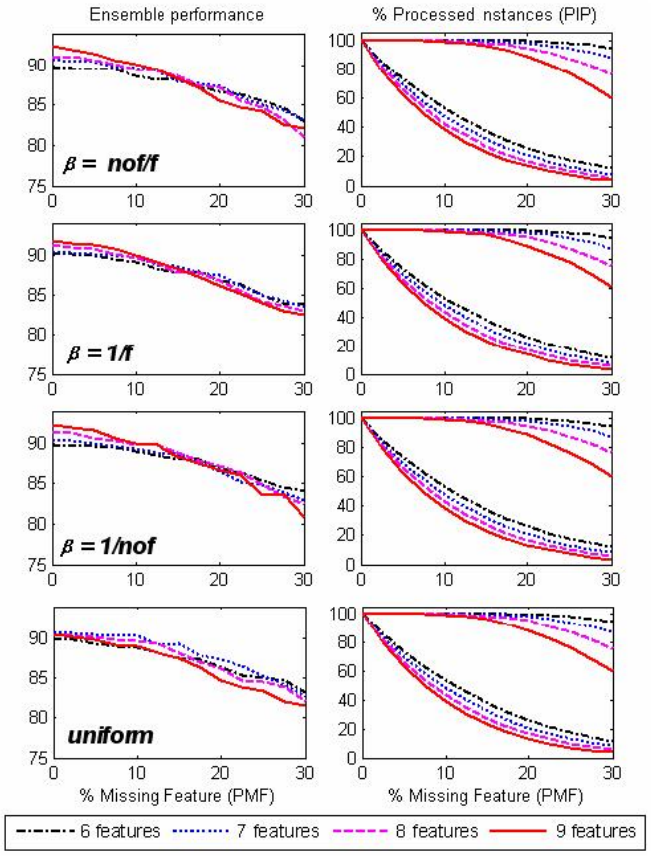

Fig 3. Learn^{++}.MF performance on PEN database

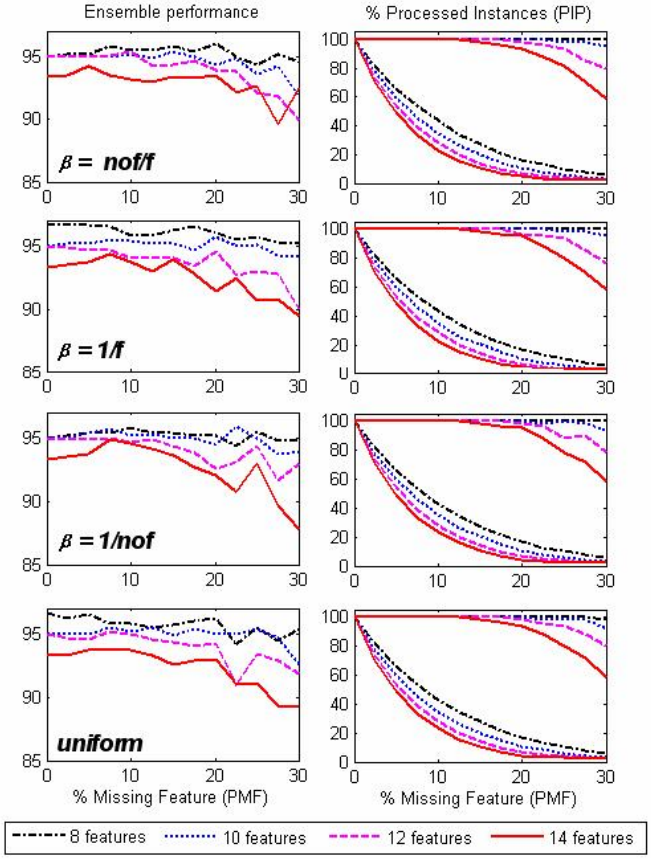

Fig 4. Learn^{++}.MF performance on ION database

IV. Conclusions

In this paper we described an ensemble based algorithm for analyzing data with missing features. The algorithm combines the random subset selection mechanism of the random subspace method, with the distribution update rule of the incremental learning algorithm Learn++. In essence, an ensemble of classifiers is generated, where each classifier is trained on a different subset of the available features. An instance **x** with missing features can then be classified by the majority voting of those classifiers whose training data did not include the features missing in **x**. We observe that this algorithm works rather well, in analyzing a dataset with as much as 30% missing data. The algorithm may be able to handle a higher PMF, depending on the dataset and the *nof* value chosen, however, this is yet to be tested.

We found out that using different distribution update rules has relatively little effect in the final generalization performance of the algorithm. While the *nof/f* performed consistently better, and the uniform distribution performed consistently poorer, the differences were rarely significant.

While the distribution update rule may not have much effect in the algorithm behavior, the value of *nof* does. In general, using a larger *nof* will yield better performances, if all features carry relevant information, since the classifier will be trained with more information carrying features. However, using fewer *nof* for training the individual classifiers allows a larger portion of the data to be processed by the ensemble.

We have not studied the specific effect of the ensemble size (the parameter *T*). In general, however, the larger the *T*, the more feature combinations can be processed. While the number of classifiers required for obtaining good overall performances may seem high in absolute terms (200 – 1000 for the applications presented in this paper), it is in fact fairly low in relative terms, compared to the number of classifiers that would have been necessary to be able to handle every possible missing feature combination. Furthermore, since each classifier is trained on a smaller dimensionality dataset, the training is often fairly fast.

Finally, we should also mention that the algorithm makes one implicit assumption: there are in fact a redundant number of features, where the redundancy is preferably distributed randomly. Of course, the identity of those redundant features are unknown to us, as otherwise they would have already been removed from the data. We note that redundancy requirement of the algorithm is not an overly restrictive one, since many applications of practical interest generate redundant data. For such applications, Learn^{++}.MF may prove to be very beneficial.

Our future work includes evaluating the algorithm on datasets with even larger feature sizes, and establishing theoretical or empirical bounds on the required ensemble size.

References

[1] K.L. Wagstaff, V.G. Laidler, "Making the most of missing values: object clustering with partial data in astronomy," 14th Astronomical Data Analysis and Systems Conf., P. L. Shopbell, M. C. Britton, and R. Ebert, Eds., vol. 30, 2005, pp. 2.1.25.

[2] R. L. Morin, D. E. Raeside, "A reappraisal of distance-weighted k-nearest neighbor classification for pattern recognition with missing data," *IEEE Trans. Systems, Man and Cybernetics*, vol. 11, pp. 241-243, 1981.

[3] A. Dempster, N. Laird, D.R. Rubin, "Maximum-likelihood from incomplete data via the EM algorithm (with discussion)," *Journal of the Royal Statistical Society*, Series B, pp. 1-38, 1997.

[4] V. Tresp, R. Neuneier, S. Ahmad, "Efficient methods for dealing with missing data in supervised learning," G. Tesauro, et al.(eds), Advances in Neural Information Processing Systems, vol. 7. MIT Press, 1995.

[5] B. Gabrys, "Neuro-Fuzzy Approach to Processing Inputs with Missing Values in Pattern Recognition Problems," *International Journal of Approximate Reasoning*, vol. 30, pp. 149-179, 2002.

[6] C. Lim, J. Leong, M. Kuan, "A Hybrid Neural Network System for Pattern Classification Tasks with Missing Features," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 648-653, 2005.

[7] P. Juszczak and R.P.W. Duin, "Combining One-Class Classifiers to Classify Missing Data," Multiple Classifier Systems MCS 2004, *Lecture Notes in Computer Science*, vol. 3077, pp. 92-101, 2004.

[8] P. Melville, N. Shah, L. Mihalkova, and R. Mooney, "Experiments on ensembles with missing and noisy data," Multiple Classifier Systems, *Lecture Notes in Computer Science,* vol. 3077, pp. 293-302, 2004.

[9] T.K. Ho, "The Random Subspace Method for Constructing Decision Forests, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 832-844, 1988.

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

[11] R. Polikar, L. Udpa, S. Udpa, V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Trans. Systems, Man and Cybernetics*, vol. 31, pp. 497-508, 2001.

[12] S. Krause and R. Polikar, "An Ensemble of Classifiers Approach for the Missing Feature Problem," *Int. Joint Conf. on Neural Networks*, 2003, . 553-556.

[13] P.K. Sharpe, R.J. Solly, "Dealing with missing values in neural network-based diagnostic systems," *Neural Computing and Applications*, vol. 3, pp. 73-77, 1995.

[14] C.L. Blake and C.J. Merz, "UCI Machine Learning Repository," [Online Document], Accessed 25 Nov 2006 . http://www.ics.uci.edu/~mlearn/MLRepository.html