SAS 2006 – IEEE Sensors Applications Symposium
Houston, Texas USA, 7-9 February 2006

An Ensemble Technique to Handle Missing Data from Sensors

Hussein Syed Mohammed, Nicholas Stepenosky and Robi Polikar*
Electrical and Computer Engineering, Rowan University,
136 Rowan Hall, Glassboro, NJ 08028, USA.
Email: {syedmo77, stepen91}@students.rowan.edu
*Corresponding author: polikar@rowan.edu

Abstract – Automated classification is often used in advanced systems to monitor system events. All data, and hence features from all sensors, must be present in order to make a meaningful classification. An ensemble approach, Learn^{++}.MF, was recently introduced that allows classification with up to 10% of feature missing, where several classifiers are trained on random subsets of the available sensor data. Given an instance with missing features, only those classifiers trained with the available features are then used in classification. In this paper, we present a modified approach that accommodates up to 30% missing features along with the effect of varying algorithm parameters.

I. INTRODUCTION

A. *The Missing Feature Problem*

Most classification algorithms require that the number and nature of the features be set before the training. Once the training is complete, the field data must contain the exact same number of features as the training data for the classifier to make a decision. Instances missing even a single feature cannot be processed. It is not unusual for training, validation or field data to have missing features in some (or even all) of their instances, as bad sensors, failed pixels, malfunctioning equipment, unexpected noise causing signal saturation, data corruption, etc. are familiar scenarios in many practical applications. A pragmatic approach often used in such cases is to ignore such instances with missing features. This rather brute-force approach is suboptimal, and may not even be feasible if all instances are missing one or more features. There are other theoretical approaches all of which are popular research topics in pattern recognition. Many of them rely on Bayesian or other estimation techniques for extracting "class" probabilities from partial data, by integrating or averaging over missing portions of the feature space [1,2]. Such methods also include data imputation [3], and expectation maximization [4]. Another alternative is searching for the optimal subset of features so that fewer features are required; however, the problem still remains if one (or more) of these optimal features is missing.

Learn^{++}.MF follows an alternate strategy, combining an ensemble of classifiers approach with random feature selection [5]. It is inspired in part by the Random Subspace Method (RSM) [6], where an ensemble of classifiers are trained using random subsets of the features to improve the diversity of the ensemble to aid in its generalization performance, or in optimal feature selection [7]. We explore an alternate application of this strategy, namely, the missing features problem. In essence, by having access to a large number of classifiers each trained with a different random subset of features, an instance with a missing feature or features can be classified by those classifiers which did not use the (currently) missing features in their training. These classifiers are henceforth referred to as *usable classifiers*. The Learn^{++}.MF algorithm aims to be a pragmatic solution to the missing feature problem in sensor data.

B. *Ensemble of Classifiers*

Ensemble based approaches have been well researched for improving classifier accuracy [8,9,10 and 11], however, their potential for addressing the missing feature problem, as well as the incremental learning problem have been mostly unexplored. Learn^{++} was originally developed for the latter, where the algorithm learns from new data, even when instances from new classes or categories are introduced [12]. Learn^{++}.MF is a modification of Learn^{++}, designed to take advantage of the ensemble approach for solving the missing feature problem. In this paper, we present the Learn^{++}.MF algorithm, results on two datasets based on sensor readings, as well as the analysis of its performance for various feature subset sizes and the ratio of features missing.

II. THE LEARN^{++}.MF ALGORITHM

The pseudocode of Learn^{++}.MF is given in Figure 1. The inputs to the algorithm are (1) the training data set D; (2) the percentage of features, *pof*, to be used for training individual classifiers; (3) the number of classifiers to be created T; and (4) the sentinel value *sen* to designate a missing feature. The data set D contains m instances, each with f number of features. At each iteration t=1,…,T, the algorithm creates an additional classifier h_t. Also, a discrete distribution $\boldsymbol{P}_t$ is created on the feature set, which essentially gives a normalized weight to each feature. At iteration t, a subset of features, $\boldsymbol{F}_{selection}(t)$, is drawn according to $\boldsymbol{P}_t$, such that those features with higher weights are more likely to be selected. These features are then used in training current classifier, h_t. $\boldsymbol{P}_1$ is initialized to be uniform, unless there is reason to choose otherwise, so that each feature initially has equal likelihood of being selected into $\boldsymbol{F}_{selection}(1)$. This is done to allow each classifier to be as diverse as possible.

For each iteration, $\boldsymbol{P}_t$ is first normalized to obtain a legitimate distribution (step 1). Next, *pof* % of the features are randomly drawn from $\boldsymbol{P}_t$ (step 2) which constitute the set $\boldsymbol{F}_{selection}(t)$. The t^{th} classifier h_t is trained (step 3) using the features in $\boldsymbol{F}_{selection}(t)$ and tested on training data (step 4). We require that h_t achieves a minimum of 50% correct classification on its training data to ensure that a meaningful classification capacity. Next, the distribution $\boldsymbol{P}_t$ is updated (step 5) such that the weights of those features that appear in $\boldsymbol{F}_{selection}(t)$ are reduced by a factor of *f*. Those features that were not in the current selection effectively receive higher weights when $\boldsymbol{P}_t$ is normalized again in step 1 of iteration $t+1$. This strategy helps ensure that every feature, on average, gets an equal probability of being selected.

```
Algorithm Learn++.MF
Input:
  • Sentinel value sen.
  • Integer T, specifying the number of iterations.
  • Training data set $D = \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1,\cdots,m\}$ with m
    instances and f features.
  • Percentage of features, pof, used to train each
    classifier
                          Training

Initialize $P_1(j) = 1/f, \forall j, j = 1,\cdots,f$
Do for $t = 1,2,...,T$:
    1. Normalize $P_t$ so that it is a proper distribution.
    2. Draw pof% of features for $\boldsymbol{F}_{selection}(t)$ from $\boldsymbol{P}_t$.
    3. Generate a weak classifier $h_t$ using only those
       features in $\boldsymbol{F}_{selection}(t)$ for each instance in training.
    4. Calculate the performance of this classifier, $Perf_t$
       on D. If $Perf_t < 50$ %, discard $h_t$ and go to step 2.
    5. Reduce the distribution weight of the current
       feature $\boldsymbol{F}_{selection}(t)$ set by a constant factor.
end loop

                      Validation / Testing

Do for $i = 1,2,...,m$:
    1. Let $\boldsymbol{M}_{feat}(i)$ be the missing features for $\mathbf{x}_i$
    2. Combine those classifiers whose feature set did not
       include $\mathbf{M}_{feat}(i)$ using weighted majority voting.
end loop
```

Fig. 1. Pseudo code of Algorithm Learn^{++}.MF

During the validation phase, the algorithm searches for sentinels (place holders for missing values). To ensure that actual values are not mistaken for the sentinel, it should be chosen as a value or character that is not expected to occur in the data. All features j, $j=1,\ldots,f$ with a sentinel value in the given instance are then flagged and placed into the set of missing features $\boldsymbol{M}_{feat}(i)$ for that instance $\mathbf{x}_i$. Finally, all classifiers h_t whose feature selection list $\boldsymbol{F}_{selection}(t)$ did not include those in $\boldsymbol{M}_{feat}(i)$ (that is, all classifiers that did not use any of the features in $\boldsymbol{M}_{feat}(i)$) are combined through majority voting to determine the classification of instance $\mathbf{x}_i$.

III. SIMULATION RESULTS

The algorithm was tested on a two real world databases. In each case, different percentages of features (the *pof* parameter) were used for training. The databases include a gas identification database and a database classifying radar returns through the ionosphere (a benchmark database from UCI [13]). In all cases, multilayer perceptron (MLP) networks were used as the base classifier. Since MLPs in the ensemble were trained as weak classifiers, training parameters, such as the error goal or number of hidden layer nodes did not need fine tuning. We define the *total number of features* in the dataset as the number of features per instance times the number of instances, and a single *test trial* as evaluating the algorithm with 0.0%, through 30.0% of total number of features randomly missing from the test dataset. For each *pof*, the *test trials* are repeated 10 times whose average values and confidence intervals are presented below. Missing features were simulated by randomly replacing actual feature values with sentinels.

A. Gas Identification (GI) Database

This database consisted of responses of six quartz crystal microbalances (QCMs) to five volatile organic compounds, including ethanol (ET), xylene (XL), octane (OC), toluene (TL), and trichloroethylene (TCE). Of the 384 six-dimensional signals, 180 were used for training and 204 for testing, whose distributions appear in Table 1. A single optimized classifier trained using all features tested on the test dataset with no missing features performed at 86.23%, setting the benchmark target for our experiments. Two values of *pof* were considered: 33.3% and 50%, corresponding to 2 and 3 features, respectively (out of 6). *T* was set as 100 classifiers. The results are presented in Tables 2 and 3.

Table 1. Data Distribution of GI Data

	Training Data	Test Data
ET	30	34
OC	30	34
TL	50	62
TCE	30	34
XL	40	40
Total	180	204

Table 2. Performance on GI Data (pof = 33.3%)

%Features Missing	% Mean Performance of 10 Trials	% Instances Processed
0.00	83.3 +/- 0.00	100
2.50	82.6 +/- 0.64	100
5.00	82.6 +/- 0.52	100
7.50	82.6 +/- 0.52	100
10.00	82.0 +/- 0.76	100
20.00	80.5 +/- 1.50	100
30.00	79.0 +/- 1.78	100

Table 3. Performance on GI Data (pof=50%)

% Features	% Mean Performance of 10 Trials	% Instances Processed
0.00	85.3 +/- 0.00	100
2.50	85.3 +/- 0.30	100
5.00	85.1 +/- 0.54	100
7.50	84.2 +/- 1.07	100
10.00	84.0 +/- 1.11	99.5
20.00	82.4 +/- 1.05	99.5
30.00	80.9 +/- 1.20	98.5

Figure 2 shows the performance of the Learn^{++}.MF algorithm on the GI database. We observe that the algorithm performed quite well, close to its target performance of 86.23%, even for substantial amount of missing features. It clearly shows, as expected, that the performance declines as the percent of missing features increases. It should be noted further that an increase in the *pof* does increase performance initially, albeit mildly, however the performance drop is steeper with increased missing features. As the percent of missing features increases, the performance of the ensemble that was trained on a smaller *pof* drops much slower than the ensemble that was trained on a larger *pof*.

The last column indicates the percent of instances that could be processed by the ensemble. Note in the last column of Table 3 that, as the percent of missing features increases, there were some instances that could not be processed by any of the available classifiers. However, this number was relatively small, since the total number of features was relatively small (six) to begin with.

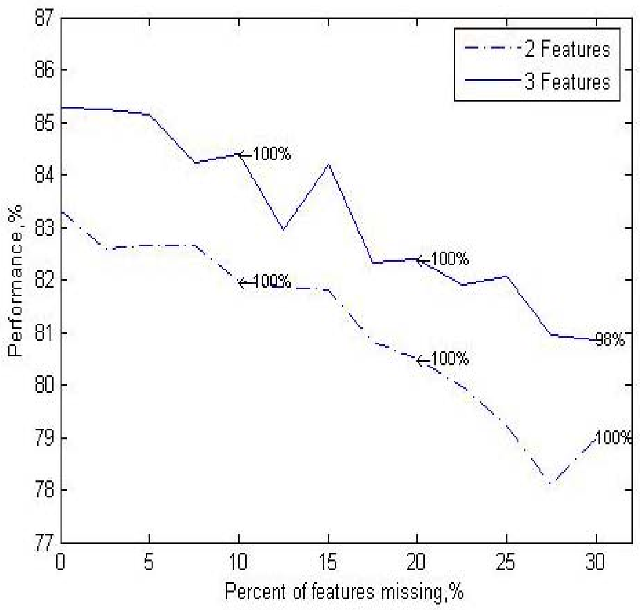

Fig. 2. Mean Performance of Learn^{++}.MF on GI Data over 10 trials

This result demonstrates the need to have a large number of classifiers in order to include as many combinations of feature sets possible. Despite the fact that the ensemble that was trained on a smaller *pof* had a better mean performance as the percent of missing features increased, it should be noted that it was only able to classify a smaller number of instances. As discussed next, the number of instances that cannot be processed can be reduced by increasing the total number of classifiers generated, but it is also more severely affected from increased percent of missing features, when the total number of features is high to begin with.

B. Ionosphere Radar Return (ION) Database

This benchmark database obtained from the UCI machine learning repository [13], consisted of 60 training instances and 210 test instances of radar returns through the ionosphere. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kW. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; whose signals pass through the ionosphere. The data distribution is given in Table 4. A single strong classifier trained with all features tested on the test dataset with no missing features performed at 92.65%. We used 3 sets of *pof* to train the ensemble separately. The network in the ensemble was trained with 8 (*pof* = 23.5%), 10 (*pof* = 29.4%) and 12 (*pof* = 35.3%) out of 34 of the available attributes. Each instance was tested with an ensemble of 1000 classifiers. The mean classification performances for the ION data are presented in Tables 5-7.

Table 4. Data Distribution of ION Data

	Training Data	Test Data
Good	30	100
Bad	30	110
Total	60	210

Table 5. Performance on ION Data (pof = 23.5%)

% Features	% Mean Performance of 10 Trials	% Instances Processed
0.00	84.9 +/- 0.00	100
2.50	84.3 +/- 0.35	100
5.00	84.5 +/- 0.29	100
7.50	83.9 +/- 0.22	100
10.00	83.5 +/- 0.49	100
20.00	83.2 +/- 0.59	99.5
30.00	80.7 +/- 1.58	93.8

Table 6. Performance on ION Data (pof = 29.4%)

% Features	% Mean Performance of 10 Trials	% Instances Processed
0.00	87.7 +/- 0.00	100
2.50	87.4 +/- 0.34	100
5.00	87.3 +/- 0.44	100
7.50	86.6 +/- 0.44	100
10.00	86.2 +/- 0.56	100
20.00	84.6 +/- 1.14	98.6
30.00	78.4 +/- 1.50	82.9

Table 7. Performance on ION Data (pof = 35.3%)

% Features	% Mean Performance of 10 Trials	% Instances Processed
0.00	90.9 +/- 0.00	100
2.50	90.7 +/- 0.28	100
5.00	90.6 +/- 0.59	100
7.50	90.5 +/- 0.50	100
10.00	90.9 +/- 0.39	100
20.00	86.0 +/- 0.93	95.7
30.00	79.3 +/- 1.49	70.0

Several interesting observations can be made from Tables 5-7. First, we observe that the algorithm performs quite well, similar to its target benchmark value of 92.65%, even for relatively large percent of features missing. Specifically, there is very little or no performance drop up until 10% of the features missing. Second, using a larger *pof* value provides a better initial performance with fewer missing features, but high *pof* values also cause a more severe performance drop as the percent of missing features past the 10% mark. For example, the initial performance of the algorithm using 6 out of 34 features (*pof*=17.6%) is 89.8% with no features missing, and drops only to 84.4% when 30% of the features are missing.

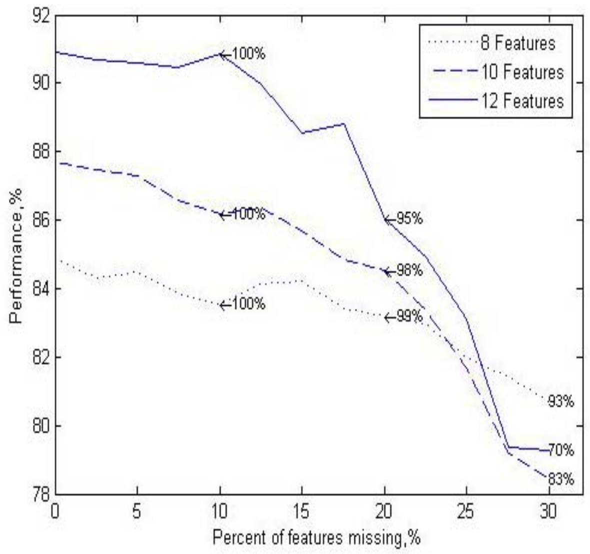

Fig. 3. Mean Performance of Learn^{++}.MF on ION Data over 10 trials

Conversely, when the classifiers are trained with 12 of the 34 features (*pof*=35.3%), the initial performance is 91% with no features missing, but it drops to 79% when 30% of the features are missing. This makes sense: a high *pof* value yields better individual classifiers resulting higher generalization performances, because each classifier is trained with more features. Figure 3 shows that fewer classifiers are then available to classify instances with missing features. Conversely, a low *pof* value cause individual classifiers to be weaker, yet, since only few features are needed to make a classification, a larger percent of missing features can be accommodated.

This argument also allows us to explain the effect of *pof* on the percent of instances that can be processed by the ensemble. The ensemble approach allows the algorithm to be able to process virtually the entire dataset up to 20% of the features missing. However, the ratio of the instances that can be processed by the ensemble decreases sharply with 30% missing features particularly when *pof* is high: with larger number of features used for training, fewer classifiers are available to accommodate when many features are missing. In fact, there may be no classifier available for a specific combination of missing features. Hence, this justifies the training of a larger number of classifiers when dealing with larger set of features. Of course, in the limiting case, if all features were used for training (*pof*=100%), no classifier would then be available even for a single missing feature, which brings us back to the motivation behind using an ensemble trained with random feature subsets

IV. DISCUSSIONS & FUTURE WORK

We present the Learn^{++}.MF algorithm as an alternate and practical solution to the missing feature problem, a common problem in many sensors applications. The algorithm creates an ensemble of classifiers, each trained with a random subset of the features, so that any instance with missing features can still be classified using classifiers that did not use those missing features in their training. Thus far, the algorithm has performed remarkably well for up to 10 – 15% of the features missing, typically for any *pof* value in the 15-50% range.

Two databases, drawn from practical real life applications, were used to evaluate the proposed algorithm. The initial results have been promising, indicating the feasibility of the approach for such applications. The behavior of the algorithm in terms of varying the *pof* and the effect of its performance with the increase in the percent of missing features has been fairly constant.

The algorithm Learn^{++}.MF would be particularly useful, when one or more of the sensors malfunction, or when some of the data become corrupt. It may be expensive, difficult, impractical or even impossible to recollect such data, making it essential to be able to classify data with missing features. However, it should be noted that the algorithm is still useful in the case of no missing data.

Our future work entails investigating and deriving the number of classifiers needed in an ensemble of classifiers to obtain meaningful performances.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. ECS-0239090, “CAREER: An Ensemble of Classifiers Approach for Incremental Learning.

REFERENCES

[1] V. Tresp, R. Neuneier, S. Ahmad, “Efficient methods for dealing with missing data in supervised learning,” G. Tesauro, *et al.*(eds), *Adv. in Neural Inf. Proc. Sys. 7*. MIT Press, 1995.

[2] A. Morris, M. Cooke, P, Green, “Some solutions to the missing feature problem in data classification, with application to noise robust ASR,” *Proc. Int. Conf. Acoustics, Speech, and Signal Proc.,* vol. 2, pp: 737 - 740, 1993.

[3] 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. XXX, P 2.1.25, 2005.

[4] M.Jordan, R.Jacobs, "Hierarchical mixtures of experts and the EM algorithm," Neural Comp., vol.6, no. 2, pp. 181-214, 1994.

[5] S. Krause and R. Polikar, “An Ensemble Approach to the Missing Feature Problem,” *Int. Joint Conf on Neural Net.* vol. 1, pp. 553-558, Portland, OR, 2003.

[6] T.K. Ho, “The Random Subspace Method for constructing Decision Trees,” *IEEE Transactions Pattern Analysis and Machine Intelligence,* vol. 20, no. 8, pp. 832-844, 1998.

[7] M. Skurichina and R Duin, “Combining Feature Subsets in Feature Selection,” *LNCS* 3541, pp. 165-175, 2005.

[8] J. Kittler, M. Hatef, R.P. Duin, J. Matas, “On combining classifiers,” *IEEE Trans. on Pattern Analysis and Machine Intelligence,* vol. 20, no.3, pp. 226-239, 1998.

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

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

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

[12] R. Polikar, L. Udpa, S. Udpa, V. Honavar, “Learn++: an incremental learning algorithm for supervised networks,” *IEEE Tran. Sys., Man Cyb, C*, vol. 31, pp. 497-508, 2001.

[13] C.L. Blake, C. Merz, UCI Repository of machine learning databases:http://www.ics.uci.edu/~mlearn/MLRepository.html.