S-fold Meta-Combiner for Missing Values Processing: Case Study

IVAN BRUHA
Department Computing & Software
McMaster University
Hamilton, Ont., L8S4K1
CANADA

Abstract
Efficient data mining (DM) algorithms have to contain high-performance procedures for processing real-world databases. One of the problems these efficient DM algorithms are faced by are unknown (missing) attribute values in databases. Therefore, robust DM algorithms should comprise some routines for processing these unknown values when acquiring knowledge from real-world databases.

There exist several such routines for each DM paradigm. Quite a few experiments have revealed that each dataset has more or less its own 'favourite' routine for processing unknown attribute values. One possibility how to process efficiently unknown values is exposed in this paper. We use the covering machine learning algorithm CN4 which contains six routines for unknown attribute values processing. Our system runs these routines independently, and afterwards, a meta-learner (meta-combiner) is used to derive a meta-classifier that makes up the overall (final) decision about the class of input unseen objects.

The meta-combiner encompasses in its internal control structure several parameters that are to be set up by the designer or user of the system. One of the crucial parameters is the number $S$ of subsets which a training set is partitioned into during the meta-learning. We are then talking about $S$-fold meta-combiner. Usually, the 'foldness' $S$ is equal to 2 or the size of the training set. This paper exhibits the performance of the meta-combiner for processing unknown attribute values as a function of $S$. The results of experiments for various values $S$ and various percentages of unknown attribute values on real-world data are presented and analyzed.

Keywords: Meta-learning, meta-combiner, unknown attribute values, S-fold meta-combiner

1 Introduction to Problem

Any proficient data mining (DM) algorithm acquiring knowledge from real-world databases must take various aspects into account (see e.g.[14]). One of the problems these efficient DM algorithms are faced by are unknown (missing) attribute values in databases. Therefore, robust DM algorithms should comprise some routines for processing these unknown values when acquiring knowledge from real-world databases.

This topic has been discussed and analyzed by several researchers in the field of machine learning [3], [4], [6], [12], [13]. Our paper [6] discusses both the sources of 'unknownness' and the six routines for processing unknown attribute values: the covering machine algorithm CN4 [7], [8], a large extension of the well-known algorithm CN2 [2], [9], [10] has been used for the experimental analysis. The paper [6] concludes that each dataset needs more or less its own 'favourite' routine for processing unknown attribute values. It evidently depends on the magnitude of noise and source of unknownness in each dataset. The conclusion is obvious: All the routines should be independently run on a small subset (window) of the given database and the suitable routine is to be selected according to their classification accuracies.

This paper describes another way of processing unknown attribute values. We were inspired by the idea of multiple knowledge, multi-strategy learning, and meta-learning, particularly by the concept of combiner and stack generalizer [11]. This concept is employed as follows. The algorithm CN4 processes a given database for each of six routines for unknown attribute values independently. (We can thus view the CN4 algorithm with various routines as independent base learners.) Consequently, we obtain six independent base classifiers. Also, a meta-database is derived from the results of base classifiers and the meta-learner induces a meta-classifier. We call the entire system meta-combiner (namely Meta-CN4 to emphasize the origin of the algorithm).

Hence, if an unknown object is to be classified, then each base classifier yields its decision (class of the input unseen object) and the meta-classifier combines their results in order to produce the final
(over-all) decision about the class of the given input object.

In this paper, we focus on one important parameter of the meta-combiner. If we go to its detailed structure, it encompasses several parameters that are to be set up by the designer or user of the system. One of the crucial parameters is the number of subsets which a training set is partitioned into during the meta-learning. We are then talking about $S$-fold meta-combiner. Usually, the ‘foldness’ $S$ is equal to 2 or the size of the training set. This paper exhibits the performance of the meta-combiner for processing unknown attribute values as a function of $S$. The results of experiments for various values of the parameter $S$ and various percentages of unknown attribute values on real-world data are presented and analyzed.

The way CN4 processes unknown attribute values are briefly presented in Section 2. Section 3 introduces the principle of the meta-combiner. Experiments exhibiting the performance of Meta-CN4 are discussed in Section 4. The results are analyzed in Section 5.

2 Unknown Attribute Value Processing

The inductive algorithm CN4 generates decision rules from a set of $K$ training examples (objects), each accompanied by its desired class $C$, $r = 1, \ldots, R$. Examples (objects) are formally represented by $N$ attributes which are either discrete (symbolic) or numerical (continuous). A discrete attribute $A_r$ comprises $J(n)$ distinct values $V_1, \ldots, V_{J(n)}$; a numerical attribute may attain any value from a continuous interval $\mathbb{R}$.

To deal with real-world situations, it is necessary to process incomplete data, i.e. data with unknown attribute values. Six routines for processing of unknown attribute values were designed for CN4 [6]. The following natural ways of dealing with unknown attribute values were incorporated:

(i) ignore the example (object) with unknown values (routine Ignore),

(ii) consider the unknown value as an additional regular value for a given attribute (routine Unknown), or

(iii) substitute the unknown value for matching purposes by a suitable value which is either

• the most common value (routine Common), or

• a proportional fraction (routine Fraction),

or

• a random value from the probabilistic distribution (routine Random), or

• any value of the known values of the attribute that occur in the training set (routine Anyvalue).

Treating unknown attribute values is determined by the following statistical parameters (here the classes are subject to the index $r = 1, \ldots, R$, attributes $A_r$ for $n = 1, \ldots, N$, their values $j = 1, \ldots, J(n)$):

• the over-all absolute frequencies $F_{n,j}$ that express the number of examples exhibiting the value $V_j$ for each attribute $A_n$;

• the class-sensitive absolute frequencies $F_{r,n,j}$ that express the number of examples of the class $C_r$ exhibiting the value $V_j$ for each attribute $A_n$;

• the over-all relative frequencies $f_{n,j}$ of all known values $V_j$ for each attribute $A_n$;

• the class-sensitive relative frequencies $f_{r,n,j}$ of all known values $V_j$ for each attribute $A_n$ and for a given class $C_r$.

The underlying idea for learning relies on the class distribution; i.e., the class-sensitive frequencies are utilized. As soon as we substitute an unknown value by a suitable one, we take the desired class of the example into consideration in order not to increase the noise in the data set. On the other hand, the over-all frequencies are applied within classification.

(A) Routine Ignore: Ignore Unknown Values

This strategy simply ignores examples with at least one unknown attribute value before learning. Consequently, this approach does not contribute to any enhancement of processing noisy or partly specified data.

(B) Routine Unknown: Unknown Value as a Regular One

An unknown value is considered as an additional attribute value. Hence, the number of values is increased by one for each attribute that depicts an unknown value in the training set. Note that some special arrangements have to be done for a numerical attribute if processed by this routine [6].

(C) Routine Common: The Most Common Value

This routine needs the class-sensitive absolute frequencies $F_{r,n,j}$ to be known before learning and the over-all frequencies $F_{n,j}$ before classification. An unknown value of a discrete attribute $A_n$ of an example belonging to the class $C_r$ is replaced by the class-sensitive common value which maximizes the Laplacian formula

$$\frac{F_{r,n,j} + 1}{F_{n,j} + R}$$

over $j$ for the given $r$ and $n$. An unknown value within classification is

---

1 There exist several procedures for discretizing numerical attributes. The one implemented in CN4 is described in [8]. An off-line discretization (KEX-preprocessor) [1] can be applied to various symbolic algorithms, including CN4, too.
replaced by the over-all common value which maximizes \( F_{r,j} \) over subscript \( j \). We use here the Laplacian formula within learning because it prefers those attribute values that are more predictive for a given class in the contrary to the conventional 'maximum frequency' scheme. Again, some special arrangements have to be done for a numerical (continuous) attribute if processed by this routine [6].

(D) Routine Fraction: Split into Proportional Fractions
The learning phase requires that the relative frequencies \( f_{r,a,j} \) above the entire training set be known. Each example \( x \) of class \( C \), with an unknown value of a discrete attribute \( A \), is substituted by a collection of examples before the actual learning phase as follows: unknown value of \( A \) is replaced by all known values \( V \) of \( A \) and \( C \). The weight of each split example (with the value \( V \)) is

\[
w_j = w(x) \cdot f_{r,a,j} \cdot j=1,...,|J(n)|
\]

where \( w(x) \) is the weight of the original example \( x \). If a training example involves more unknown attribute values, then the above splitting is done for each unknown value. Again, special arrangements are done for numerical attributes [6].

(E) Random Value: Generate Attribute Value Randomly
An unknown attribute value is replaced by one of the values of the given attribute by utilizing a random number generator; it yields a random number in the range \(<0; 1>\) which is exploited to select corresponding value by utilizing the distribution of its attribute values. In the learning phase, the distribution is formed by the class-sensitive relative frequencies \( f_{s,a,j} \) of all known values \( V \) for each attribute \( A \) and for a given class \( C \). In classification phase, the over-all relative frequencies \( f_{s,a,j} \) are used.

(F) Routine Anyvalue: Any Value Matches
An unknown value matches any existing attribute value of an example (object), both in learning and classification. This routine in fact emulates the situation that a designer of a training database does not care about a value of a certain attribute for a given example (so-called dont-care scenario).

3 Methodology: Meta-Learner
In our experiments with processing missing attribute values, we use the covering machine learning algorithm CN4. It contains six routines for unknown attribute values processing. Our system runs these routines independently, and afterwards, a meta-learner (meta-combiner) is used to derive a meta-classifier that makes up the overall (final) decision about the class of input unseen objects.

In fact, we follow [11] in its idea. Each of the six base learners (CN4 with different routines for processing unknown attribute values) generates a base classifier. Afterwards, the decisions of the base classifiers form a meta-database, a set for training meta-objects (examples) for the meta-learner. The meta-learner then generates a meta-classifier. The meta-classifier does not select the best base classifier (routine for processing unknown attribute values) but rather combines the decisions (predictions, classes) of all the base classifiers. In the classification phase, the base classifiers first derive their predictions (classes, decisions); then a meta-object is derived from these predictions which is then classified by the meta-learner.

More precisely, the meta-combiner consists of two phases: meta-learning and meta-classifying; we will now define both these phases in detail.

A training set is split for the meta-learning purposes into two subsets: the genuine-training and examining ones. The genuine-training subset is applied for inducing the base classifiers; the examining one for generating a meta-database.

Let \( q \) be the \( q \)-th base classifier, \( q=1,...,Q \) (where \( Q \) is the number of the base classifiers; in our project \( Q=6 \)). Each example of the examining subset (examining example) \( x \) generates a meta-object of the meta-database as follows. Let \( z_{q,s} \) be the decision (class) of the \( q \)-th base classifier for the examining object \( x \); then the corresponding meta-object of the meta-database looks as follows:

\[
[z_1, \ldots, z_Q, Z]
\]

where \( z_{q,s} \), \( q=1,...,Q \) is the decision the of \( q \)-th base classifier, \( Z \) is the desired class if the input examining object. This rule is denoted as class-combiner [11].

Let \( T \) be a training set of \( K \) training examples, \( S \) be an integer in the range \( <2;K> \). Let us assume to have \( Q \) different base learners \( BL_{k,q} \), \( q=1,...,Q \), and a meta learner \( ML \). The flow chart of the meta-learner looks as follows:

**procedure** Meta-Learning-Phase(\( T, S \))
1. Partition the training set \( T \) randomly into \( S \) disjoint subsets of equal size (as equal as possible). Let \( T_s \) be the \( s \)-th such subset, \( s=1,...,S \). card(\( T_s \)) the number of its objects (examples)\(^2\)
2. Form \( S \) pairs \( [T_s, T \setminus T_s] \), \( s=1,...,S \)^3
3. Let Metadatabase be empty

\(^2\) The splitting (partition) procedure has to preserve the original distribution of classes as in \( T \).

\(^3\) For each \( s, T \setminus T_s \) is the genuine-training subset and \( T_s \) the examining one, generated from the training set \( T \).
4. for s=1,...,S do
   4.1 Train all base learners \( B_{L_s} \) using the genuine-training subset \( T' \setminus T_s \); the result is \( Q \) base classifiers \( B_{Cl_q} \), \( q=1,...,Q \)
   4.2 Classify the examining objects from \( T_s \) by these base classifiers
   4.3 Generate card(\( T_s \)) meta-objects using the above class-combiner rule and add them to \( \text{MetaDatabase} \)
endo
5. Train the meta-learner \( ML \) using the meta-database \( \text{MetaDatabase} \); the result is a meta-classifier \( MCl^* \)
6. Generate the base classifiers \( B_{Cl_q^*} \), \( q=1,...,Q \) using the entire training set \( T \)
   which will be used in the meta-classification

Similar scenario is applied for the classifying an unseen object \( x \):

procedure Meta-Classifying-Phase(\( x \))
1. Classify the unseen object \( x \) by all \( Q \) base classifiers \( B_{Cl_q^*} \) (generated in the step 6 of the meta-learning phase); let the output of the \( q \)-th base classifier \( B_{Cl_q^*} \) be \( z_q \), \( q=1,...,Q \)
2. Generate the corresponding meta-object \([z_1, ..., z_Q]\) by utilizing the class-combiner rule
3. Classify the above meta-object by the meta-classifier \( MCl^* \); its result (decision) is the class to which the given input object \( x \) is classified

The number \( S \) of split subsets is crucial for this system. Therefore, we call it \( S \)-fold meta-learner or \( S \)-fold meta-combiner. The paper [11] introduces two architectures of their meta-system: combiner and stacked generalization. Their combiner corresponds to the 2-fold and stacked generalized to the \( K \)-fold meta-learner.

This was the reason, why we in this paper focused on the ‘foldness’ \( S \) of the meta-combiner. In the following, we present a few experiments whose purpose is to compare the performance of the \( S \)-fold meta-combiner for various values of the parameter \( S \).

4 Experiments

In order to find out empirically the dependence of the meta-combiner’s performance on various values of its ‘foldness’ \( S \), we carried out several experiments. Unlike the previous experiments and analysis [5] (where we compared Meta-CN4 and Meta-ID3 with the original CN4 with various routines for unknown attribute value processing, as well as C4.5), here we focus just on Meta-CN4; we have studied its performance for various values \( S \) and various percentages of unknown attribute values on real-world data.

All the above experiments were tested on four databases whose description can be found e.g. in [5], [6]. Each database was randomly split to two sets (70% training, 30% testing) and this scenario has been executed 10 times for each combination. The following table and figure thus involve in each slot an average of classification accuracy (of testing sets) acquired from 40 runs.

We have to realize that the above splitting procedure has nothing common with the splitting procedure within the meta-learner. The 70% of training examples are furthermore split within the \( S \)-fold meta-learner into a genuine-training subset of the size \( 70*(S-1)/S \) % and an examining subset of the size \( 70/S \) % of the original database.

Following [11], we selected 2-fold, then the promising 4-fold meta-combiner (\( S=4 \)), and also \( S=8 \), \( S=16 \), and \( S=32 \) for comparison.

To achieve extensive and comprehensive comparison of the above meta-combiners’ behaviour we have also decided to find how classification accuracy depends on various percentage of unknown attribute values in databases. One database of ours exhibit a reasonable size of ‘unknownness’. As for the remaining databases, to emulate various number of unknown values, we have run the original data through a filter which randomly changes attribute values to unknown ones. The filter procedure (‘unknownizer’) has the percentage of unknown attribute values as its parameter.

Table 1 comprises the average classification accuracy (in %) above all four datasets for various percentage of unknown values and various values of the parameter \( S \). Figure 1 then depicts the same dependancy.

5 Conclusion

This research paper continues in our larger project whose purpose is to design, implement, and empirically compare the meta-learner Meta-CN4 with other algorithms for processing of missing attribute values. Namely, this paper exhibits a portion of the above project that considers the importance of the ‘foldness’ \( S \) of such a meta-combiner as its crucial parameter. The only, but widely used criterion in our experiments was the classification accuracy acquired from testing sets.

By analyzing the results of our experiments we came to the following:
• Although there were carried out experiments only for a few values of the parameter \( S \), we can ob-
serve that there is the ‘optimal’ value $S$ that maximizes the classification accuracy. One can easily observe it namely along the series for $S=32$ that exhibits always worse performance than that for other values.

• To be more precise, the statistical results of the t-test (with the confidence level 0.05) depict that the performance of the meta-combiner for $S=4$, $S=8$, and $S=16$ are statistically equivalent, but they are significantly better than that for $S=2$ and $S=32$.

• Because of time limitations, we did not perform more experiments. We did not use the stack generalizer (fold $S=K$) because it is much more time consuming; the paper [11] indicates that the timing cost for the stack generalizer is much more larger than that for the meta-combiner for relatively small parameters $S$.

For the future research, we plan to perform much more experiments and to study how the optimal value of the parameter $S$ depends on a processed database. It is just our impression that even for this issue (to find an optimal value of $S$), we would need to introduce another ‘meta-level’.

References:


Fig. 1. Averaged classification accuracy vs. percentage of unknownness for various values of the parameter ('foldness') $S$.

Table 1. Average classification accuracy (in %) of Meta-CN4 as a function of unknownness and ‘foldness’ $S$ (above all four databases).

<table>
<thead>
<tr>
<th></th>
<th>$S=2$</th>
<th>$S=4$</th>
<th>$S=8$</th>
<th>$S=16$</th>
<th>$S=32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5%</td>
<td>79.6</td>
<td>80.6</td>
<td>80.7</td>
<td>80.7</td>
<td>79.6</td>
</tr>
<tr>
<td>10%</td>
<td>78.5</td>
<td>79.3</td>
<td>79.2</td>
<td>79.4</td>
<td>78</td>
</tr>
<tr>
<td>20%</td>
<td>77</td>
<td>77</td>
<td>77.4</td>
<td>77</td>
<td>74.7</td>
</tr>
<tr>
<td>30%</td>
<td>75.6</td>
<td>76.7</td>
<td>76.3</td>
<td>76</td>
<td>75.5</td>
</tr>
</tbody>
</table>