A Non-Parametric EM-Style Algorithm for Imputing Missing Values

Rich Caruana
Center for Automated Learning and Discovery
Carnegie-Mellon University
Pittsburgh, PA 15213
caruana@cs.cmu.edu

Abstract

We present an iterative non-parametric algorithm for imputing missing values. The algorithm is similar to EM except that it uses non-parametric models such as k-nearest neighbor or kernel regression instead of the parametric models used with EM. An interesting feature of the algorithm is that the E and M steps collapse into a single step because the data being filled in *is* the model – updating the filled-in values updates the model at the same time.

The main advantages of this approach compared to parametric EM methods are that: 1) it is more efficient for moderate size data sets, and 2) it is less susceptible to errors that parametric methods make when the parametric models do not fit the data well. The robustness to model failure makes the non-parametric method more accurate when models of the data are not known apriori and cannot be determined reliably. We evaluate the method using a real medical data set that has many missing values.

1 Motivation

The Port Pneumonia Database [2] contains 2287 pneumonia patients. For our experiments we use 216 of the 1000+ features available in the raw database. 6.3% of the values of the 216 attributes are missing. More than 99% of the cases in the database are missing at least one of the 216 values, and one of the 216 attributes is missing in 61% of the cases. Values in this database may be missing because they were not recorded, because they were not measured, or because they were assumed normal.

We wanted to apply various machine learning methods to this data set to see how they would perform. Some of the machine learning methods we wanted to test cannot easily handle missing values [2]. One way to deal with missing values when using a method that does not deal with them naturally is to add an additional value to each attribute that indicates "missing", or to add indicator variables that are true iff the variable they indicate for is missing. One potential problem with this approach is that whether or not a value is missing may be predictive of an outcome we want to predict, but in a way that will not hold true for future test cases. For example, if we want to predict whether a pneumonia patient should be hospitalized or treated as an outpatient, the values for lab tests that usually are ordered for patients in the hospital may often be missing for outpatients. Knowing that theses variables are missing or not is predictive in the training set, but misleading on the test set where these values would often be missing because none of the patients have yet been admitted to the hospital. This problem could be solved if we had time stamps indicating when the variables where measured, but this information is not available in the dataset.

To evaluate if the pattern of missing values was informative, we replaced all missing values with the value "1", and all non-missing values with the value "0". We then used k-nearest neighbor (kNN) to predict one of the outcomes we were interested in from the recoded data. Recoding the data this way eliminates all information except for which values are missing or not. The outcome could be predicted from the recoded data with 85% accuracy. For comparison, the baseline accuracy of predicting the predominant class is 65%. This result indicates that the pattern of missing values is informative.

More than 99% of the cases in the database have at least one missing value, so we cannot throw away cases that have missing values. The experiments above indicate that the pattern of missing values is informative for one of the outcomes we wish to predict, so we cannot safely use indicator variables or special values to code for missing. This is left us one choice: to impute the missing values.

Because the data is real, we did not know what parametric distribution to use to model each of the 216 variables. Studying each of the 216 variables carefully enough to make reliable decisions about what parametric model to use for each variable appeared daunting. We were concerned that if we used the wrong distribution for a *few* variables and then used EM to impute missing values, it might hurt the filled-in values for *other* variables as well, not just the variables that were poorly modelled. This led us to develop the nonparametric imputation method described here. The method uses k-nearest neighbor to iteratively improve the quality of the filled-in values. It is a nonparametric algorithm that can be viewed as a form of EM that depends on Gibbs sampling instead of parametric models.

2 The Algorithm

Consider a data set containing the set of cases {C}. Some attribute values are missing for some cases in {C}. We refer to missing values as "nknown", even after they have been filled in because their true values are never known. We refer to values that were never missing are "known". We use the term "missing" to refer to values that have not yet been filled in by the algorithm. Values that were missing but have been estimated are referred to as "filled-in" or "imputed."

The algorithm is easiest to understand using 1-nearest neighbor (1NN). Extending the algorithm to use kNN or kernel regression is straightforward. See Table 1 for pseudocode for the 1NN version of the algorithm.

Each iteration through the Repeat loop *re-estimates* the imputed values for each unknown attribute value. The process is repeated until the filled-in values converge or begin to cycle (more about cycling later). The key difficulty is finding nearest neighbors when c and/or its neighbors have missing values for attributes *other than* the attribute in c that is being filled-in. The approach the algorithm follows is to use imputed values from the previous iteration in the nearest neighbor distance calculation. This solves the problem of how to compute distances between cases that have missing values because there are estimates for the values that were missing. The goal of iteration is to improve the filled-in values so that nearest neighbors can be found more accurately. If nearest neighbors can be found more accurately, the filled-in values derived from those nearest neighbors also should be more accurate.

More specifically, given a case c and an attribute a whose value is unknown, we want to find the nearest neighbor n to c in the data set that has a known value for attribute a. To do this, we compute the distance between c and every other case in the data set that has a known value for attribute a. If 1-nearest neighbor is being used for imputation, we then pick the case whose distance is smallest, and use it's value for a as the new filled-in value for attribute a in case c.

Table 1: Pseudocode for non-parametric imputation of missing values using 1-nearest neighbor (1NN).

```
IMPUTE(dataset{C}, DIST())

  Inputs:
     {C}: data set with M cases of N attributes
     DIST(): Distance Metric defined on cases
             that have no missing values

  Repeat until convergence {
  for c ∈ {1...M} do
     for a ∈ {1...N} do
           If attr(c, a) = unknown do
              Re-impute attr(c, a) using 1NN:
              for n ∈ {1..M} such that
                    (c ≠ n) and
                    attr(n, a) ≠ missing and
                    ∀j ≠ c, nDist(c, n) ≤ Dist(c, j)
              attr(c, a) ← attr(n, a)
  }
```

3 Jump-starting the Process

During the first iteration the unknown values have not yet been filled-in. How do we compute the difference between the values of attribute x when x is unknown for c or its neighbor? We compute the average value when x is continuous (or the mode if x is discrete) from the cases with known values of x, and use the average(mode) as the initial filled-in value for x.

If, however, c and its neighbor are both missing the value of x, we compute the average difference between the values of x over the cases in the dataset that have known values of x, and use this difference as an initial estimate of the distance between c and its neighbor along variable x. Note that once filled-in values have been re-estimated during the first iteration, these jump-start procedures no longer are needed because filled-in values now exist for all values that were missing.

4 A Small Example

Suppose we have a database with 9 cases and five ordinal attributes A-E.

	A	B	C	D	E
1	2	4	9	6	5
2	6	8	1	9	7
3	6	7	1	8	9
4	9	2	6	2	3
5	1	4	8	5	6
6	9	2	6	3	2
7	1	3	9	4	6
8	8	2	5	3	2
9	5	8	3	9	7

Suppose that missing values have been peppered throughout this database. The table below has 12 missing values. Each attribute is missing for at least one case, and each case has at least one missing value.

	A	B	C	D	E
1	2	*?*	9	6	5
2	*?*	8	1	9	7
3	*?*	7	1	*?*	*?*
4	9	*?*	6	2	3
5	1	4	8	5	6
6	9	2	6	*?*	2
7	*?*	3	9	4	6
8	8	2	*?*	3	*?*
9	5	*?*	3	*?*	7

The first step is to fill-in each missing value with the mean/median values calculated from cases that are not missing that value. This allows kNN to be applied to re-estimate the filled-in values using the algorithm described above. If both cases being compared were missing values for attribute x, the distance along dimension x will be 0 since they will be filled in with the same value. This causes cases missing many values to appear to be artificially close to each other, which is not ideal. As described above, when both cases are missing the value of x, we use the mean distance between cases not missing x as the estimated difference along dimension x in the first loop. This subtlety is not critical for the proper behavior of the method, but does speed convergence on data sets that have many missing values.

The first iteration of the algorithm yields the filled-in table below. Five of the filled-in values are correct after the 1st iteration. Another five are close to the correct value. Two are substantially wrong: attribute B in case 9 has been filled-in with a value of 2, but the correct value is 8, and the value of attribute C on case 8 should be near 5, not 9. These two filled-in values are this wrong because a single iteration of 1NN is unable to find the true nearest neighbors to cases 8 and 9 given the poor initial estimates for the many other filled-in values.

ITERATION 1

	A	B	C	D	E
1	2	*4*	9	6	5
2	*5*	8	1	9	7
3	*5*	7	1	*9*	*7*
4	9	*2*	6	2	3
5	1	4	8	5	6
6	9	2	6	*2*	2
7	*1*	3	9	4	6
8	8	2	*9*	3	*2*
9	5	*2*	3	*9*	7

A 2nd round of 1NN filling-in (2nd pass through the Repeat loop) improves the filled-in values (below). Eight filled-in values remained unchanged from the previous iteration. Four values changed, including the two filled-in values that had the most error in the previous iteration (attribute B case 9 and attribute C case 8). These now have better filled-in values. But attribute D on case 9, which used to be filled-in with a correct value, now has a poorer value. On average, the new table has more accurate filled-in values, but not every value is as good as it was on the previous iteration.

ITERATION 2

	A	B	C	D	E
1	2	*4*	9	6	5
2	*5*	8	1	9	7
3	*5*	7	1	*9*	*7*
4	9	*2*	6	2	3
5	1	4	8	5	6
6	9	2	6	*2*	2
7	*1*	3	9	4	6
8	8	2	*6*	3	*3*
9	5	*8*	3	*5*	7

One more iteration yields the following filled-in table:

ITERATION 3

	A	B	C	D	E
1	2	*4*	9	6	5
2	*5*	8	1	9	7
3	*5*	7	1	*9*	*7*
4	9	*2*	6	2	3
5	1	4	8	5	6
6	9	2	6	*2*	2
7	*1*	3	9	4	6
8	8	2	*6*	3	*3*
9	5	*8*	3	*9*	7

Only one filled in value changed: attribute D in case 9 switched from a value of 5 in the previous iteration to 9, the correct value. Further iteration does not change the filled-in values on this simple example. The algorithm has converged. The final values are fairly accurate. Five are filled-in correctly, six of the remaining seven are off by 1. Only one value is off by more than 1: attribute E on case 3 is filled-in with a 7, but should be a 9.

5 Relationship to EM

The algorithm we present does not use expectation maximization (EM), yet we describe it as an EM-style algorithm. Why? A real EM algorithm repeatedly alternates between two steps, the E step and the M step. In the E step the expected value of unknown parameters are estimated using the most recent hypothesis. In the M step the hypothesis is updated by calculating the new maximum likelihood hypothesis assuming the unknown parameters take on the expected values from the previous E step. [5] When imputing missing values, the E step corresponds to re-estimating the missing values, and the M step corresponds to inducing a new density model of the data using the filled-in data set from the previous E step.

In most EM algorithms, both the E and M steps depend on parametric models. The parametric model is used to impute the missing values, and then new maximum likelihood parameters for the parametric models are found using the filled-in data. The algorithm we present is non-parametric. The E step in our non-parametric algorithm is evident: the filled-in values are re-estimated using a kernel-based method such as kNN. The M step, however, appears to be missing? Where is a new model formed using the newly estimated values?

If unweighted Euclidean distance is used to measure distances between cases, the M step merges with the E step. Re-estimating the filled-in values updates the model because the data *is* the model for kNN with unweighted Euclidean distance. The M step is implicit in the fact that the newly filled-in data is used as the case-base in the next iteration of the algorithm. Although the algorithm we present has no explicit model of the data, and thus lacks an explicit M step, it's iterations are similar to what occurs in parametric EM.

The similarity between real EM algorithms and the non-parametric EM-style algorithm we present is more obvious if we use weighted Euclidean distance instead of unweighted Euclidean distance. With weighted Euclidean distance, the weights to be applied to each dimension in the distance metric will be estimated from the data. As the data is changed by filling-in, the weights in the distance metric may have to change. Typically, new weights will be selected that maximize the likelihood of the data (or some measure similar to likelihood) given the current filled-in values. Thus an explicit M-like step emerges if the non-parametric kNN method for imputing missing values uses weighted Euclidean distance.

There are two important differences between true EM algorithms and the non-parametric imputation method presented in this paper. The first is that the non-parametric method does not require the parametric models of the variables that the EM methods require. This is an advantage when correct parametric models are not known for all of the variables because parametric models that don't fit the data well can lead to errors that cannot be repaired by additional iterations. Inappropriate parametric models change the fixed-point of the search. The non-parametric method probably has less sample efficiency than the true parametric models, but it also will not suffer from the negative impacts of model error. The non-parametric method is more robust.

The second significant difference between the EM and non-parametric methods is that guarantees can be made for EM that we currently do not know how to make for the non-parametric approach. Each iteration of EM is guaranteed to be nondecreasing in maximum likelihood. Thus EM converges to a local maximum in likelihood. We do not yet know how to make similar guarantees for non-parametric methods of imputing missing values.

6 Convergence of the Filled-In Values

In the previous section we pointed out that with a true EM algorithm it is possible to prove convergence by showing that each iteration is non-decreasing in likelihood, but that we are not able to make similar proofs for the non-parametric method. In this section we empirically show the convergence of the non-parametric method when applied to the pneumonia dataset.

Figure 1 shows the average distance the attribute values move from successive iterations when the algorithm is applied to the pneumonia data set. If the motion drops to zero, no missing value has changed and the method has converged. Note that in this case motion does not drop all the way to zero, indicating that the algorithm converged to a cycle. (We have never seen the method diverge.) Cycles are rare with parametric EM methods if density calculations are exact. They are more likely with the non-parametric kNN approach when k is small or when values are filled-in with single values (as above). Kernel regression, kNN with larger values of k, or filling-in missing values with a sample that extensionally represents a distribution of values (see the Discussion Section) reduce the chances of converging to a cycle. There also are methods for forcing iterative procedures such as this to converge, though we have not found them to be necessary in practice.

Each iteration of imputation requires less than 1 CPU second on a modern workstation with the pneumonia dataset (2287 cases with 216 variables each) using unweighted Euclidean distance.

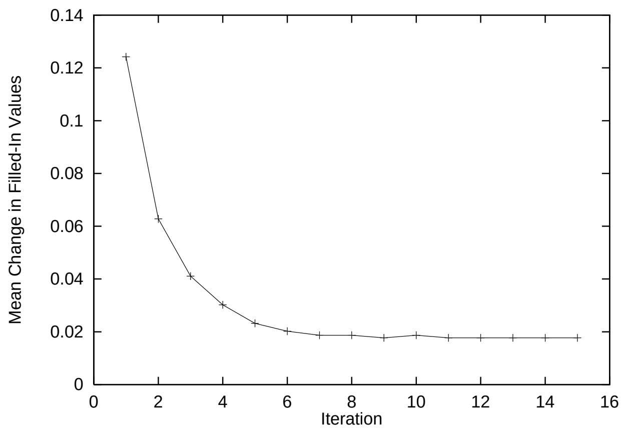

Figure 1: Average change in filled-in values for the pneumonia data set for successive iterations. Attributes were scaled to variance 1.0 prior to applying 1NN, so motions larger than 0.1 represent substantial changes.

7 Quality of the Imputed Variables

The filled-in values in the Port Pneumonia Dataset stabilize after about a dozen iterations using 1NN. Are the filled-in missing values getting better, or just converging? This section presents the results of an experiment where we predict the outcome CARE (inpatient/outpatient) using K=3 nearest neighbor as the learning method after each iteration of filling-in the values in the pneumonia database. (Preliminary experiments indicated that k=3 was optimal for predicting CARE.) Do not confuse k=1 nearest neighbor that is being used to fill-in missing values with the 3NN learning method being used to predict CARE from the filled-in data sets. Figure 2 shows the accuracy with which CARE is predicted as a function of the number of iterations of filling in. The graph shows that CARE is predicted better using the filled-in values of later iterations, suggesting that the quality of the filled-in values is improving.

8 Discussion

Earlier we emphasized that one advantage of the non-parametric method is that it cannot suffer from the

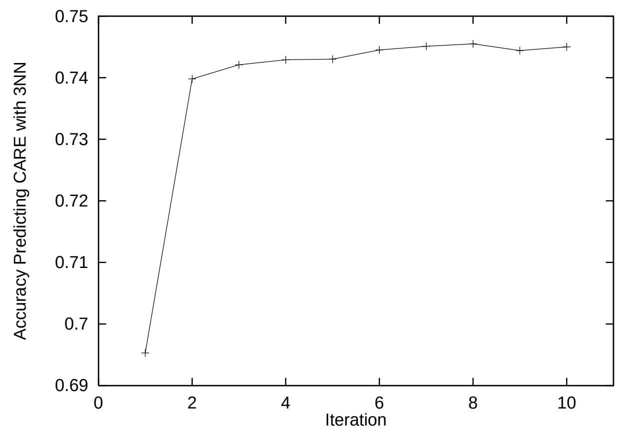

Figure 2: Accuracy of CARE prediction using 3NN from the filled-in database after each iteration of filling in. Most of the benefit occurs during the first few iterations of filling-in.

negative effects of model failure because it does not use parametric models. This does not mean that the non-parametric method is immune to poor modelling of the data. The non-parametric method's model of the data is defined by it's distance metric. While good asymptotic (large sample) performance of kNN methods is insured for most reasonable distance metrics, this does not mean that kNN imputation will perform well for all distance metrics with moderate sample sizes. It is important to use as good a distance metric as possible. Devising good distance metrics is not always easy. For example, sometimes it is dificult to divise a distance metric that combines distances measured between boolean, nominal, and continuous variables. (To be fair, similar difficulties also arise when using Bayesian methods for density estimates of mixed variable types.)

One approach that can be used to bridge the gap between the point estimates used for filled-in values by the non-parametric kNN method and the distribution estimates typically made with full EM methods, is to use kNN to estimate more than one filled-in value for each missing value. [6] For example, one could use kNN to estimate k (not necessarily unique) values for each missing value. These k values form an extenional distribution for each missing value. Distances between neighbors can then be calculated by sampling repeatedly from this set of values, or multiple nearest neighbors can be found by sampling. We have not yet experimented with this enhancement to the method to see what improvement it yields.

The goals for presenting this work at the workshop are: 1) to discuss the pros and cons of non-parametric methods for filling-in missing values (e.g., parametric methods have higher sample efficiency, but can perform poorly when there is model failure); and 2) to discuss ways of proving that non-parametric EM-style algorithms like this one converge (or how best to modify the algorithm to insure convergence).

References

[1] Caruana, Rich, "Iterated K-Nearest Neighbor Method and Article of Manufacture for Filling in Missing Values." United States Patent 6,047,287, Assignee: Justsystem Pittsburgh Research Center, Pittsburgh, Pennsylvania, filed May 5, 1998, granted April 4, 2000.

[2] Cooper, G. F., Aliferis, C. F., Ambrosino, R., Aronis, J., Buchanan, B. G., Caruana, R., Fine, M. J., Glymour, C., Gordon, G., Hanusa, B. H., Janosky, J. E., Meek, C., Mitchell, T., Richardson, T., Spirtes, P., "An Evaluation of Machine Learning Methods for Predicting Pneumonia Mortality." Artificial Intelligence in Medicine 9, 1997, pp. 107-138.

[3] Ghahramani, Z., and Jordan, M.I. (1994) Supervised learning from incomplete data using an EM approach. In J.D. Cowan, G. Tesauro, and J. Alspector (eds.), *Advances in Neural Information Processing Systems.* **6**:120–127. San Francisco, CA: Morgan Kaufmann Publishers.

[4] Lauritzen, S. L. (1995) The EM algorithm for graphical association models with missing data. Computational Statistics and Data Analysis. 19:191-201.

[5] Mitchell, T. (1997). *Machine Learning.* McGraw-Hill, New York. pp:191-6.

[6] Little, R. J. A. and Rubin, D. B. (1987). *Statistical Analysis with Missing Data.* Wiley, New York.

[7] Tresp, V., Ahmad, S., and Neuneier, R. (1994). Training neural networks with deficient data. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, San Francisco, CA. Morgan Kaufman Publishers.