An Analysis of Hot Deck and CLIP4 Algorithm for Imputing Missing Values in Database

Mrs.Kalpana Wani
Department of Computer Science,
PIIT New Panvel, Navi Mumbai, India

Prof. Madhu Nashipudimath
Department of Information Technology
PIIT New Panvel, Navi Mumbai, India

Abstract

One relevant problem in data quality is the presence of missing data. Despite the frequent occurrence and the relevance of missing data problem, many Machine Learning algorithms handle missing data in a rather naïve way. However, missing data treatment should be carefully thought, otherwise bias might be introduced into the knowledge induced. In this work we analyses the use of the Hot Deck and CLIP4 as an imputation method. Imputation is a term that denotes a procedure that replaces the missing values in a data set by some plausible values. Hot-deck imputation is a means of imputing data, using the data from other observations in the sample at hand. The algorithm CLIP4 first partitions data into subsets using a tree structure and then generates production rules only from subsets stored at the leaf nodes. The unique feature of the algorithm is generation of rules that involve inequalities. The algorithm works with the data that have large number of examples and attributes, can cope with noisy data, and can use numerical, nominal, continuous, and missing-value attributes.

Keywords—Missing Value, Production Rule, Machine Learning, Imputation

1. Introduction

All Most of the real world databases are characterized by an unavoidable problem of incompleteness, in terms of missing or erroneous values. A variety of different reasons result in introduction of incompleteness in the data. Examples include Manual data entry procedures, incorrect measurements, equipment errors, and many others. Existence of errors, and in particular missing values, makes it often difficult to generate useful knowledge from data, since many of data analysis algorithms can work only with complete data. Therefore different strategies to work with data that contains missing values and to impute or another words fill in missing values in the data are developed.

Data quality is a major concern in Machine Learning — ML — and other correlated areas, such as Data Mining and Knowledge Discovery from Databases i.e. KDD. Despite the frequent occurrence of missing data in real world data sets, ML algorithms handle missing data in a rather naive way. Missing data treatment should be carefully thought, otherwise bias might be introduced into the knowledge induced. In most cases, data sets attributes are not independent from each other. Thus, through the identification of relationships among attributes, missing values can be determined. Imputation is a term that denotes a procedure that replaces the missing values in a data set by some plausible values. One advantage of this approach is that the missing data treatment is independent of the learning algorithm used. This allows the user to select the most suitable imputation method for each situation.

In general two groups of algorithms used to preprocess databases that contain missing values can be distinguished. First group concerns unsupervised algorithms that do not use target class values. Second group are supervised algorithms that use target class values, and which are most commonly implemented by using supervised ML algorithms [6].The unsupervised algorithms for handling missing data range from very simple methods like Mean imputation to statistical methods based on parameter estimation are used to impute the values.

The objective of this work is to analyses the performance of the Hot-Deck imputation Algorithm and CLIP4 algorithm as an imputation method. The first method is unsupervised learning method which refers to the problem of trying to find hidden structure in unlabelled data. Since the examples given to the learner are unlabelled, there is no error or reward signal to evaluate a potential solution. The other method is a

Supervised learning method in which the data model is used to classify examples into a set of predefined classes, which in case of missing value imputation are just all distinct values of an attribute that has missing values. Second, during the testing step, the generated model is used to impute missing data for the testing data.

2. Hot-Deck imputation Algorithm [4]

Hot-deck imputation is a means of imputing data, using the data from other observations in the sample at hand. This paper deals with a method of imputation we used for the Survey of Adults on Probation. In this method, we first attempt to impute race based on other variables in an observation. We impute the remaining missing race data, and age and gender, using the same variable in prior or latter observations in the data set. This method uses macro variables to subset the data into groups within which imputation is done. It also uses arrays to keep track of values to use for imputation.

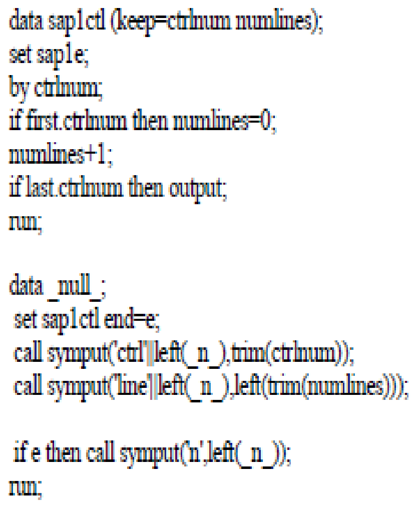
Database used for execution of this algorithm is the Survey database of Adults on Probation based on a sample of prisons from across the country. Inmates were selected using a roster filled out by a probation officer for the given prison. Each prison was considered a group (or ctrlnum), for imputation purposes. The probation officer filled out a questionnaire for each inmate. The questionnaire was later compared to one completed by the inmate. The imputation discussed here was for the data missing from the questionnaire filled out by the probation officer.

The imputation requirements for SAP are based on the data we need to impute, and the survey sample design.

1. We impute age, race and gender independently.
2. For race, we first try to base imputation on other Data (e.g.ethnicity) for the same person.
3. We impute all remaining missing values using only "good" (Unimputed) data for another person in the same group, or Ctrlnum.
4. We use each "good" data value only once for imputation.
5. We work backwards over the ctrlnum then, if necessary, we work forwards.
6. If no "good" data can be used from within the ctrlnum, we assign some type of "out of range" value.

The imputation process is accomplished using four successive steps, taking into account the above requirements. First, we produce a data set containing the number of persons per ctrlnum. Next, we create macro variables for the ctrlnums, and number of persons per ctrlnum, using the output from the first step. As the first step in the actual imputation, we attempt to resolve race based on other data for the same person if possible. We finally use hot-deck imputation to fill in the remaining data, where possible. This is accomplished by first creating the necessary data and flag arrays for age, race and gender. For race, we create a new temporary flag array, so we can retain new race variable imputations. Finally, we work backwards and then forwards through a given ctrlnum, to impute data, by evaluating the data for other persons in the ctrlnum. Further details on these principles, including imputation through data step concepts, are given below.

To produce a data set containing the number of persons per ctrlnum, we "set" the original data set "by" ctrlnum. The data set is sorted by ctrlnum. Within each ctrlnum, we use a counter for the number of lines encountered (numlines), using first. Ctrlnum to initialize numlines. We retain the previous value with each datastep iteration, and output to the new data set only for the last. ctrlnum. In this way, we produce a new data set (sap1ctl). This data set contains one observation for each ctrlnum, with two variables: ctrlnum, and total number of persons in the ctrlnum, numlines. Next, we create macro variables for the above two variables and number of ctrlnums. Using a null datastep, we SET sap1ctl and use the call symput routine to create the macro variables &&ctrlnum&i and &&line&i for the ith observation in sap1ctl. On the last iteration of this datastep we again call the symput routine to create &n, the macro variable for the total number of ctrlnums encountered. The code for this section of the program is given below.



```
data sap1ctl (keep=ctrlnum numlines);
set sap1e;
by ctrlnum;
if first.ctrlnum then numlines=0;
numlines+1;
if last.ctrlnum then output;
run;

data _null_;
 set sap1ctl end=e;
 call symput('ctrl'||left(_n_),trim(ctrlnum));
 call symput('line'||left(_n_),left(trim(numlines)));

 if e then call symput('n',left(_n_));
run;
```

Table 1. Sample Database

```
      SAP1CTL

  CTRLNUM    NUMLINES

   1001          32
   1002          16
   1003          42
   1004          51
   1005          19
   1006          54
   1008          63
   1009         210
   1010          34
   1011          23
```

The overall imputation process is done using a macro, "subseta", within which we work with each ctrlnum, one at a time. This is done with a %do loop using the macro Variable &n, defined above, for the number of ctrlnums in the sample. The last step in this macro is a proc append, which accumulates the component data sets for ctrlnums into a final dataset, sap1imp. Because we're using a PROC APPEND, if it is necessary to rerun the program,sap1imp must be deleted beforehand, since it is the base data added to, during the procedure.

The first step in the imputation process is to attempt to resolve race based on other data for an observation. First, we subset the original data set, sap1e, BY ctrlnum, to get sap1e&i. We do this so we can use the automatic iteration indicator, _n_, in the next datastep in the race resolve process. In the next DATA step, using sap1e&i, we produce data sets of imputed race only (sap1r&i), and ctrlnum-level arrays of imputed race flags (sap1a&i). Each element in the array corresponds to a line where race has (or has not) been imputed. The final steps in the initial race resolve process are to MERGE sap1e&i with sap1r&i, and finally the resulting sap1e&i with sap1a&i. The code for this section of the program, the first part of the macro "subseta", is shown below.

```
%macro subseta;

%do i=1 %to &n;

/*  subset sap1e by ctrlnum, and first resolve race based */
/*  on other data - create flag array for imputing race     */

data sap1e&i;
set sap1e;
if ctrlnum="&&ctrl&i";
run;

data sap1r&i (keep=imprace)
    sap1a&i (keep=ctrlnum racfl1-racfl&&line&i);
set sap1e&i;
by ctrlnum;
if ctrlnum="&&ctrl&i";
retain racfl1-racfl&&line&i;
array arrfl{&&line&i} $ racfl1-racfl&&line&i;
if (b4='5' or b4='8' or b4='') and b5='1' then do;
  imprace='1'; arrfl{_n_}='1'; end;
if ctrlnum='2601' and (b4='8' or b4='') then do;
  imprace='1'; arrfl{_n_}='1'; end;
if last.ctrlnum then output sap1a&i;
output sap1r&i;
run;

data sap1e&i;
merge sap1e&i sap1r&i;
run;

data sap1e&i;
merge sap1e&i sap1a&i;
by ctrlnum;
run;
```

The remainder of the imputation process constitutes the hot deck portion, i.e., the part where we use the corresponding data from previous or latter observations in a given group. The first step in this part is to create ctrlnum-level arrays for race, age and gender, to be included on each person's observation. We use sap1e&i from the last step as input, and use ctrlnum as the BY variable. On each iteration, we feed the value of a demographic variable into the array, and RETAIN the values of the array over iterations. After the last case of a ctrlnum, we output. In this way, we obtain a new data set sap1b&i, containing one observation for each ctrlnum, with one variable for each of the elements in the three arrays.

The new version of sap1e&i resulting from this step contains the ctrlnum-level arrays for race, age and gender. We MERGE sap1b&i with sap1e&i from above, BY ctrlnum, to get a new version of sap1e&i, with an observation for each person. Each person observation has all values for all other persons in the ctrlnum, for each of the three demographic variables. The code for this section of the program is shown in below.

```
/*  create arrays necessary for imputing age, sex and     */
/*  race using hot deck method                            */

data sap1b&i (keep=ctrlnum age1-age&&line&i sex1-sex&&line&i
                       rac1-rac&&line&i);
set sap1e&i;
by ctrlnum;
retain age1-age&&line&i sex1-sex&&line&i rac1-rac&&line&i;
array arage{&&line&i} $ age1-age&&line&i;
array arsex{&&line&i} $ sex1-sex&&line&i;
array arrac{&&line&i} $ rac1-rac&&line&i;
array arrfl{&&line&i} $ racfl1-racfl&&line&i;
arage{_n_}=age; arsex{_n_}=b2; arrac{_n_}=b4;
if last.ctrlnum then output;
run;

data sap1e&i;
merge sap1e&i sap1b&i;
by ctrlnum;
run;
```

The next step is to create flags to indicate whether an observation has been used for imputation. This is done in the usual manner of creating arrays, and we will again RETAIN the value of each element over iterations. Because we have to RETAIN values, it is necessary to create a new "temporary" race array (trcfl{}). This is because variables created in previous data steps cannot be retained in the present one. See the code for this.

```
/*  hot deck imputation for age, sex and race          */

data sap1f&i (drop=i age1-age&&line&i sex1-sex&&line&i
                         rac1-rac&&line&i agefl1-agefl&&line&i
                         sexfl1-sexfl&&line&i racfl1-racfl&&line&i
                         trcfl1-trcfl&&line&i);
attrib impage length=$3 impsex length=$1;
set sap1e&i;
retain agefl1-agefl&&line&i sexfl1-sexfl&&line&i
      trcfl1-trcfl&&line&i;
array arage{&&line&i} $ age1-age&&line&i;
array arsex{&&line&i} $ sex1-sex&&line&i;
array arrac{&&line&i} $ rac1-rac&&line&i;
array arafl{&&line&i} $ agefl1-agefl&&line&i;
array arsfl{&&line&i} $ sexfl1-sexfl&&line&i;
array arrfl{&&line&i} $ racfl1-racfl&&line&i;
array trcfl{&&line&i} $ trcfl1-trcfl&&line&i;

/*  create temporary array for race flags, since pre-   */
/*  viously created array cannot be retained            */

if _n_=1 then do;
  %do j=1 %to &&line&i;
    trcfl&j=racfl&j;
    %end;
  end;
```

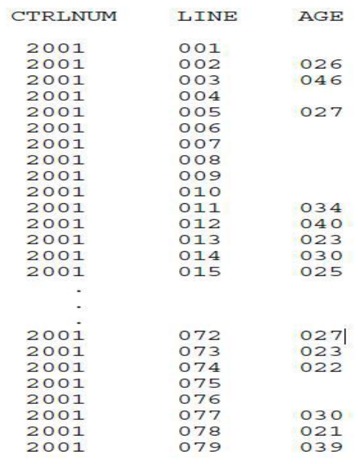
For the hot-deck imputation of any of the three demographic variables, if the variable is blank, we work backwards, and then forwards through the ctrlnum,for _n_>1, until we find an acceptable value. We work forward only, if the blank value occurs for the first person in the ctrlnum. We use the same "insertion code" for each of the three possible success situations: for _n_>1, working backwards, and then forwards; and for _n_=1,working forward only. For each observation, we are "looping" over the elements of the "retained" flag arrays, so we know if a previous "good" observation's value has already been used. If we don't have success in a ctrlnum, we code an out of range value. We repeat the same imputation process for each of the three demographic variables in each datastep iteration. The code for this is given below. The code is shown for the variable AGE that for the other two variables is similar.

```
/*  hot deck imputation for age  */
if age='' then do;
  if _n_>1 then do;
    do i=_n_-1 to 1 by -1;
      if arage{i}^='' and arafl{i}^='1' then do;
              arafl{i}='1'; impage=arage{i};
              goto nextsex;
              end;
      end;
    if impage='' then do;
      do i=_n_+1 to &&line&i;
        if arage{i}^='' and arafl{i}^='1' then do;
              arafl{i}='1'; impage=arage{i};
              goto nextsex;
              end;
        end;
      end;
    if impage='' then impage='999';
    end;
    /*  _n_=1  */
    else do;
      do i=_n_+1 to &&line&i;
        if arage{i}^='' and arafl{i}^='1' then do;
              arafl{i}='1'; impage=arage{i};
              goto nextsex;
              end;
        end;
      if impage='' then impage='999';
    end;
  end;
```

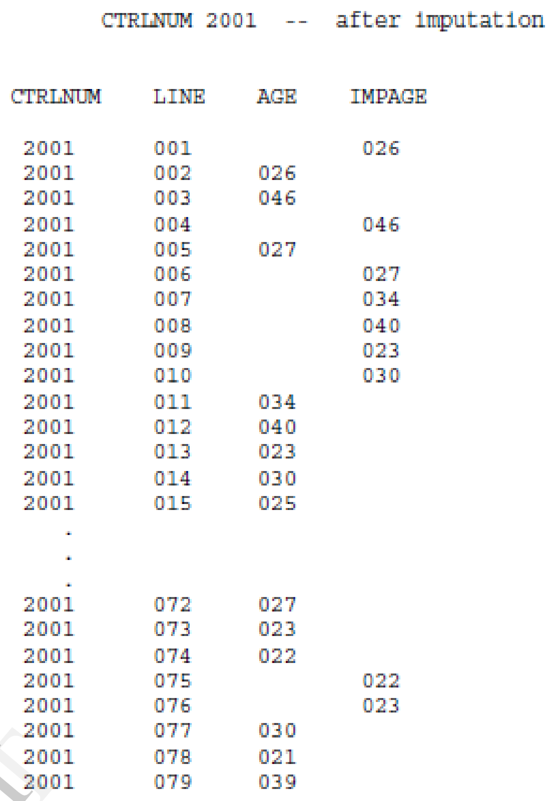
An example of how the imputation process works is shown below, for ctrlnum "2001". Some of the AGE data in this example is fabricated, to more fully illustrate how the algorithm executes. I show the demographic variable of AGE only, for simplicity of observing the working process. This example contains basically four sets of imputations, three of which occur within lines 001-014. The last set occurs for lines 075-076. The first imputation occurs for line 001.Since we cannot work backward, we must take the AGE value for the next line, 002. The AGE value for line 004 can be imputed using that for line 003, so we use it. We must impute for lines 006-010 using the backward forward approach. Line 005 is the only one of the previous observations in the ctrlnum which has a "good" value of AGE which has not already been used for imputation. Therefore, its value is the only one of the beforehand ones we can use. For the remainder of the lines in the sequence (007-010), we work successively forward, using the AGE values from lines 011-014, respectively. The last set of imputations for ctrlnum "2001" occurs for lines 075-076. These can be done using the simple "backward" approach. The AGE values for lines 074 and 073 are assigned to lines 075 and 076, respectively.

Table 2. Database Before and after Imputation

CTRLNUM 2001 – Before Imputation



CTRLNUM	LINE	AGE
2001	001	
2001	002	026
2001	003	046
2001	004	
2001	005	027
2001	006	
2001	007	
2001	008	
2001	009	
2001	010	
2001	011	034
2001	012	040
2001	013	023
2001	014	030
2001	015	025
.		
.		
.		
2001	072	027
2001	073	023
2001	074	022
2001	075	
2001	076	
2001	077	030
2001	078	021
2001	079	039



CTRLNUM 2001 -- after imputation

CTRLNUM	LINE	AGE	IMPAGE
2001	001		026
2001	002	026	
2001	003	046	
2001	004		046
2001	005	027	
2001	006		027
2001	007		034
2001	008		040
2001	009		023
2001	010		030
2001	011	034	
2001	012	040	
2001	013	023	
2001	014	030	
2001	015	025	
.			
.			
.			
2001	072	027	
2001	073	023	
2001	074	022	
2001	075		022
2001	076		023
2001	077	030	
2001	078	021	
2001	079	039	

2. CLIP4 Algorithm [1]

CLIP4 is a hybrid algorithm that combines ideas of two families of inductive ML algorithms, namely the rule algorithms and the decision tree algorithms. It uses rule generation schema from AQ algorithms. It also uses the tree growing technique that divides training data into subsets at each level of the tree, and pruning. To describe the algorithm we first introduce some notations. Let us denote the set of all training examples by S. The sets of the positive examples, SP, and SN the negative examples, SN, must satisfy these properties: Sp U Sn= S, SP ∩SN= Ǿ, SN# Ǿ, and SP # Ǿ. The positive examples are those that describe the class for which we currently generate rules, while the negative examples are all remaining examples.

The CLIP4 algorithm generates rules in the form of: IF(s1 ^ _ _ _ ^ sm)THEN class = classi; where si =[aj # vj] is a selector. We define SP and SN as matrices whose rows represent examples and columns correspond to attributes. Matrix of the positive examples is denoted as POS and the number of positive examples by NPOS, while matrix and the number of negative examples as NEG and NNEG, respectively.

Positive examples from the POS matrix are described by a set of values: posi[j] where j = 1 . . . K, is the column number, and i is the example number (row number in the POS matrix). The negative examples are described similarly by a set of negi[j] values. CLIP4 also uses binary matrices (BIN) that are composed of K columns, and filled with either 1 or 0 values. Each cell of the BIN matrix is denoted as bini[j], where i is a row number and j is a column number. These matrices are results of operations performed by CLIP4, and are modeled as the SC problem.

The positive data is partitioned into subsets of similar data in a decision-tree like manner. Each node of the tree represents one data subset. Each level of the tree is built using one negative example to find selectors that can distinguish between all positive and this particular negative example. The selectors are used to create new branches of the tree. During tree growing, pruning is used to eliminate noise from the data, to avoid its excessive growth, and to reduce execution time. A set of best terminal subsets (tree leaves) is selected using two criteria. First, large subsets are preferred over small ones since the rules generated from them can be "stronger" and more general, while all accepted subsets (between them) must cover the entire positive training data. Second, we want to use the completeness criterion. To that end, we first perform a back-projection of one of the selected positive data subsets using the entire negative data set, and then we convert the resulting matrix into a binary matrix and solve it using the SC method. The solution is used to generate a rule, and the process is repeated for every selected positive data subset.

A set of best rules is selected from all the generated rules. Rules that cover the most positive examples are chosen, which promotes selection of general rules. If there is a tie between "best rules" the shortest rule is chosen, i.e. the rule that uses minimal number of selectors.

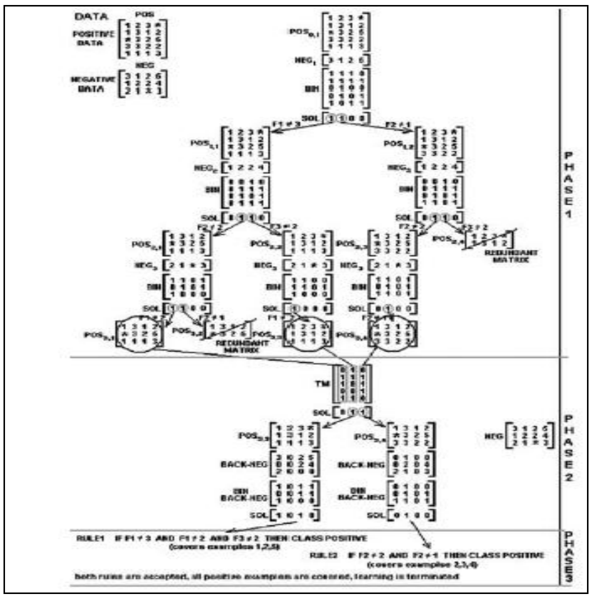

DATA
POSITIVE DATA
NEGATIVE DATA
PHASE 1
PHASE 2
PHASE 3
TM
REDUNDANT MATRIX
BACK-NEG
RULE1 IF F1≠3 AND F1≠2 AND F3≠2 THEN CLASS POSITIVE (covers examples 1,2,5)
RULE2 IF F2≠2 AND F2≠1 THEN CLASS POSITIVE (covers examples 2,3,4)
both rules are accepted, all positive examples are covered, learning is terminated

Fig.1. The low-level example of rule generation

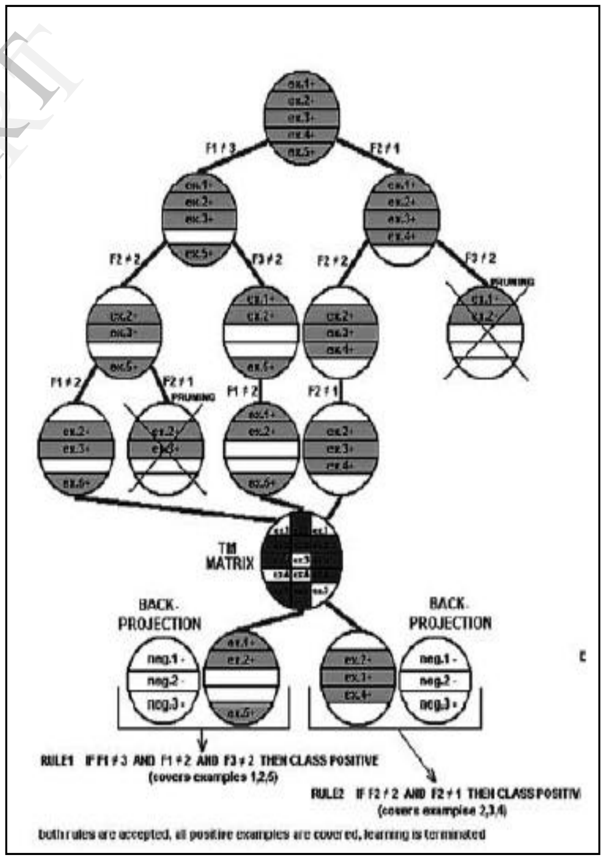

F1≠3
F2≠1
F2≠2
F3≠2
PRUNING
F1≠2
F2≠1
TM MATRIX
BACK-PROJECTION
BACK-PROJECTION
RULE1 IF F1≠3 AND F1≠2 AND F3≠2 THEN CLASS POSITIVE (covers examples 1,2,5)
RULE2 IF F2≠2 AND F2≠1 THEN CLASS POSITIV (covers examples 2,3,4)
both rules are accepted, all positive examples are covered, learning is terminated

Fig 2. The high-level example of rule generation by CLIP4

3. Comparative Study of Experimental Result

The The experiments were performed using different datasets and the different missing data imputation algorithms. The selected datasets originally do not contain missing values. The missing data were introduced artificially, using the MCAR model, into each of the datasets. As a result missing values were introduced into all attributes, including class attribute.
The missing data was artificially generated to enable verification of the quality of imputation, which was performed by comparing the imputed values with the original values. Each dataset is described by a set of characteristics. The selected datasets cover entire spectrum of values for each of the characteristics. Following graphs shows the performance of different imputation algorithm on different dataset for different characteristics.

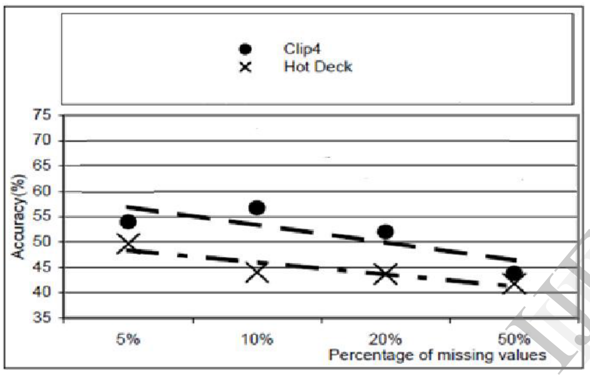


Fig.3. Accuracy against amount of missing values

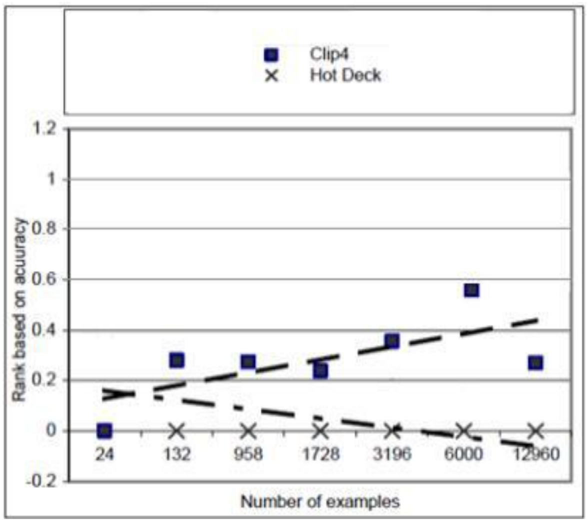


Fig.4. Normalized rank of the average imputation accuracy versus the number of example

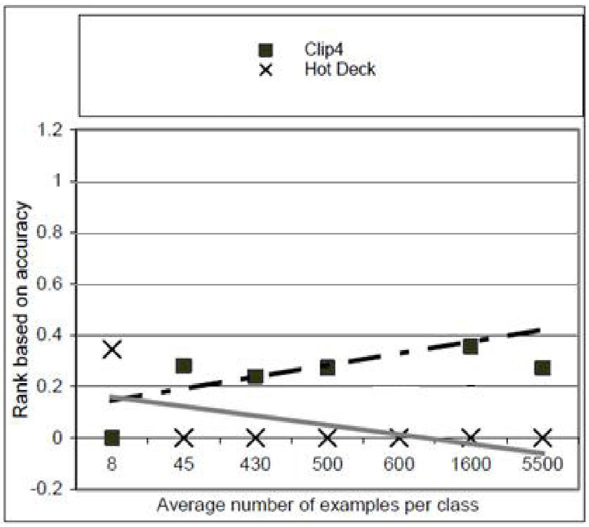


Fig.5. Normalized rank of the average imputation Accuracy vs. the average number of examples / class

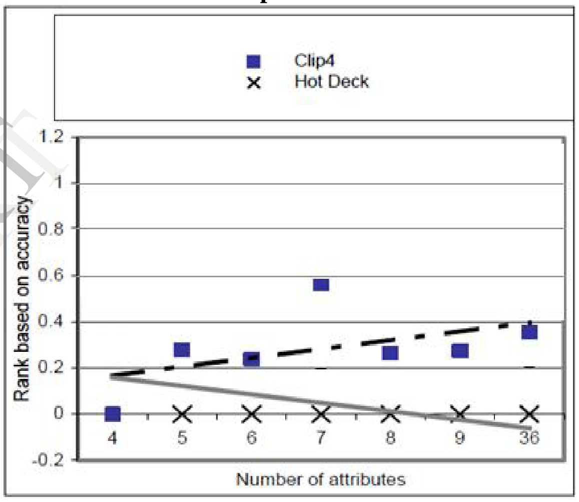


Fig.6. Normalized rank of the average imputation accuracy vs. number of attributes

accuracy versus the number of examples, Normalized Fig.3 to Fig.6 compare two missing data imputation methods based on the Accuracy against amount of missing values, Normalized rank of the average imputation rank of the average imputation accuracy vs. the average number of examples / class and Normalized rank of the average imputation accuracy vs. number of attributes. The normalized rank enables side by side comparison of the imputation methods, which is independent of the quality of the considered datasets.
Figure above shows that the unsupervised imputation methods are more stable comparing to the supervised methods. The main reason is that the supervised methods must have a training dataset of proper quality to develop an accurate model that is used to impute the missing information. On the other hand, the

unsupervised imputation methods are less sensitive to the amount of missing values. The results of the experiments show the superiority of supervised imputation methods. We also note that the unsupervised methods are more stable with respect to increasing amount of missing information.

Their performance decreases slower than the performance of the supervised methods. It can be expected that their performance may be better for databases with large amounts of missing values. The results also indicate that unsupervised imputation methods do not depend on the size of the input data, both in terms of the number of the attributes and the number of examples. On the other hand, the supervised imputation methods improve their performance with the increasing size of the input data.

4. Conclusion

Supervised method like CLIP4 perform better for databases with large amounts of missing values. Unsupervised imputation methods do not depend on the size of the input data, both in terms of the number of the attributes and the number of examples. On the other hand, the supervised imputation methods improve their performance with the increasing size of the input data.

Unsupervised imputation methods are more stable comparing to the supervised methods. The main reason is that the supervised methods must have a training dataset of proper quality to develop an accurate model that is used to impute the missing information. On the other hand, the unsupervised imputation methods are less sensitive to the amount of missing values. The results of the experiments show the superiority of Supervised imputation methods.

5. References

1) K..J.Cios, L.A. Kurgan, Hybrid Inductive Machine Learning: An Overview of CLIP Algorithms, In: L.C. Jain, and J. Kacprzyk, (Eds.), New Learning Paradigms in Soft Computing, pp. 276-322, Physica-Verlag (Springer), 2001
2) K. Lakshmi Narayan, S.A. Harp, and T. Samad, Imputation of Missing Data in Industrial Databases, Applied Intelligence, vol 11, pp. 259 – 275, 1999
3) K. Lakshmi Narayan, S.A. Harp, R. Goldman, and T. Samad, “Imputation of missing data Using machine learning techniques,” in Proceedings: Second International Conference On Knowledge Discovery and Data Mining, edited by Simoudis, Han and Fayyad, AAAI Press: Menlo Park, CA, pp. 140–145, 1996.
4) B.L. Ford, “An overview of hot-deck procedures,” In Incomplete Data in Sample Surveys, Volume 2, Theory and Bibliographies,edited by G.W. Madow, I. Olkin, and D.B. Rubin, Academic Press: New York, pp. 185–207, 1983.
5) Gustavo E. A. P. A. Batista and Maria Carolina “An Analysis of Four Missing Data Treatment Methods for Supervised Learning “, Monard University of S˜ao Paulo – USP
6) Alireza Farhangfar a , Lukasz Kurg an b , Witold Pedrycz c, “Experimental analysis of methods for imputation of missing values in Databases”, Proceedings of SPIE, 2004 - spie.org
7) K.J. Cios, W. Pedrycz, and R. Swiniarski, Data Mining Methods for Knowledge Discovery, Kluwer Academic Publishers, 1998