CHAPTER – V
EMPIRICAL ANALYSIS OF THE FRAMEWORK
5.1 Introduction

Mathematical Analysis of every module or function is very difficult, instead Empirical Analysis is easy. Empirical Analysis of an algorithm derives the practical time complexity. Asymptotic notations describe theoretical complexity of the algorithm, where by performing empirical analysis, one can easily estimate the amortized complexity of the performance of the algorithm.

The International Software Benchmarking Standards Group (ISBSG) database is no exception. It has a large fraction of missing data, in some variables more than 40 percent. There are several reasons why observations may have missing values. High data collection cost may cause missing values. The cost of gathering and reporting data from software projects is non-negligible. For example, it is more difficult, and, therefore, costly to collect data on Interfaces and Effort than on Users, Sites and Modules. Therefore, we expected, and did indeed find, that there are more missing values in Interfaces and Effort than in Users, Sites, and Modules.

Another reason for missing values is that some values are so called wild values. A wild value is a value we know is untrue.

5.2 Selection and Description of Datasets

As a standard of benchmarking the sample of the dataset is chosen for analysis. The data is sampled tuple wise and attribute wise.
Sampled Attributes and their Existence in the Sample Data are shown in the following illustrations: [A]

### R_cap-color

<table>
<thead>
<tr>
<th>Category</th>
<th>Total Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>48 (23.6%)</td>
</tr>
<tr>
<td>n</td>
<td>105 (51.7%)</td>
</tr>
<tr>
<td>w</td>
<td>50 (24.6%)</td>
</tr>
</tbody>
</table>

### R_cap-shape

<table>
<thead>
<tr>
<th>Category</th>
<th>Total Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>148 (72.9%)</td>
</tr>
<tr>
<td>f</td>
<td>55 (27.1%)</td>
</tr>
</tbody>
</table>

### cap-surface

<table>
<thead>
<tr>
<th>Category</th>
<th>Total Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>48 (23.6%)</td>
</tr>
<tr>
<td>s</td>
<td>148 (72.9%)</td>
</tr>
<tr>
<td>y</td>
<td>7 (3.4%)</td>
</tr>
</tbody>
</table>

### gill-attachment

<table>
<thead>
<tr>
<th>Category</th>
<th>Total Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>96 (47.3%)</td>
</tr>
<tr>
<td>f</td>
<td>107 (52.7%)</td>
</tr>
</tbody>
</table>

### gill-spacing

<table>
<thead>
<tr>
<th>Category</th>
<th>Total Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>106 (52.2%)</td>
</tr>
<tr>
<td>w</td>
<td>97 (47.8%)</td>
</tr>
</tbody>
</table>

### gill-size

<table>
<thead>
<tr>
<th>Category</th>
<th>Total Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>200 (98.5%)</td>
</tr>
<tr>
<td>n</td>
<td>3 (1.5%)</td>
</tr>
</tbody>
</table>
Sampled Attributes and their Existence in the Sample Data are shown in the following illustrations: [B]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Category</th>
<th>Total Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R_gill-color</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>32 (15.8%)</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>32 (15.8%)</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>32 (15.8%)</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>33 (16.3%)</td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>42 (20.7%)</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>32 (15.8%)</td>
<td></td>
</tr>
</tbody>
</table>

| **stalk-surface-above-ring**     |          |              |
| f                                | 1 (0.5%)  |
| k                                | 48 (23.6%) |
| s                                | 146 (71.9%) |
| y                                | 8 (3.9%)  |

| **stalk-surface-below-ring**     |          |              |
| f                                | 3 (1.5%)  |
| k                                | 48 (23.6%) |
| s                                | 144 (70.9%) |
| y                                | 8 (3.9%)  |

| **R_stalk-color-above-ring**     |          |              |
| n                                | 8 (3.9%)  |
| o                                | 96 (47.3%) |
| w                                | 99 (48.3%) |

| **R_stalk-color-below-ring**     |          |              |
| n                                | 9 (4.4%)  |
| o                                | 96 (47.3%) |
| w                                | 98 (48.3%) |
Sampled Attributes and their Existence in the Sample Data are shown in the following illustrations: [C]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Category</th>
<th>Total Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_veil-color</td>
<td>n</td>
<td>48 (23.6 %)</td>
</tr>
<tr>
<td></td>
<td>o</td>
<td>48 (23.6 %)</td>
</tr>
<tr>
<td></td>
<td>w</td>
<td>107 (52.7 %)</td>
</tr>
<tr>
<td>R_ring-number</td>
<td>o</td>
<td>99 (48.8 %)</td>
</tr>
<tr>
<td></td>
<td>t</td>
<td>104 (51.2 %)</td>
</tr>
<tr>
<td>R_ring-type</td>
<td>e</td>
<td>1 (0.5 %)</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td>2 (1.0 %)</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>200 (98.5 %)</td>
</tr>
<tr>
<td>R_spore-print-color</td>
<td>b</td>
<td>24 (11.8 %)</td>
</tr>
<tr>
<td></td>
<td>h</td>
<td>2 (1.0 %)</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>24 (11.8 %)</td>
</tr>
<tr>
<td></td>
<td>o</td>
<td>24 (11.8 %)</td>
</tr>
<tr>
<td></td>
<td>w</td>
<td>105 (51.7 %)</td>
</tr>
<tr>
<td></td>
<td>y</td>
<td>24 (11.8 %)</td>
</tr>
</tbody>
</table>
Sampled Attributes and their Existence in the Sample Data are shown in the following illustrations: [D]

<table>
<thead>
<tr>
<th>Category</th>
<th>Total Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>48 (23.6%)</td>
</tr>
<tr>
<td>n</td>
<td>48 (23.6%)</td>
</tr>
<tr>
<td>s</td>
<td>49 (23.6%)</td>
</tr>
<tr>
<td>v</td>
<td>49 (24.1%)</td>
</tr>
<tr>
<td>y</td>
<td>10 (4.9%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Total Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>6 (3.0%)</td>
</tr>
<tr>
<td>g</td>
<td>96 (47.3%)</td>
</tr>
<tr>
<td>l</td>
<td>97 (47.8%)</td>
</tr>
<tr>
<td>p</td>
<td>4 (2.0%)</td>
</tr>
</tbody>
</table>

The sampling mechanism is implemented from the Orange Tool. This tool implements procedures of machine learning for sampling data. *Example sampling* is one of the basic procedures in machine learning. Example Sampling is a variant of Random Sampling. If for nothing else, everybody needs to split dataset into training and testing examples. The example sampling is a mechanism where the attributes are selected and examples of the attributes, about their existence in the sample data are presented. User is allowed to select the attributes based on the attribute's qualities.

5.2.1. Using Orange for Sampling

It is easy to select a subset of examples in Orange. The key idea is the use of indices: first construct a list of indices, one corresponding to each example. Then we can select examples by indices, say take all examples with index 3. Or with index other than 3. It is obvious that this is useful for many typical setups, such as 70-30 splits or cross-validation. Orange provides methods for making
such selections, such as `ExampleTable's select method. And, of course, it provides methods for constructing indices for different kinds of splits. For instance, for the most common used sampling method, cross-validation, the Orange's class `MakeRandomIndicesCV` prepares a list of indices that assign a fold to each example.

Sampling with Orange is visually mapped as shown below. Several widgets are selected and connected to develop the samples of datasets. The widgets (any interactive graphical object in a document) used for sampling in this research work are: File, Select-Data, Data-Sampler, and Attribute-Statistics.

Fig: 5.1 The Orange data sampler illustrating sampling mechanism.
The data sampler presets with various options to tune the sampling mechanism.

Fig:5.2 Presets of Data Sampler
5.3 Illustrations on Datasets

The following illustrations describe the distribution of data with various types of values in the attributes.

Fig: 5.3 Visualization of Sampled Data using Parallel Coordinates
Fig: 5.4 Visualization of Sampled Data using Linear Projection
5.4 Various Exercises on the Framework

Experimental Evaluation of Algorithms

In this work a Java-based Efficient Imputation System that can be easily deployed on any Java Virtual Machine (JVM) platform and gathered datasets from several online data repositories as the data source. The experiments tested the viability of the three major components in this work, namely, our new term weighting scheme and our new sentence-modeling scheme.

Throughout, each successive design proposal is evaluated after describing that proposal in order to isolate its performance impact and motivate further refinements. The total work is practically carried out using Java SDK on mushroom, tic-tac-toe, Mammographic and Hepatitis datasets. The previous works of the imputation algorithms are experimented and tested in C/C++ on Linux, their performance is compared with proposed experiment.

5.4.1 Data Selection:

To meet the experimental feasibility of the work, the imputation procedure has been carried out as follows: The database consists of entities with tuples, where some of the tuples contain the missing attribute values. The tuple to be imputed by the tool developed in this work is given as a query tuple and placed in a separate file. The whole data (original data) that contains the imputed tuple with all other tuples is given as training data set to the tool. The query tuple is stored in query.txt and whole original data as dataset.txt. The mushroom data set consisting of 23 attributes, 8124 tuples inclusive of missing attributes is pre-processed by the tool for about 10 minutes and processed for imputation on less then one second.

Data set

The data set for this assignment consists of descriptions of mushrooms drawn from the Audubon Society Field Guide. Each mushroom is described by 22
physical characteristics (e.g., scaly, yellow) and also by whether it is poisonous or edible.

Information about the data base can be obtained at


The file agaricus-lepoita.names contains information about the data base and the 22 attributes, and the file agaricus-lepiota.data contains the actual data. The data set contains some records with missing values (indicated by question marks). These items are removed from the data set, and split the data into a training set and a test set. You should retrieve the data from the site:


The first column of the data contains the class label, “p” for poisonous and “e” for edible.

Various sites have been visited to collect the bench mark data to test the propensity of the tool. Some of the links are given below:

(a) http://www.nd.edu/~oss/Data/data.html
(b) http://kdd.ics.uci.edu/summary.data.type.html
(c) http://data.eol.ucar.edu/codiac/ds Proj?Operational
(d) http://www.lshtm.ac.uk/msu/missingdata/biblio.html

5.4.2 Application Development

A swing based application is developed to implement complete imputation model of the work. Dataset are read into the application and represented by the collection utilities of JDK. This is to avoid the time consumption during disk input output and to amortize the overhead of time complexity in overall algorithmic implementation. The application consumes considerable amount of time in preprocessing. The preprocessing stage includes, converting the file data into the collection utility framework, selection of
attributes relevant for the imputation, and various samples of attributes for checking the highest probability of the value that is to be imputed. While converting the file data into the collection utility framework of the platform, the digest values are generated and stored in the repository. The tool provides the walkthrough into the corpus available. The missing value digest, which is a collection of selected characteristic weights of the tuple are represented as corpus. This helps in the algorithm for searching the more relevant tuple that meets the needs of finding the plausible value for imputation. The tool EfficientImputer is developed in the present process of ongoing research work. The total experiment is carried our in three stages. The application development has strategic stages [78]. Java contributes excellent API in its core as well Developing Code Patterns also have been used as per [1,54,103].

5.4.3 Attribute Selection Analysis and Imputation

For this analysis 3 experiments have been conducted. All the experiment listed below explains the various tasks of the imputation tool illustratively. First all the list of attributes are considered for generation of digest values and then allowed for selection. Subsequently, the numbers of attributes that are going to participate actively are not chosen from the user input. Without the numbers of attributes the scope of generating the digest values and storing the corpus becomes huge and the search space for the prime imputation mechanism becomes vast leading to insignificant loss of time. If the numbers of attributes that are going to participate actively are chosen from the user, then the search space consideration is limited to a feasible extent.

Various samples of the attributes are selection as a part of attribute selection, stating the possibilistic idea to obtain the plausible value for imputation. Even though the attribute selection is manually taken care, it is adjudged by the relevant knowledge of the experimenter about the tuple and the domain of the selected dataset.
**Experiment 1:**

Base Test Case:

<table>
<thead>
<tr>
<th>Original Data</th>
<th>8124 tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(23 species of gilled mushrooms in the Agaricus and Lepiota Family)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query Data</th>
<th>e,x,y,u,f,n,f,c,n,w,e,?, s,f,w, w,p, w,o,f,h,y,d</th>
</tr>
</thead>
</table>

| Scope Attributes Considered | 12 |

<table>
<thead>
<tr>
<th>Selections</th>
<th>3 (Three Attribute Selections)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1) 9, 10, 12</td>
</tr>
<tr>
<td></td>
<td>(2) 10, 11, 12</td>
</tr>
<tr>
<td></td>
<td>(3) 1, 5, 10, 12</td>
</tr>
</tbody>
</table>

**Results:**

Position of ? is 11th in the query tuple.

Attribute Selection number (2) is ignored, since it contains (11th) attribute reference to the ? in the query tuple.

Plausible value is searched using (1) and (3) attribute selections.

Tuple #12 contains plausible value.

Value imputed is 'c' at '?'

*The tool developed is illustrated briefly with all screens and operations*

![Fig:5.5 A Typical Screen containing inputs of all parameters of EfficientImputer](image)

146
Fig: 5.6 EfficientImputer: Loading and Reading Datasets and Query

Fig: 5.7 EfficientImputer: Attribute Selection
**Experiment 2:**

**Base Test Case:**

| Original Data | 6 tuples  
(6 varieties of Cyclo-Hexane Compound) |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Data</td>
<td>b,a,c,?,d,a</td>
</tr>
<tr>
<td>Scope Attributes Considered</td>
<td>6</td>
</tr>
<tr>
<td>Selections</td>
<td>3 (Three Attribute Selections)</td>
</tr>
<tr>
<td></td>
<td>(1) 0, 1, 2</td>
</tr>
<tr>
<td></td>
<td>(2) 0, 3, 4, 5</td>
</tr>
<tr>
<td></td>
<td>(3) 2, 3, 4</td>
</tr>
<tr>
<td>Results:</td>
<td>Position of ? is 3rd in the query tuple</td>
</tr>
<tr>
<td></td>
<td>Attribute Selection number (2),(3) are ignored, since it contains (3rd) attribute reference to the ? in the query tuple.</td>
</tr>
<tr>
<td></td>
<td>Plausible value is searched using (1) attribute selection.</td>
</tr>
<tr>
<td></td>
<td>Tuple #3 contains plausible value.</td>
</tr>
<tr>
<td></td>
<td>Value imputed is 'c' at '?'</td>
</tr>
</tbody>
</table>
The corpus is generated for all the data in the experiment. The Characteristic Weights and Missing Value Digest of the tuples participate in generating the corpus required for the experiment. The corpus is the basic search space to find the plausible value for the imputation. Characteristic Weights are described using Digest Values (a variant of hash values), bitmapped equivalent values of the data set. Several items in the data set are identified and the radix of the data set is calculated. Based on radix an $n$-bit string is assumed to possess the binary values indicating the each item of the dataset uniquely. $n$ is calculated as follows:

Let $r$ be radix

$$n = \sqrt{r}$$

If $r$ is not a perfect square, then $n$ is incremented by 1 of its integer part and $n$ is assumed as int($n$)+1. Where int($n$)+1 bits are used to represent the bit string to assume the digest value of the item in the dataset.

In the experiment the data set used contains 23 attributes including the p or e attribute which is the first attribute of the tuples.

No. of items = 23 i.e., $r = 23$

$$n = \sqrt{r}$$

$$n = 4.7958315233127195415974380641627$$

No. of bits in the bit-string is int($n$) + 1 i.e., $4 + 1 = 5$

5 bits are used to form a bit-string that can accommodate the digest value of all the items in the current data set.
Following is the verbose output of the experiment conducted

0,1,2
0,3,4,5
2,3,4
Selected Indexes 3
0**
1**
2**
LIST OF SELECTIONS
3
---
0,1,2
Query digest to locate in [CW] MVD Corpus
00010:00001:00011:

* * * * *
00010:00001:00011:
00010:00011:00001:
00010:00010:00101:
00010:00001:00011:
Located at Record #3

Corpus for Attribute Selections shown in the Verbose output of the experiment

No. of attributes considered for experiment : 6
No. of Attribute Selections: 3
  Attribute Selection-1: 0,1,2
  Attribute Selection-2: 0,3,4,5
  Attribute Selection-3: 2,3,4

Selection : 1
00001:00010:00011:
00010:00011:00001:
00010:00010:00101:
00010:00001:00011:
00011:00010:00001:
00110:00011:00001:
***

Selection : 2
00001:00100:00101:00110:
00010:00011:00100:00001:
00001:00100:00011:00110:
00010:00011:00100:00001:
00111:00100:00101:00110:
00110:00011:00101:00100:
***

Selection : 3
00011:00100:00101:
00001:00011:00100:
00101:00100:00001:
00010:00011:00100:
00011:00010:00101:
00001:00011:00101:
***
In the above verbose output, the selection-1, selection-2, selection-3 are the attribute selection sequences that are used to find the plausible value for imputation. As it is very tedious to locate the plausible value in the entire database, according to users' certain guesses or based on the domain knowledge (background information) of the user, the attribute selection sequences can be generated. This is a sampling activity where the user has to be aware of the attributes and their positions which are having higher probability of having the plausible values for the imputation. Such attributes are selected and combined with various other attributes as various types of attribute selections, for example in the above verbose output, the three attribute selection sequences are 0,1,2; 0,3,4,5; 2,3,4. The 0,1,2 indicates the first attribute selection that contains the combination of attributes 0, 1, 2. (The position of the first attribute is identified as to be at the position 0). Similarly, 0,3,4,5 means the attribute selection later considered, that contains the combination of attributes 0,3,4,5. This illustrates that the attribute considered for the attribute selection not necessarily be in the sequence, they may be selected in assorted style also, except that all the item positions are organized in the order. 0, 3, 4, 5 or 2, 3, 4 are valid attribute selections, but 0,3,2,4 is not a valid selection. 0,3,2,4 is also an assorted selection of attributes but not organized in the ascending order.
### Experiment 3:

**Base Test Case:**

| Original Data | 6 tuples  
|              | (6 varieties of Mammographic Mass Data) |
| Query Data   | 4, 43, 1, 1, ?, 1 |
| Scope Attributes Considered | 6 |
| Selections   | 2 (Two Attribute Selections)  
|              | (1) 0, 2, 3  
|              | (2) 0, 3, 4, 5 |
| Results:     | Position of ? is 4th in the query tuple  
|              | Attribute Selection number (2) is ignored, since it contains no attribute reference to the ? in the query tuple.  
|              | Plausible value is searched using (1) attribute selection.  
|              | Tuple #4 contains plausible value.  
|              | Value imputed is '3' at '?' |

The search and replace procedure of the missing value with plausible value is carried out with specially developed API based code. Many fragments of code are written to represent the logical idea of the problem and it's solving procedure. As seen in the previous example experiment, the data set is converted into digest data set, where all the digest values of the items in the data set are present. Digest values are not necessarily compressed bitmap representation; rather, these values especially exhibit the characteristic features of the items in the data set. Digest values are generated for all the items of the data set and further they are organized in various sequences of bit strings based on the attribute selection sequences.

The attribute selection sequence governs the generation of the characteristic weights of items in each tuple, subsequently combining all the characteristic weights of the items i.e., digest of the tuple based on the attribute selection, are collected as missing value digest.
The searching part has been coded diligently to enable the search choose sequential approach for small data sets and faster search with pattern matching in the case of huge data sets, thus by regulating the time complexity.

```java
String getBits(String s)
{
    String bits;
    Character cs = new Character(s.charAt(0));
    switch(cs.charValue())
    {
        case 'a': bits="00001"; break;
        case 'b': bits="00010"; break;
        case 'c': bits="00011"; break;
        case 'd': bits="00100"; break;
        case 'e': bits="00101"; break;
        case 'f': bits="00110"; break;
        case 'g': bits="00111"; break;
        case 'h': bits="01000"; break;
        case 'i': bits="01001"; break;
        case 'j': bits="01010"; break;
        case 'k': bits="01011"; break;
        case 'l': bits="01100"; break;
        case 'm': bits="01101"; break;
        case 'n': bits="01110"; break;
        case 'o': bits="01111"; break;
        case 'p': bits="10000"; break;
        case 'q': bits="10001"; break;
        case 'r': bits="10010"; break;
        case 's': bits="10011"; break;
        case 't': bits="10100"; break;
        case 'u': bits="10101"; break;
        case 'v': bits="10110"; break;
        case 'w': bits="10111"; break;
        case 'x': bits="11000"; break;
        case 'y': bits="11001"; break;
        case 'z': bits="11010"; break;
        case '0': bits="11011"; break;
        case '1': bits="11100"; break;
        case '2': bits="11101"; break;
        case '3': bits="11110"; break;
        case '4': bits="11111"; break;
        case '5': bits="100000"; break;
        case '6': bits="100001"; break;
        case '7': bits="100010"; break;
        case '8': bits="100011"; break;
        case '9': bits="100100"; break;
        case 'A': bits="100101"; break;
        case 'B': bits="101010"; break;
        case 'C': bits="101011"; break;
        case 'D': bits="101100"; break;
        case 'E': bits="101101"; break;
        case 'F': bits="101110"; break;
        case 'G': bits="101111"; break;
        case 'H': bits="110000"; break;
        case 'I': bits="110001"; break;
        case 'J': bits="110010"; break;
        case 'K': bits="110011"; break;
        case 'L': bits="110100"; break;
        case 'M': bits="110101"; break;
        case 'N': bits="110110"; break;
        case 'O': bits="110111"; break;
        case 'P': bits="111000"; break;
        case 'Q': bits="111001"; break;
        case 'R': bits="111010"; break;
        case 'S': bits="111011"; break;
        case 'T': bits="111100"; break;
        case 'U': bits="111101"; break;
        case 'V': bits="111110"; break;
        case 'W': bits="111111"; break;
    }
}
```
```java
    case 'S': bits="110111"; break;
    case 'T': bits="111000"; break;
    case 'U': bits="111001"; break;
    case 'V': bits="111010"; break;
    case 'W': bits="111011"; break;
    case 'X': bits="111100"; break;
    case 'Y': bits="111101"; break;
    case 'Z': bits="111110"; break;
    default: bits="00000";
    }
    return bits;
}
```

The above code fragment is supported by `getDigest("a")`

```java
    String getDigest(String line)
    {
        int i=0;
        String d="";
        String r[] = line.split(",");
        for(i=0;i<r.length;i++)
            d+=getBits(r[i]);
        return d;
    }
```

The generated digest are stored in Vector collections and further are used in the process of locating pattern of query in data sets. The Characteristic Weights are stored in Vector collection, which contains digest values of attributes that belong to the selected attribute selections only. The Vector that stores Characteristic Weights is an array that contains number of elements equal to the number of attribute selections. These attribute selections are considered by imputer during the input provided by the user at the time of selecting query trigger. When the query trigger button is selected, immediate input to be given is the position of attribute in query tuple that contains missing value i.e., ?. As soon as the position is input, the attribute selections that contain the position as the subset will be simply ignored for search space. Only the digest values of those attribute selections that does not contain the position of the missing value in the query tuple are allowed search for the plausible value.
public void locateQueryDigestInCorpus(String strQueryDigest, int idx) {
    int i=0;
    boolean located=false;
    System.out.println("Query digest to locate in [CW MVD Corpus");
    System.out.println(strQueryDigest);
    System.out.println("* ****");
    jtaVerbose.append("Query digest to locate in [CW MVD Corpus
")
    jtaVerbose.append(strQueryDigest+
"*
*
*
*

for(i=0;i<
CWeights[idx].size();i++)
{
    String vStr = vCWeights[idx].get(i).toStringO;
    jtaVerbose.append(vStr+
"
"+
"
")
    System.out.println(vStr);
    if(vStr.equals(strQueryDigest)){ located=true; break; } }
    if(located){
        jtaVerbose.append("Located at Record "+i+
"
"+
"
")
        System.out.println("Located at Record "+i);
        jtaQuery.append(" Record "+i+" is found with plausible value
")
        jtaQuery.append(" Query Tuple ":rd.mquery+
"
"
")
        jtaQuery.append(" Record :rd.msdata.get(i).toStringO+
"
"
")
        String strImputed=replaceAttribute(rd.mquery, rd.msdata.get(i).toStringO);
        jtaQuery.append(" Imputed Query Tuple;"+strImputed+
"
")
    }
    else
    {
        jtaVerbose.append("Record Not Located imputation Not Possible... Trying with another selection");
        System.out.println("Record Not Located Imputation Not Possible... Trying with another selection");
    }
}

The code fragment replaceAttribute, shown below illustrate the procedure of finding the position of missing value, identifying the plausible value and replacing the missing value with the plausible value.

public String replaceAttribute(String q, String t) {
    String res="";
    int pos = q.indexOf("?");
    char s = t.charAt(pos);
    StringBuffer qs = new StringBuffer(q);
    qs.replace(pos,pos+1,s+"";
    res = qs.toStringO;
    return res;
}

5.4.4 Type of data in the attributes

Attributes that are selected for the process are alphanumeric in type of one byte width. If the value is more than one byte width, then equivalent bitmapped values are generated for data set in the attribute. The bitmapped values can be
assumed for any type of attributes. For example if data set in the attribute are {positive, negative}, then the equivalent bits for values can be taken and digest value can be generated.

\{\text{positive, negative}\} = \{0, 1\}

Data set related to Hepatitis Domain is another example, where the attribute class is considered for processing. As a number of values in the attribute class is only two a binary digit can be considered for generating a digest value.

1. Class: DIE, LIVE
2. AGE: 10, 20, 30, 40, 50, 60, 70, 80
3. SEX: male, female
4. STEROID: no, yes
5. ANTIVIRALS: no, yes
6. FATIGUE: no, yes
7. MALAISE: no, yes
8. ANOREXIA: no, yes
9. ALBUMIN: 2.1, 3.0, 3.8, 4.5, 5.0, 6.0
10. PROTIME: 10, 20, 30, 40, 50, 60, 70, 80, 90
11. HISTOLOGY: no, yes

\{\text{DIE, LIVE}\} = \{0, 1\}

Apart from the conversion of the one-byte alphanumeric data into bits, the string or word data can be converted into bits (digest values) by considering the bitmap. A bit map is designed dynamically and a bitset are constructed to design the digest value. By using the basic structure of the data set i.e., attributes information it will be easy to generate the bit map and the equivalent bitset to generate the digest value of the attribute and further it can be merged with the whole digest value of the tuple.
5.5 Case Wise Analysis

The core parameters of the experiment are number of records, number of attributes, and position of the attribute that contains the missing value in the query tuple. The variable parameters of the experiment that change, to influence the efficiency and time for finding out the plausible value are, number of attributes that are considered for the whole imputation process, number of attribute selection sequence samples, processing time (in msec.).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8124</td>
<td>16</td>
<td>11</td>
<td>13</td>
<td>4</td>
<td>0.45</td>
</tr>
<tr>
<td>8124</td>
<td>16</td>
<td>11</td>
<td>13</td>
<td>3</td>
<td>0.35</td>
</tr>
<tr>
<td>8124</td>
<td>16</td>
<td>11</td>
<td>13</td>
<td>2</td>
<td>0.3</td>
</tr>
<tr>
<td>8124</td>
<td>15</td>
<td>11</td>
<td>13</td>
<td>4</td>
<td>0.35</td>
</tr>
<tr>
<td>8124</td>
<td>15</td>
<td>11</td>
<td>13</td>
<td>3</td>
<td>0.3</td>
</tr>
<tr>
<td>8124</td>
<td>15</td>
<td>11</td>
<td>13</td>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td>8124</td>
<td>14</td>
<td>11</td>
<td>13</td>
<td>3</td>
<td>0.3</td>
</tr>
<tr>
<td>8124</td>
<td>13</td>
<td>11</td>
<td>13</td>
<td>2</td>
<td>0.175</td>
</tr>
</tbody>
</table>

Table: 5.1 Table showing the Parameters for various experiments using EfficientImputer

The performance of the imputer changes according to the core input and the variable inputs provided during the experiment. The core input remain constant as the data set is not changed in the experiment, where the variable inputs change, as the thirst of find the plausible value as quickly as possible increases. The process of finding of the plausible value is narrowed based on the number of observations made in the experiment. However, this requires the knowledge about the domain of the data set or the higher probabilistic rates of guess. As the guess is not the measure to consider the attribute selections, various possibilities of attribute selection sequences are made. By considering the combinations of the items in the tuples of the data set, various attribute selection sequences are constructed.
The above comparative chart is illustrating the curve with proportionate increase of time based on the increase of the number of attribute selections. For the above; the number of tuples selected is 8124, Number of attributes that are selected for the experiment is 16, and Position of the missing value in the query tuple is 11th position, Number of attributes that are participating in the current instance of experiment is 13, number of attribute selection samples and the time taken to process the imputation are graphed.

The above comparative chart is illustrating the curve with proportionate increase of time based on the increase of the number of attribute selections. For the above; the number of tuples selected is 8124, Number of attributes that are...
selected for the experiment is 15, and Position of the missing value in the query tuple is 11th position, Number of attributes that are participating in the current instance of experiment is 13, number of attribute selection samples and the time taken to process the imputation are graphed.

![Comparative Chart](image)

Fig: 5.11 A Comparative Chart showing No. of Attribute Selection Samples vs. Time [C]

The above comparative chart is illustrating the curve with proportionate increase of time based on the increase of the number of attribute selections. For the above; the number of tuples selected in 8124, Number of attributes that are selected for the experiment is 14 and 13, and the Position of the missing value in the query tuple is 11th position, Number of attributes that are participating in the current instance of experiment is 13, number of attribute selection samples and the time taken to process the imputation are graphed.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8124</td>
<td>20</td>
<td>11</td>
<td>13</td>
<td>4</td>
<td>0.5625</td>
</tr>
<tr>
<td>8124</td>
<td>20</td>
<td>11</td>
<td>13</td>
<td>3</td>
<td>0.5125</td>
</tr>
<tr>
<td>8124</td>
<td>20</td>
<td>11</td>
<td>13</td>
<td>2</td>
<td>0.4625</td>
</tr>
<tr>
<td>8124</td>
<td>18</td>
<td>11</td>
<td>13</td>
<td>4</td>
<td>0.4825</td>
</tr>
<tr>
<td>8124</td>
<td>16</td>
<td>11</td>
<td>13</td>
<td>3</td>
<td>0.3225</td>
</tr>
<tr>
<td>8124</td>
<td>15</td>
<td>11</td>
<td>13</td>
<td>2</td>
<td>0.3</td>
</tr>
<tr>
<td>8124</td>
<td>14</td>
<td>11</td>
<td>13</td>
<td>3</td>
<td>0.2825</td>
</tr>
<tr>
<td>8124</td>
<td>13</td>
<td>11</td>
<td>13</td>
<td>2</td>
<td>0.2325</td>
</tr>
</tbody>
</table>

Table: 5.2 Table showing the Parameters for various experiments using EfficientImputer
The above comparative chart is illustrating the curve with proportionate increase of time based on the increase of the number of attribute selections. For the above; the number of tuples selected in 8124, Number of attributes that are selected for the experiment is 20, 18, 16, 15, 14 and 13, and the Position of the missing value in the query tuple is 11\textsuperscript{th} position, Number of attributes that are participating in the current instance of experiment is 13, number of attribute selection samples and the time taken to process the imputation are graphed.

**Observation from Graphs**

All the above graphs show the two parallel or disjoint curves. This indicates, there is a proportional difference between the number of attribute selection samples and the time. From the above observation it may be concluded that “The rate of increase of attribute selection samples influences the increase of time”.

**5.6 Evaluation**

In this section, we report a systematic empirical study using real data sets and synthetic data sets. All the experiments were conducted on a PC computer running the Microsoft Windows XP Professional Edition operating
system, with a 3.0 GHz Pentium 4 CPU, 1.0 GB main memory, and a 160 GB hard disk. Our algorithms were implemented in Java. By default, our method was implemented.

5.6.1 Sampling

Sampling methods are classified as either probability or nonprobability. In probability samples, each number of the population has a known non-zero probability of being selected. Probability methods include random sampling, systematic sampling, and stratified sampling. In nonprobability sampling, numbers are selected from the population in some nonrandom manner. These include convenience sampling, judgment sampling, quota sampling, and snowball sampling.

An optimistic method of sampling is implemented in order to select the sample tuples that contains all tuples having the missing values and adequate number of tuples that propose the suggestion for the plausible values; however they are algorithmically matched by the characteristic weights.

5.6.2 Attribute Selection

Selecting the attributes into the attribute sets; the attribute set plays a very important role, in the overall framework. As per the definition given to the attribute set, the set of attributes which are most likely posses no missing value are selected as members of the attribute set, such sets having existent attributes are built. Number of iterations have been performed to understand the attribute as an existent attribute. In the strict case, the attribute even having a single missing value will become a non-existent attribute. The existent-attributes are grouped together into several groups based on the information provided by the general characteristics and background knowledge about the attributes. A simple iterative heuristic algorithm is implemented to perform attribute selection successful.
5.6.3 Generating Characteristic Weights

Generating Characteristic Weights is process of developing unique digest values using a hashing algorithm. The hash algorithm generates the digest value for all the tuples that does not contain missing values. The digest values are generated for each set of attributes.

5.6.4 Implementing Search

The record where the missing value is available is identified. In order to start imputation, the possible set of attributes possible from the record is prepared. For all the attribute sets, the characteristic weight (hash value) is generated and the CW is searched from the MVD of that attribute set from the characteristic corpus. The CWs of the attribute set are arranged in a list with sorted order. The CW is searched on the sorted data using a binary search technique and the corresponding record position is located.

5.6.5 Framework Reliability

"Reliability is the consistency of a set of measurements or measuring instrument, often used to describe a test. This can either be whether the measurements of the same instrument give or are likely to give the same measurement (test-retest), or in the case of more subjective instruments ".

The components and their integration are optimistic. The connection between the components is reliable with regard to the transactions between them. The reliability in this research work is deposited in two fold.

First, while maintaining the Characteristic Corpus, The characteristic corpus is a collection of CWs of various categories. For the generation of characteristic weights it is compulsory that to select the set of attributes. If the CW is calculated for the entire set of attributes, the complexity of storing the MVD increases. For a database having 1500 attributes, typically the database
belongs to Cancer, DNA structures etc. The CW calculation will be ineffective and not suggested to be reliable. Since, there are chances of duplicate CWs generated for a huge set of attributes, the attributes are grouped, and this is a reliable measure implemented to generate CWs uniquely with various categories of attributes.

Second fold of reliability is, implementing the search and replacement of plausible value in the place of missing value. An efficient search mechanism for sorted data is implemented. The CWs are sorted and the search and replacement algorithm is performed on the CWs list, to locate the suitable record containing the plausible value.

5.7 Experimental Setup

The mining techniques proposed in the current research work are implemented and evaluated. The language used was Java and the experiments were performed on a Pentium IV 3.0 GHz with 1GB of memory, running Windows XP Professional. Some of the important phases of the framework have been implemented using Orange tool, which developed in Python language. Python is a variant of Object Oriented Programming Languages like C++ and Java. Effective GUI tool Orange presents widget-oriented approach for:

1. Sampling the data set
2. Visualizing the sample dataset

The data sample is very essential for testing the framework of imputation proposed in the research work, for various alternatives of samples of data. Unlike in the single and multiple imputation techniques, which propose a stochastic process (random variable set) as the result of experiment to be replaced in the data in the missing places, the framework proposes more exactly fitting value for the missing locations.
Data samples are very effectively presented visualization widgets of Orange tool. Distributions, Attribute Statistics, Scatterplot, Linear Projection, Radial Visualization, Polynomial Visualization, Parallel Coordinates, Survey Plot, Mosaic Display and Sieve Diagram. Radial visualization and Parallel Coordinates has been effectively used to present samples extracted from the dataset.

5.7.1 Sampling using “The Orange Data Sampler”

The File widget is used to select the input dataset. The input dataset for the experiment is presented in various formats. The Orange tool allows the user to specify the data in delimited data format, tabbed data format and any other user defined data formats. The input data set for the experiment is Mushroom (agaricus lepiota) Population Data. The data set consists of 8124 entries. The data set is a benchmark data set for testing the data mining algorithms, whose propensity is huge. The algorithms that have huge propensity, means, they produce huge sizes of patterns like classification, clustering. The data set selected contains more illustrious number of entries with all the possible combinations of the characteristics, and particularly the missing values are denoted by “?”.  

The missing value dataset is chosen with various types of criteria, all the different types of datasets are supplied to the framework to prove the efficiency of the framework An example of the criteria built for sampling the dataset is shown below:

Example 1:

![Data Selection Criteria](image)

**Fig: 5.13 Selection of the data using Attribute Selection Sequence in Orange [A]**
With the criteria specified as above, the size of sample data is 2239 entries. On random sampling the 326 instances are selected as input and 30 percent of instances are produced as output i.e., 98 instances. On the resulting sample of data 98 instances, the algorithm for imputation proposed in the framework is executed.

Example 2:

![Data Selection Criteria](image)

Fig: 5.14 Selection of the data using Attribute Selection Sequence in Orange [B]

With the criteria specified as above, the size of sample data is 2239 entries. On random sampling the 12 instances are selected as input and 30 percent of instances are produced as output i.e., 4 instances. On the resulting sample of data 4 instances, the algorithm for imputation proposed in the framework is executed.

A best case sampling will develop more number of instances where the strength of criteria is low. A worst case sampling develops less number of instances where the strength of criteria is high therefore the resulting sample will be small.

5.7.2 Generating Corpus

Corpus is an important storage in this framework. All the digest values are stored in the corpus orderly. The data structures are implemented meticulously to draw each digest value carefully for searching and replacing. The weights which are developed from each tuple are represented in a categorical attribute set
called as Missing Value digest, a separate data structure (grouped) is implemented to store all CWs as a MVD. All the categorical vectors of MVDs are collectively represented as Corpus.

Hashing algorithms are implemented to generate the weight values for each tuples.

5.7.3 Imputation

This is a two phase algorithm. In the first phase, (searching) the tuple for replacing the missing value is selected, selecting the attribute set (category of attributes) is selected and a CW is generated. The CW is searched in the Corpus, The MVDs which reflect the values of CW are selected and the entry positions of the data set is found, which contains the plausible value to replace in the missing value location. The second phase, the plausible value to be replaced is found and the same is replaced in the missing value location.

The algorithm complexity of the first phase of imputation is based on the complexity of 1) finding the missing value entry, 2) select the category of attribute set, 3) generate the characteristic weight, 4) search the characteristic weight, 5) find the location of missing value and 6) replace the plausible value with missing value.

5.7.4 Comparative Analysis

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Description</th>
<th>Mean</th>
<th>Regression</th>
<th>EfficientXmputer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Iteration</td>
<td>4</td>
<td>17</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Records</td>
<td>Limited</td>
<td>Up to 100</td>
<td>Unlimited</td>
</tr>
<tr>
<td>3</td>
<td>Accuracy</td>
<td>70% - 80%</td>
<td>Probabilistic</td>
<td>90% - 100%</td>
</tr>
</tbody>
</table>

Table: 5.3 Comparative Analysis tabulated

In the above table the comparative analysis between the related work and the proposed work has been described empirically. The Mean and Regression methods are considered as related work, which are implemented in C/C++
programming language and their algorithmic properties such as number of iterations, on number of tuples that algorithm in tolerant enough to produce result and accuracy of the results. The EfficientImputer produces imputation results efficiently compared to mean and regression methods. The time complexity of mean and regression changes linearly according to the number of records, as they perform full scan of dataset. The EfficientImputer works on the selected samples of the data set, the time complexity compared with mean and regression considerably low.

Summary

Orange Tool of Python Platform is a complimentary support for this chapter. Although this tool is a graphical user interface, the algorithms for sampling have been coded in python language and deployed as Orange Python widgets. Visual interface for selecting the database/dataset, attributes for performing important operations such as sampling, distribution check and visualization of sampled data are carried out elegantly. The total framework has been materialized with programming skills using Java language, and has been implemented successfully on the 4 real time data sets. In this chapter my research is forced to conclude with respect to the human limitations such as time, space and size of the problem; however the extensions and future work have been suggested as an upcoming research direction.