

P-2687



**TEST-COST SENSITIVE CLASSIFICATION ON DATA WITH
MISSING VALUES**

By

K.R.BASKARAN

Reg. No. 71206805001

of

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE – 641 006



A PROJECT REPORT

Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements

for the award of the degree

of

MASTER OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

JULY 2009

BONAFIDE CERTIFICATE

Certified that this project report titled entitled "TEST-COST SENSITIVE CLASSIFICATION ON DATA WITH MISSING VALUES" is the bonafide work of Mr.K.R.BASKARAN, who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.



SIGNATURE OF THE GUIDE

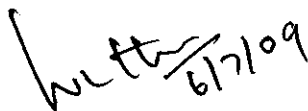
Mr.M.Nageswara Gupta, M.E.,
Senior Lecturer,
Department of Computer Science &
Engineering



HEAD OF THE DEPARTMENT

Dr. S.Thangasamy,
Dean & HOD
Department of Computer Science
& Engineering

The candidate with University Register No. 71206805001 was examined by us in the project viva-voce examination held on 6.7.09



INTERNAL EXAMINER



EXTERNAL EXAMINER

ABSTRACT

While classifying the data, the source data might have some missing values. In such cases, the total cost incurred in classification of data will be the sum of misclassification cost (cost incurred by misclassification errors) and test cost (cost incurred for obtaining missing attribute values).

Several methods are employed in predicting the missing values. The CPU time taken for predicting the various attributes of a record by each one of these prediction methods is tabulated. The efficiency (how correctly the missing value is predicted) of each one of these methods is also tabulated. The efficiency and processor time are in different units. Both of these values are converted into some constant value. The sum of these two values will be the total test cost for the missing value prediction. A comparison is made for the total test cost by each one of these methods for each one of the attributes of the database and the best prediction method (having least total test cost) is chosen for predicting the missing value attribute in a given input record for classification.

ஆய்வுச் சுருக்கம்

தரவினை வகைப்படுத்துகையில், மூலத் தரவினில் சில மதிப்புகள் பிசுகியிருக்கலாம். அவ்வாறு இருப்பின் தரவு வகைப்படுத்த ஆகும் மொத்த செலவு, தவறிய வகைப்படுத்தலின் செலவு (தவறிய வகைப்படுத்தலினால் ஏற்படும் செலவு) மற்றும் சோதனைச் செலவு (தவறிய பண்பு மதிப்புகள் பெறப்படுதலால் ஏற்படும் செலவு) ஆகியவையின் கூட்டுத்தொகையின் அளவு இருக்கும்.

தவறிய மதிப்புகளை முன் கணிக்க பல முறைகள் பயன்படுத்தப்பட்டன. பல்வேறு பண்புகளை முன் கணிக்க மேற்கூறிய ஒவ்வொரு முன் கணிப்பு முறையிலும் தேவைப்படும் மையச் செயலக நேரம் பட்டியலிடப்பட்டது. ஒவ்வொரு முன் கணிப்பு முறையின் செயற்றிறமையும் (எந்த அளவு துல்லியமாக தவறிய மதிப்புகள் முன் கணிக்கப்படுகின்றன) பட்டியலிடப்பட்டது. செயற்றிறமையும் மையச் செயலக நேரமும் வெவ்வேறு அளவைகளில் அளக்கப்படுகின்றன. இவ்விரு மதிப்புகளும் ஒரு நிலையான மதிப்பிற்கு மாற்றப்பட்டது. இவ்விரு மதிப்புகளின் கூட்டுத்தொகையே தவறிய மதிப்புகள் முன் கணிப்பிற்கான மொத்த சோதனைச் செலவாகும். தரவுத்தளத்தின் ஒவ்வொரு பண்பிற்கும் மேற்கூறிய முறைகள் ஒவ்வொன்றின் மொத்த சோதனைச் செலவும் ஒப்பிடப்பட்டு, கொடுக்கப்படும் உள்ளீடுகளை வகைப்படுத்திட தேவைப்படும் தவறிய மதிப்புடைய பண்புகளை முன் கணிக்க தேர்ந்தெடுக்கப்படுகின்றது

ACKNOWLEDGEMENT

I express my profound gratitude to our esteemed Chairman **Arutselvar Dr. N. Mahalingam, B.Sc., F.I.E.**, and Vice Chairman **Dr.K.Arumugam, B.E. (Hons), M.S (USA), M.I.E.**, for giving this great opportunity to pursue this course.

I thank **Prof. M.Annamalai**, Vice-Principal, Kumaraguru College of Technology, Coimbatore, for providing me with the necessary facilities and Infrastructure to work on this project.

I express my sincere thanks to **Dr.S.Thangasamy, Ph.D.**, Professor and Dean, Department of Computer Science and Engineering, for being my greatest of inspiration and for embedding the quest for innovative ideas.

I record my thanks to the Internal guide **Mr.M.Nageswara Guptha, M.E.**, Senior Lecturer in Department of Computer Science and Engineering , and **Ms.M.S.Hema, M.E.**, Senior Lecturer in Department of Computer Science and Engineering who have helped me to a great extent towards the successful implementation of this project.

I express my deep sense of gratitude and gratefulness to project coordinator **Mrs.V.Vanitha, M.E**, Asst. Professor, Department of Computer science and Engineering, for her supervision, tremendous patience, active involvement and guidance.

I would also like to convey my honest thanks to all teaching staff members and non-teaching staff members of the department for their support.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	iii
	ABSTRACT (TAMIL)	iv
	LIST OF TABLES	ix
	LIST OF FIGURES	x
	LIST ABBREVIATIONS	xi
1	INTRODUCTION	1
	1.1 Problem Definition.	1
	1.2 Prediction of Missing Values.	1
	1.3 Classification and Prediction.	2
	1.4 Preparing the Data for Classification and Prediction	4

2	Literature Survey	5
2.1	Cost of Misclassification errors.	
2.1.1	Constant Error Cost.	6
2.1.2	Conditional Error Cost	6
2.1.2.1	Error Cost Conditional on Individual Case.	6
2.1.2.2	Error Cost Conditional on Time of Classification.	6
2.1.2.3	Error Cost Conditional on Classification of other Cases.	7
2.1.2.4	Error Cost Conditional on Feature Value.	7
2.3	Cost of Tests.	7
2.3.1	Constant Test Cost.	7
2.3.2	Conditional Test Cost.	8
2.3.2.1	Test Cost Conditional on Prior Test Selection.	8
2.3.2.2	Test Cost Conditional on Prior Test Results.	8
2.3.2.3	Test Cost Conditional on True Class of Case.	8
2.3.2.4	Test Cost Conditional on Test Side-Effects.	8
2.3.2.5	Test Cost Conditional on Time of Test.	9
2.3	Building Decision Tree with Minimal Costs.	9

3	DETAILS OF PROPOSED METHODOLOGY	13
	3.1 Classification by Decision Tree Induction.	13
	3.2 Decision tree Induction.	13
	3.3 Attribute Selection Measure.	15
4	RESULTS AND DISCUSSION	17
	4.1 Performance Evaluation	17
5	CONCLUSION AND FUTURE WORK	21
	APPENDICES	23
	SCREEN SHOTS	52
	REFERENCES	55

LIST OF TABLES

TABLE NO.	TABLE NAME	PAGE NO.
2.0	Three different sets of attribute costs	11
4.0	Equivalence of CPU time	17
4.1	Mean method	17
4.2	Median method	18
4.3	K-Nearest Neighbor method	18
4.4	Highest Information Gain method	19

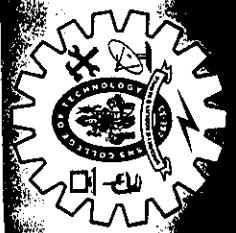
LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
1.0 a)	Learning.	2
1.0 b)	Classification.	3
2.0	Three different decision trees built with different test costs	11
2.1	Three different decision trees built with three different test costs as in Table 2.0.	12
4.1	Mean method.	17
4.2	Median method.	18
4.3	K-Nearest Neighbor method.	19
4.4	Highest Information Gain method.	20
4.5	Comparison of four methods.	20

LIST OF ABBREVIATIONS

FP False Positive.

FN False Negative.



SNS COLLEGE OF TECHNOLOGY

Approved by AICTE and Affiliated to Anna University
(An ISO 9001:2000 Certified Institution)



Contributors: S. Panikattu



Department of Information Technology
&
Society for Information Sciences and Computing Technology
Third National Conference on

NETWORKS, INTELLIGENCE AND COMPUTING SYSTEMS

This is to certify that **Dr./Prof./Mr./Ms. K.R. Baskaran M.E CSE**
of **Kannaguru College of Technology** has presented a technical paper
titled **Test-Cost Sensitive Classification on Data with Missing Values**
in the National Conference **NCNICS** held on **16th February 2009**.

Convenor

Prof. L.M. Nithya

Principal

Dr. V.P. Arunachalam

Director cum Secretary

Dr. S.N. Subbramanian

CHAPTER 1

1. INTRODUCTION

1.1. PROBLEM DEFINITION

Classification technique is used for classification of data. In this technique a model is constructed based on sample data and this model is used to classify the unknown data. During model construction process, the sample data is taken from the source data. The source data might have some missing values which will build a decision tree with misclassification and test cost. In the area of classification of data, only the cost of misclassification was predominantly considered earlier. It is equally important to consider how to handle the test costs associated with the querying of missing values.

Test cost is the cost associated with the finding of missing values through physical tests. In this project missing data is predicted using four different methods. The efficiency and CPU time for each one of these methods for predicting each one of the attributes of a record is computed. For each one of the attributes, the best method is predicted based on the minimum total cost.

1.2. PREDICTION OF MISSING VALUES

In this project emphasis is given for the minimization of combined test cost and misclassification cost. Test cost is the cost involved in prediction of missing values.

There are many methods for predicting missing values. Some of the popular methods are described below.

a. Ignore the tuple: This is usually done when the class label is missing (assuming the mining task involves classification or description) [3]. This method is not very effective, unless the tuple contains several attributes with missing values. It is especially poor when the percentage of missing values per attribute varies considerably.

b. Filling the missing value manually: In general, this approach is time-consuming and may not be feasible given a large data set with many missing values.

c. Use a global constant to fill in the missing value: replace all missing attribute values by the same constant, such as a label like "Unknown" or $-\infty$. If missing values are replaced by, say, "Unknown", then the mining program may mistakenly think that they form an interesting

concept, since they all have a value in common – that or “Unknown”. Hence, although this method is simple, it is not recommended.

d. Use the attribute mean to fill in the missing value: Calculate the average value of an attribute from a database. Use this average value to fill in a missing value for this attribute.

e. Use the attribute mean for all samples belonging to the same class as the given tuple: For example, if classifying customers according to credit risk, replace the missing value with the average income value for customers in same credit risk category as that of the given tuple.

f. Use the most probable value to fill in the missing value: This may be determined with regression, inference-based tools using a Bayesian formalism, or decision tree induction.

1.3. CLASSIFICATION AND PREDICTION

Classification and prediction are two forms of data analysis that can be used to extract models describing important data classes or to predict future data trends. Whereas classification predicts categorical labels, prediction models continuous-valued functions. For example, a classification model may be built to categorize bank loan applications as either safe or risky, while a prediction model may be built to predict the expenditures of potential customers on computer equipment given their income and occupation.

Data classification is a two-step process (Figure 1.0). In the first step, a model is built describing a predetermined set of data classes or concepts.

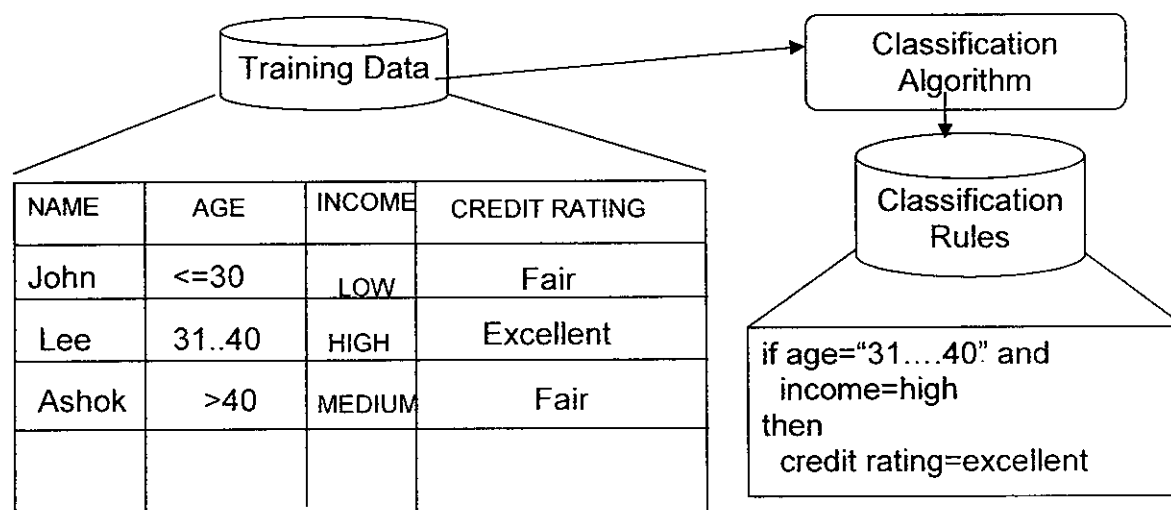


Fig. 1.0 (a) - Learning: Training data analyzed by a classification algorithm. Here, the class label attribute is credit rating, and the learned model or classifier is represented in the form of classification rules.

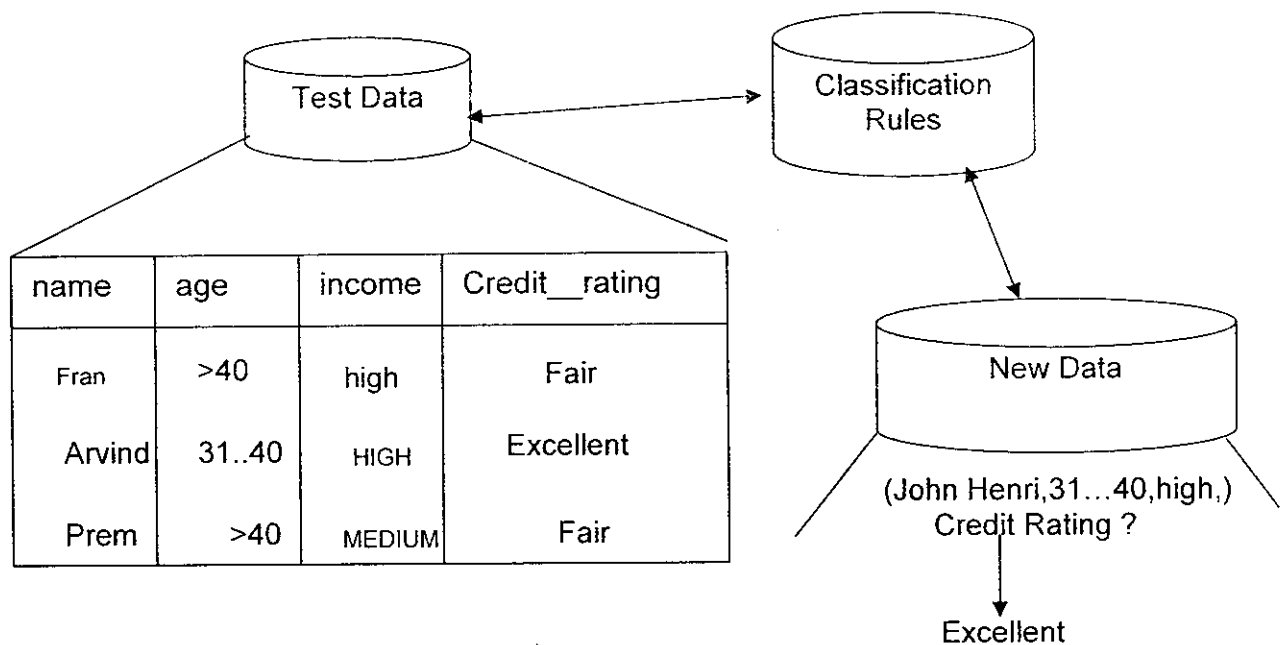


Fig 1.0 (b) Classification: Test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

The model is constructed by analyzing database tuples described by attributes. Each tuple is assumed to belong to a predefined class, as determined by one of the attributes, called the **class label attribute**. In the context of classification, data tuples are also referred to as samples, or objects. The data tuples analyzed to build the model collectively form the **training data set**. The individual tuples making up the training set are referred to as training samples and are randomly selected from the sample population. Since the class label of each training sample is provided, this step is also known as supervised learning. It contrasts with unsupervised learning, in which the class label of each training sample is not known, and the number or set of classes to be learned may not be known in advance.

Typically, the learned model is represented in the form of classification rules, decision trees, or mathematical formulae. For example, given a database of customer credit information, classification rules can be learned to identify customers as having either excellent or fair credit ratings. The rules can be used to categorize future data samples, as well as provide a better understanding of the database contents.

In the second step, the model is used for classification. First, the predictive accuracy of the model is estimated. The holdout method is a simple technique that uses a test set of class-labeled samples. These samples are randomly selected and are independent of the training samples. The accuracy of a model on a given test set is the percentage of test set samples that

are correctly classified by the model. For each test sample, the known class label is compared with the learned model's class prediction for that sample. If the accuracy of the model were estimated based on the training data set, this estimate could be optimistic since the learned model tends to over fit the data. Therefore, a test set is used.

If the accuracy of the model is considered acceptable, the model can be used to classify future data tuples or objects for which the class label is not known.

Prediction can be viewed as the construction and use of a model to assess the class of an unlabeled sample, or to assess the value or value ranges of an attribute that a given sample is likely to have. Classification is used to predict discrete or nominal values, while regression is used to predict continuous or ordered values. We refer to the use of prediction to predict class labels as classification, and the use of prediction to predict continuous values.

Classification and prediction have numerous applications including credit approval, medical diagnosis, performance prediction, and selective marketing.

1.4. PREPARING THE DATA FOR CLASSIFICATION AND PREDICTION

The following preprocessing steps may be applied to the data in order to help improve the accuracy, efficiency, and scalability of the classification or prediction process.

- **Data cleaning:** This refers to the preprocessing of data in order to remove or reduce noise and the treatment of missing values. This step can help reduce confusion during learning.
- **Relevance analysis:** Many of the attributes in the data may be irrelevant to the classification or prediction task. Hence, relevance analysis may be performed on the data with the aim of removing any irrelevant or redundant attributes from the learning process.
- **Data transformation:** The data can be generalized to higher-level concepts. Concept hierarchies may be used for this purpose. This is particularly useful for continuous-valued attributes. For example, numeric values for the attribute 'income' may be generalized to discrete ranges such as low, medium, and high. Similarly, nominal-valued attributes, like 'street', can be generalized to higher-level concepts, like 'city'.

CHAPTER 2

2. LITERATURE SURVEY

Inductive learning techniques, such as the naive Bayesian and decision tree algorithms, have met great success in building classification models with an aim to minimize the classification errors. However, previous inductive learning research has only focused on how to minimize classification costs such as the cost of false positive (FP) and the cost of false negative (FN)[1]. The classification errors are useful in deciding whether a learned model tends to make correct decisions on assigning class labels for new cases and is useful for dealing with data with unbalanced classes. However, misclassification costs are not the only costs to consider in practice. When performing classification on a new case, values for some attributes may be missing. In such a case, one may have the option of performing additional tests in order to obtain a value for these attributes. However, performing these additional tests may incur more costs, where some costs are in the form of lengthy waiting time and others include monetary payment. Still, some tests are worthwhile to perform because having the additional values might greatly increase the classification accuracy. Thus, one must often consider the “test cost” when missing values must be obtained through physical “tests” which incur costs themselves. These costs are often as important as the misclassification costs. As an example, consider the task of a medical practice that examines incoming patients for a certain illness. Suppose that the doctors’ previous experience has been compiled into a classification model such as a naïve Bayesian classifier. When dealing with an incoming patient, it is often the case that certain information for this patient may not yet be known; for example, the blood tests or the X-ray test may not have been done yet. At this point, the doctor (that is, the classification model) must exercise its judgment appropriately: Performing these tests will incur certain extra costs, but some tests may provide useful informational benefits toward reducing the classification costs. In the end, it is the balancing act of the two types of costs—namely, the classification costs and the test costs—that will determine which tests will be done. Tasks that incur both misclassification and test costs associated with missing values abound in industrial practice ranging from medical diagnosis to scientific research and to drug design. In the past, inductive learning methods that consider a variety of costs are often referred to as cost-sensitive learning. In this project both misclassification cost and test cost for prediction of missing values are considered.

2.1. COST OF MISCLASSIFICATION ERRORS

Suppose there are C classes. In general, we may have a $C \times C$ matrix, where the element in row i and column j specifies the cost of assigning a case to class i , when it actually belongs in class j [2]. Typically (but not necessarily) the cost is zero when i equals j . In a minor variation on this approach, one may have a rectangular matrix, where there is an extra row for the cost of assigning a case to the *unknown* (or “too-difficult-for-this-learner”) class.

2.1.1 Constant error cost

The cost of a certain type of error (the value of a cell in the cost matrix) may be a constant (the same value for all cases). This is the most commonly investigated type of cost. If the cost is zero if i equals j and one otherwise, then our cost measure is the familiar *error-rate* measure. If the cost is one if i equals j and zero otherwise, then our cost measure is the familiar *accuracy* measure.

2.1.2 Conditional error cost

The cost of a certain type of error may be conditional on the circumstances.

2.1.2.1 Error cost conditional on individual case

The cost of a classification error may depend on the nature of the particular case. For example, in detection of fraud, the cost of missing a particular case of fraud will depend on the amount of money involved in that particular case. Similarly, the cost of a certain kind of mistaken medical diagnosis may be conditional on the particular patient who is misdiagnosed. For example, the misdiagnosis may be more costly in elderly patients. It may be possible to represent this situation with a constant error cost by distinguishing sub-classes. For example, instead of two classes, “sick” and “healthy”, there could be three classes, “sick-and-young”, “sick-and elderly”, and “healthy”. This is an imperfect solution when the cost varies continuously, rather than discretely.

2.1.2.2. Error cost conditional on time of classification

In a time-series application, the cost of a classification error may depend on the timing. Consider a classifier that monitors sensors that measure a complex system, such as a manufacturing process or a medical device. Suppose that the classifier is intended to signal an alarm if a problem has occurred or will soon occur. The sensor readings must be classified as

either “alarm” or “no alarm”. The cost of the classification depends on whether the classification is correct and also on the timeliness of the classification. The alarm is not useful unless there is sufficient time for an adequate response to the alarm. Again, it may be possible to represent this situation with a constant error cost by distinguishing sub-classes. Instead of two classes, “alarm” and “no-alarm”, there could be “alarm-with-lots-of-time”, “alarm-with-a-little-time”, “alarm-with-no-time”, and “no-alarm”. Again, this is an imperfect solution when the cost varies continuously as a function of the timeliness of the alarm.

2.1.2.3. Error cost conditional on classification of other cases

In some applications, the cost of making a classification error with one case may depend on whether errors have been made with other cases. The familiar *precision* and *recall* measures, widely used in the information retrieval literature, may be seen as cost measures of this type. For example, consider an information retrieval task, where one is searching for a document on a certain topic. Suppose that he/she would be happy if he/she could find even one document on this topic. If we are given a collection of documents to classify as “relevant” or “not-relevant” for the given topic, then the cost of mistakenly assigning a relevant document to the not relevant class depends on whether there are any other relevant documents that one has correctly classified. As another example, in activity monitoring, if one issues an alarm twice in succession for the same problem, the benefit of the second alarm is less than the benefit of the first alarm, assuming both alarms are correct classifications.

2.1.2.4. Error cost conditional on feature value

The cost of making a classification error with a particular case may depend on the value of one or more features of the case

2.2. COST OF TESTS

Each test (i.e., attribute, measurement, and feature) may have an associated cost. For example, in medical diagnosis, a blood test has a cost. One can only rationally determine whether it is worthwhile to pay the cost of a test when he/she knows the cost of misclassification errors. If the cost of misclassification errors is much greater than the cost of tests, then it is rational to purchase all tests that seem to have some predictive value. If the cost of misclassification errors is much less than the cost of tests, then it is not rational to purchase any tests.

2.2.1 Constant test cost

The cost of performing a certain test may be a constant. Each test has a different cost, but the cost of a given test is the same for all cases.

2.2.2 Conditional test cost

The cost of performing a certain test may be conditional on the circumstances surrounding the test.

2.2.2.1 Test cost conditional on prior test selection

The cost of performing a certain test on a given patient may be conditional on the previous tests that have been chosen for the patient. For example, a group of blood tests ordered together may be cheaper than the sum of the costs of each test considered by it, since the tests share common costs, such as the cost of collecting blood from the patient.

2.2.2.2 Test cost conditional on prior test results

The cost of performing a certain test on a patient may be conditional on the results of previous tests. For example, the cost of a blood test is conditional on the patient's age. Thus a blood test must be preceded by a "patient-age" test, which determines the cost of the blood test.

2.2.2.3. Test cost conditional on true class of case

The cost of performing a certain test on a patient may be conditional on the correct diagnosis of the patient. For example, the cost of an exercise stress test on a patient may be conditional on whether the patient has heart disease. The stress test could cause heart failure, which adds to the total cost of the test.

2.2.2.4 Test cost conditional on test side-effects

The cost of performing a certain test on a patient may be conditional on possible side-effects of the test. For example, some patients are allergic to the dyes that are used in certain radiological procedures. One side-effect of such a radiological test is an allergic reaction, which may increase the cost of the test.

2.2.2.5 Test cost conditional on time of test

The cost of performing a certain test may depend on the timing of the test.

2.3. BUILDING DECISION TREE WITH MINIMAL COSTS

One assumes that the training data may consist of missing values (whose values cannot be obtained) [4]. A static cost structure is also assumed where the cost is not a function of time or cases. Further, it is assumed that the test cost and the misclassification cost have been defined on the same cost scale, such as the dollar cost incurred in a medical diagnosis.

The new decision-tree learning algorithm is quite simple. One considers discrete attribute and binary class labels; extensions to other cases can also be made. It is also assumed that FP is the cost of one false positive example, and FN is the cost of one false negative example. The algorithm uses a new splitting criterion of minimal total cost on training data, instead of minimal entropy, to build decision trees. At each step, rather than choosing an attribute that minimizes the entropy, the algorithm chooses an attribute that reduces and minimizes the total cost, the sum of the test cost and the misclassification cost, for the split. Then, the algorithm chooses a locally optimal attribute without backtracking. Thus the resulting tree may not be globally optimal. However, the efficiency of the tree-building algorithm is generally high.

. In many variations of decision tree algorithms, the unknown value is treated as an ordinary value. However, in this work, the strategy is that all unknown values (“?”) are treated as a special “value”: no leaf or sub-tree will be built for examples with the “?” value. This is because it is unrealistic to assume the unknown values would be as useful for classification as the known values. In addition, when a test example is stopped at an attribute whose value is unknown, if the attribute has a “?” branch, it is impossible to decide whether the test should be performed by the tree. Therefore, the examples with unknown attribute values are not grouped together as a leaf, or built into a sub-tree; instead, they are “gathered” inside the node that represents that attribute. One then calculates the ratio of the positive and negative examples in the internal node. The second test algorithm will incorporate such ratios in making predictions.

Another important point is how the leaves are labeled. In traditional decision tree algorithms, the majority class is used to label the leaf node. In our case, as the decision tree is used to make predictions to minimize the total cost, the leaves are labeled also to minimize the total cost. That is, at each leaf, the algorithm labels the leaf as either positive or negative (in a binary decision case) by minimizing the misclassification cost. Suppose that the leaf has P positive examples, and N negative examples. If $P \times FN > N \times FP$ (i.e., the cost of predicting negative is greater than the cost of predicting positive), then the leaf is labeled as positive; otherwise it is labeled as negative. Therefore, the label of a leaf does not just depend on the majority class of the leaf, but also on the cost of misclassification.

Let us look at a concrete example. Assume that during the tree building process, there is a set of P and N positive and negative examples respectively to be further classified by possibly building a sub-tree. If we assume that $P \times FN > N \times FP$, then, if no sub-tree is built, the set would be labeled as positive, and thus, the total misclassification cost is $T = N \times FP$. Suppose that an attribute A with a test cost C is considered for a potential splitting attribute. Assume that A has two values, and there are P_1 and N_1 positive and negative examples with the first value, P_2 and N_2 positive and negative examples with the second value, and P_0 and N_0 positive and negative examples with A 's value unknown. Then the total test cost would be $(P_1 + N_1 + P_2 + N_2) \times C$ (i.e., cases with unknown attribute values do not incur test costs). Assume that the first branch is labeled positive (as $P_1 \times FN > N_1 \times FP$), and the second branch is labeled negative, then the total misclassification cost of the two branches is $N_1 \times FP + P_2 \times FN$. As we have discussed earlier in this section, examples with the unknown value of A stay with the attribute A , and we have assumed that the original set of examples is labeled as positive. Thus, the misclassification cost of the unknowns is $N_0 \times FP$. The total cost of choosing A as a splitting attribute would be:

$$TA = (P_1 + N_1 + P_2 + N_2) \times C + N_1 \times FP + P_2 \times FN + N_0 \times FP$$

If $TA < T$, where $T = N \times FP$, then splitting on A would reduce the total cost of the original set, and we would then choose an attribute with the minimal total cost as a splitting attribute. One then applies this process recursively on examples falling into branches of this attribute. If $TA \geq T$ for all remaining attributes, then no further sub-tree will be built, and the set would become a leaf, with a positive label.

Finally, as the tree attempts to minimize the total cost, it may also overfit the training dataset. Aimed at minimizing the total cost of test and misclassification, the new decision-tree algorithm has several desirable features. This dataset, after pre-processing, has 332 labeled examples, which are described by six attributes. The numerical attributes are first discretized using the minimal entropy method, as this tree building algorithm can only accept discrete attributes (but it is straightforward to extend this algorithm to accept continuous attributes). The attribute values are renamed as 1, 2, 3, and so on.

The first property is that the relative difference between misclassification and test costs can affect the tree dramatically. If the former is less than the latter, then no test should be performed, and the decision tree would be simply a one-node leaf. On the other hand, if the former is much larger than the latter, then all tests should be done, as long as they are relevant; i.e., they can improve the predictive accuracy. This can be seen clearly from the "Ecoli" dataset. Indeed, if the misclassification cost is set to 200 for both FP and FN, and all test cost is

set to 300, then the algorithm returns a one-leaf node as shown in Figure 3.0 (a). On the other hand, when all test costs are set to zero, then the tree is the “largest”; in this case, the tree has 13 nodes in total, and can be seen in Figure 2.0(c). As an “intermediate” case, if all test costs are set to 20, then the decision tree with the minimal cost has six nodes in total, and the tree can be seen in Figure 2.0 (b).

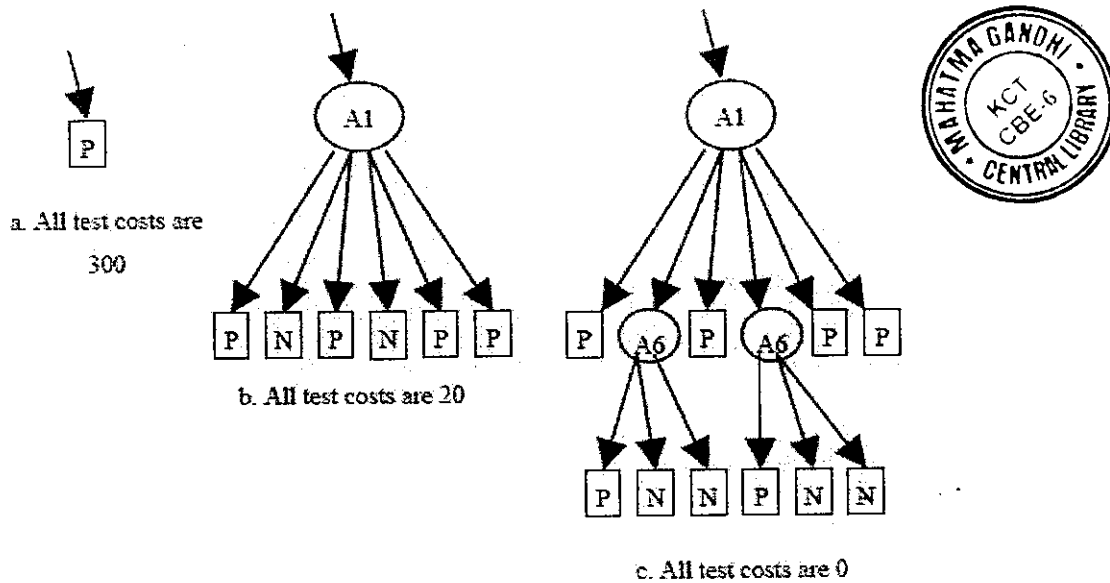


Fig.2.0. Three different decision trees built with different test costs.

The second important and desirable property is that for attributes with different test costs, this new algorithm is likely to choose an attribute with zero or the smallest cost at the top (or root) of the tree. This is because the attribute at the root will be tested by all examples, and thus the total attribute cost would be relatively high. Choosing an attribute with zero or the smallest cost helps reduce the total cost. Of course attribute selection also depends on the distribution of attribute values and class labels of the training examples.

Table 2.0 - Three different sets of attribute costs.

COST	A1	A2	A3	A4	A5	A6
Tree # 1	20	20	20	20	20	20
Tree # 2	200	20	100	100	200	200
Tree # 3	200	100	100	100	20	200

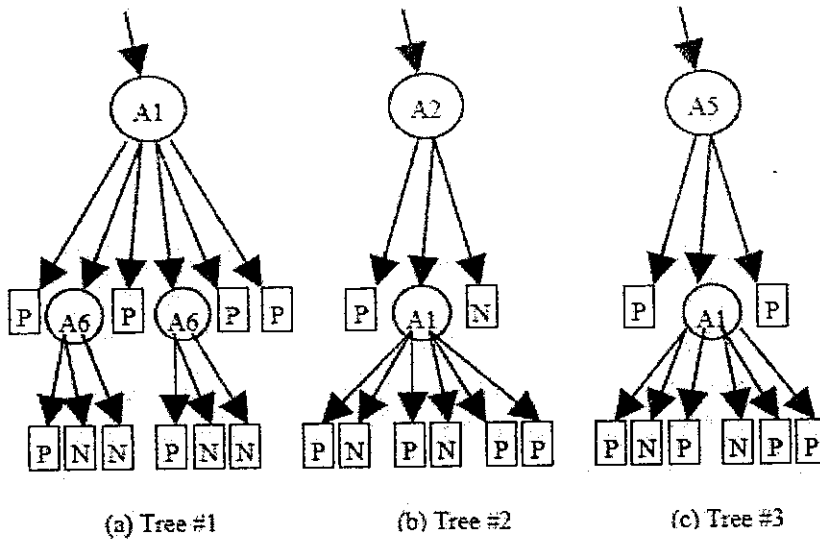


Fig. 2.1 - Three different decision trees built with three different test costs as in Table 2.0.

Table 2.0 shows three cases in which attribute costs are different. In the first case (the baseline), all attribute costs are set to 20. In the second and third cases attribute costs are set differently. The misclassification cost is set at 800 for both FP and FN. As we can see, in the second case, the attribute A2 has the smallest test cost, and it is indeed chosen as the root of the tree as shown in Figure 2.1(b). In the third case, attribute A5 has the smallest test cost, and it is chosen as the root (Figure 2.1(c)).

The third property, related to the second one, is that when the test cost of an attribute is increased, that test attribute will be “pushed” down in the tree, until it “falls out” of the tree (when the test cost becomes too large). If the test cost of A1 is set to 20, 50, and 80, respectively, while other costs are fixed, we obtain trees (not shown here) with A1 at the root (similar to Figure 2.1(a)), in the middle of the tree (similar to Figure 2.1 (b)), and not in the tree, respectively

CHAPTER 3

3. DETAILS OF PROPOSED METHODOLOGY

3.1 CLASSIFICATION BY DECISION TREE INDUCTION

A decision tree is a flow - chart like structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and leaf nodes represent classes or class distributions [3]. The topmost node in a tree is the root node. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals.

In order to classify an unknown sample, the attribute values of the sample are tested against the decision tree. A path is traced from the root to a leaf node that holds the class prediction for that sample. Decision trees can easily be converted to classification rules.

Basic algorithm for inducing a decision tree from training samples

- a. Create a node N;
- b. If samples are all of the same class, C then
- c. Return N as a leaf node labeled with the class C;
- d. If attribute-list is empty then
- e. Return N as a leaf node labeled with the most common class in samples;
- f. Select test-attribute, the attribute among attribute-list with the highest information gain;
- g. Label node N with test-attribute;
- h. For each known value a_i of test-attribute
- i. Grow a branch from node N for the condition test-attribute = a_i ;
- j. Let s_i be the set of samples in samples for which test-attribute = a_i ;
- k. If s_i is empty then
- l. Attach a leaf labeled with the most common class in samples;
- m. Else attach the node

3.2 DECISION TREE INDUCTION

The basic algorithm for decision tree induction is a greedy algorithm that constructs decision trees in a top-down recursive divide and conquer manner. The basic strategy is as follows

- The tree starts as a single node representing the training samples

- If the samples are all of the same class, then the node becomes a leaf and is labeled with that class
- Otherwise, the algorithm uses an entropy based measure known as information gain as a heuristic for selecting the attribute that will best separate the samples into individual classes. This attribute becomes the “test” or “decision” attribute at the node. In this version of the algorithm, all attributes are categorical, that is discrete-valued. Continuous-valued attributes must be discretized.
- A branch is created for each known value of the test attribute, and the samples are partitioned accordingly.
- The algorithm uses the same process recursively to form a decision tree for the samples at each partition. Once an attribute has occurred at a node, it need not be considered in any of the node’s descendents.
- The recursive partitioning stops only when any one of the following conditions is true:
 - a. All samples for a given node belong to the same class, or
 - b. There are no remaining attributes on which the samples may be further partitioned. In this case, majority voting is employed. This involves converting the given node into a leaf and labeling it with the class in majority among samples. Alternatively, the class distribution of the node samples may be stored.
 - c. There are no samples for the branch test-attribute = a_i . In this case, a leaf is created with the majority class in samples.

3.3 ATTRIBUTE SELECTION MEASURE

The information gain measure is used to select the test attribute at each node in the tree. Such a measure is referred to as an attribute selection measure or a measure of the goodness of split. The attribute with the highest information gain is chosen as the test attribute for the current node. This attribute minimizes the information needed to classify the samples in the resulting partitions and reflects the least randomness or “impurity” in these partitions. Such an information- theoretic approach minimizes the expected number of tests needed to classify an object and guarantees that a simple tree is found.

Let S be a set consisting of s ’ data samples. Suppose the class label attribute has ‘ m ’ distinct values defining ‘ m ’ distinct classes, C_i (for $i= 1, \dots, m$). Let s_i be the number of

samples of S in class C_i . The expected information needed to classify a given sample is given by

$$I(s_1, s_2, \dots, s_m) = -\sum_{i=1}^m p_i \log_2(p_i)$$

Where p_i is the probability that an arbitrary sample belongs to class C_i and is estimated by s_i/s .

Let attribute A have v distinct values $\{a_1, a_2, \dots, a_v\}$. Attribute A can be used to partition S into v subsets, $\{S_1, S_2, \dots, S_v\}$, where S_j contains those samples in S that have value a_j of A. If A were selected as the test attribute, then these subsets would correspond to the branches grown from the node containing the set S. Let s_{ij} be the number of samples of class C_i in a subset S_j . The entropy, or expected information based on the partitioning into subsets by A, is given by

$$E(A) = \sum_{j=1}^v \frac{s_{1j} + \dots + s_{mj}}{s} I(s_{1j}, \dots, s_{mj})$$

The term $\frac{s_{1j} + \dots + s_{mj}}{s}$ acts as the weight of the j th subset and is the

number of samples in S. The smaller the entropy value, the greater the purity of the subset partitions. For a given subset S_j ,

$$I(s_{1j}, s_{2j}, \dots, s_{mj}) = -\sum_{i=1}^m P_{ij} \log_2(p_{ij})$$

where $p_{ij} = \frac{s_{ij}}{|S_j|}$ and is the probability that a sample in s_j belongs to class C_i .

The encoding information that would be gained by branching on A is

$$\text{Gain}(A) = I(s_1, s_2, \dots, s_m) - E(A)$$

In other words, Gain (A) is the expected reduction in entropy caused by knowing the value of attribute A.

The algorithm computes the information gain of each attribute. The attribute with the highest information gain is chosen as the test attribute for the given set S . A node is created and labeled with the attribute, branches are created for each value of the attribute, and the samples are partitioned accordingly.

CHAPTER 4

4. PERFORMANCE EVALUATION

EQUIVALENCE OF CPU TIME:

CPU Time Range	Equivalence cost(assumption)
0~0.025	10
0.026~0.030	15
0.031~0.040	20
0.041~0.050	25
>0.051	30

Tab 4.0 Equivalence of CPU time

Method1 –Using Mean method to find the missing value:

Attributes	Cost
a1(Age)	30
a2(student)	30
a3(Credit Rating)	25
a4(Income)	30

Tab 4.1 Method1 –Using Mean method to find the missing value

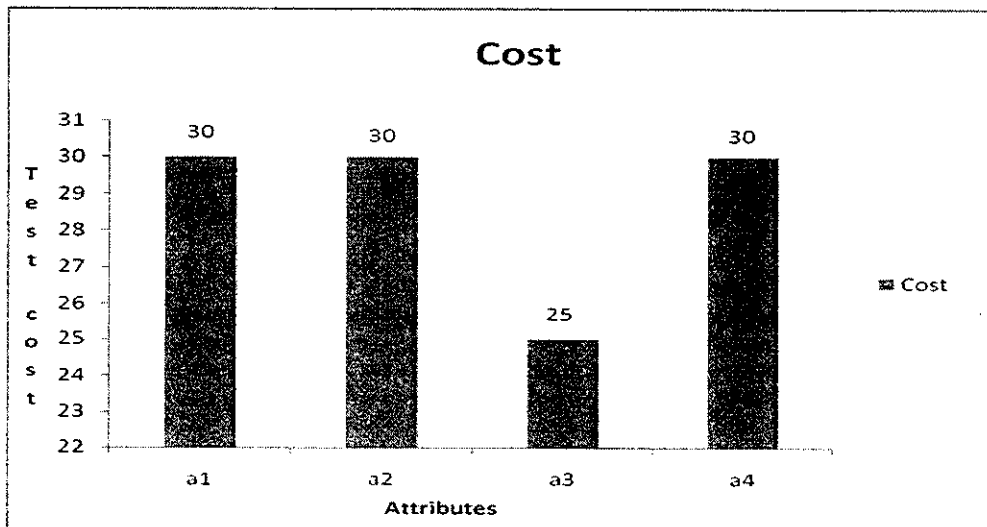


Fig 4.1 Method1- Using Mean method to find the missing value

Method2- Using Median method to find the missing value:

Attributes	Cost
a1(Age)	15
a2(student)	20
a3(Credit Rating)	15
a4(Income)	15

Tab 4.2 **Method2** – Using Median method to find the missing value

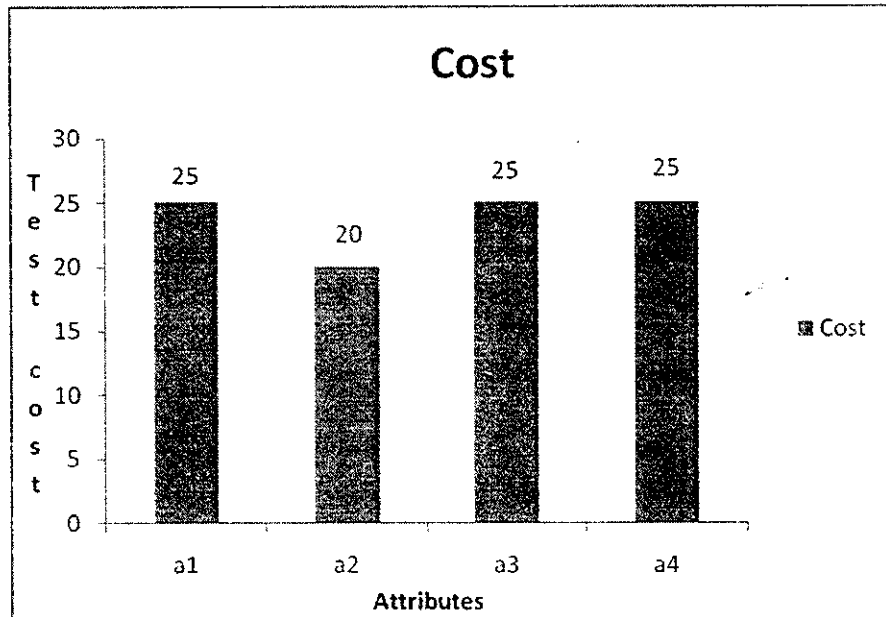


Fig 4.2 Method2 –Using Median method to find the missing value

Method3 – Using K-Nearest Neighbor method to find the missing value:

In this method the missing value for an attribute is computed by taking the mean value of the last ten tuples (in general last n tuples) for that attribute from the database

Attributes	Cost
a1(Age)	5
a2(student)	10
a3(CreditRating)	15
a4(Income)	20

Tab 4.3 Method3- Using K-Nearest Neighbor method to find the missing value

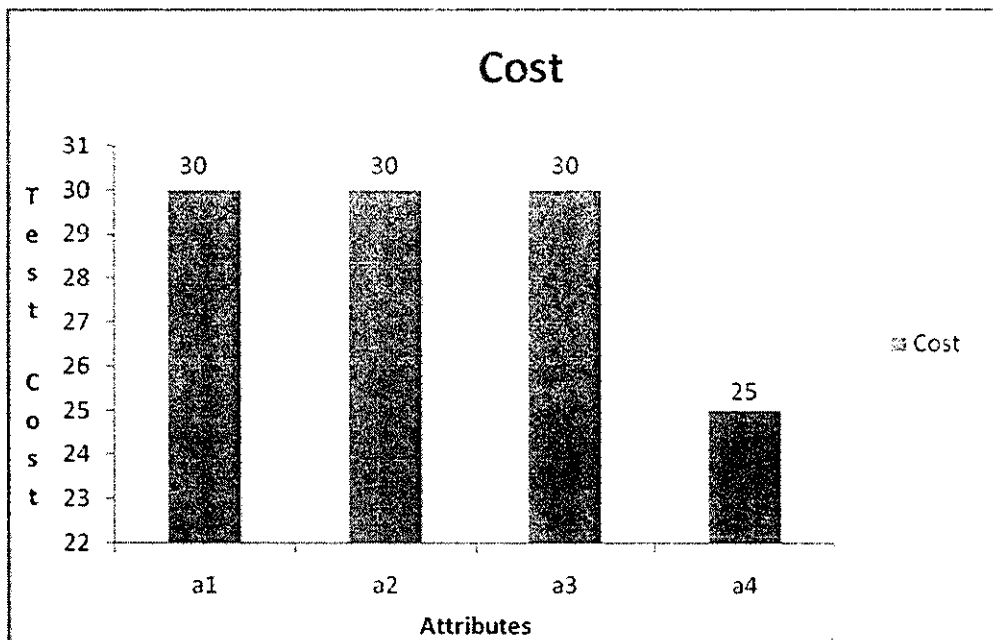


Fig 4.3 Method3- Using K-Nearest Neighbor method to find the missing value

Method 4 - Based on highest information gain of remaining attributes:

Select the attribute that has the highest Information gain excluding the attribute that has missing value. Find the value of that highest information gain attribute. From the data base find the maximum occurrence value for the missing value attribute that corresponds to the value of the selected attribute having highest information gain.

Attributes	Cost
a1(Age)	10
a2(student)	15
a3(Credit Rating)	20
a4(Income)	25

Tab 4.4 Method 4 - Based on highest information gain of remaining attributes:

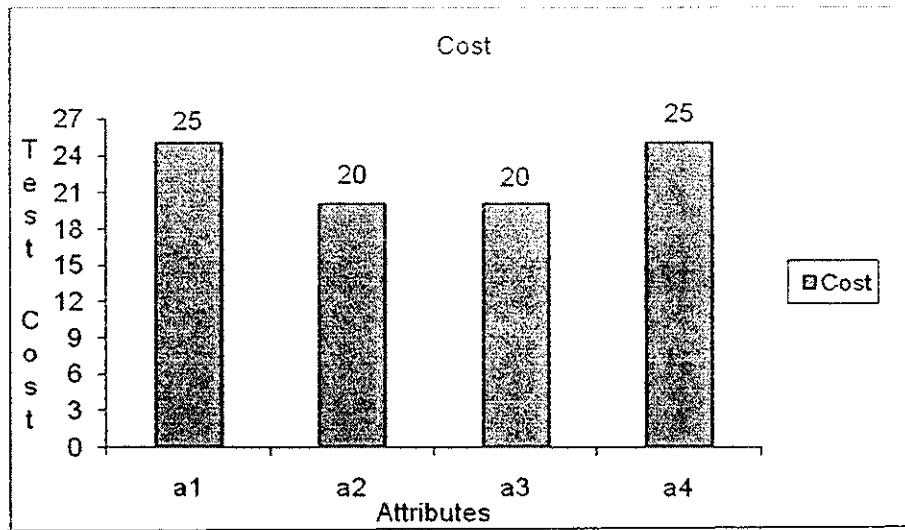


Fig 4.4 Method 4- Based on highest information gain of remaining attributes:

COMPARISON OF TOTAL COST USING THE ABOVE FOUR METHODS OF PREDICTION OF MISSING VALUES.

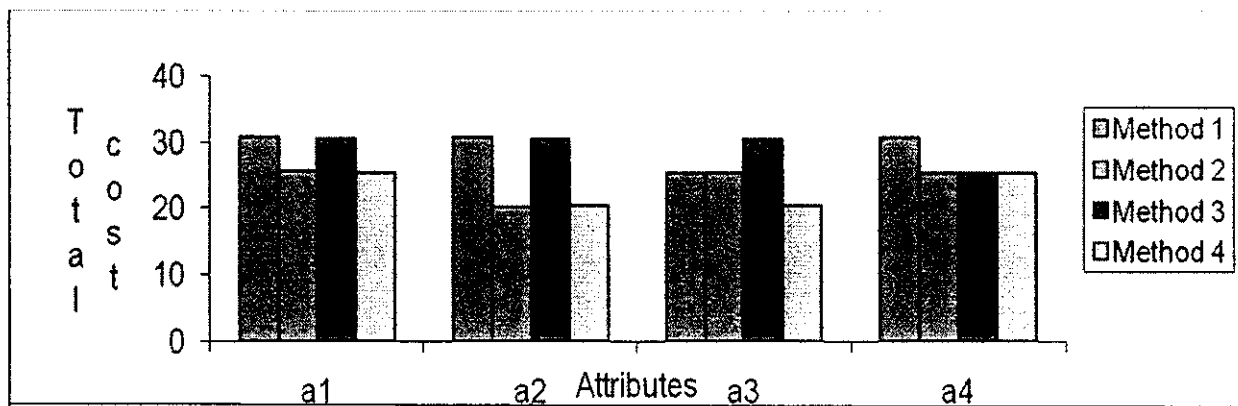


Fig 4.5 Comparison of total test cost of four methods of missing value prediction.

CHAPTER 5

5. CONCLUSION

In our Application,

For **Age** attribute, Method4 ie., **Based on highest information gain** of remaining attributes incurs low total cost when compared to all the other three methods.

For **Student** attribute, Method2 ie., **Median value of that attribute** incurs low total cost when compared to all the other three methods.

For **Credit rating** attribute, Method4 ie., **Based on highest information gain** of remaining attributes incurs low total cost when compared to all the other three methods.

For **Income** attribute, Method4 ie., **Based on highest information gain** of remaining attributes incurs low total cost when compared to all the other three methods.

FUTURE ENHANCEMENTS

Presently, we have focused on single missing value in a tuple. But if there are more than one missing value in a single tuple, then the concept of Threshold factor can be introduced. Threshold factor is the maximum limit up to which the test cost can be incurred. When the total cost goes beyond the threshold fixed value, there will be no gain in further prediction of missing value. If two or more attributes have missing values in a tuple, attributes are predicted in order of their Information Gain value up to the fixed threshold for test cost. For the remaining missing attributes, value is guessed based on the number of tuples classified according to a particular value for that attribute. In the future, one can also do the classification of missing data using Naïve -Bayesian algorithm and make a comparison between the two methods.

APPENDICES

APPENDIX

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
FILE *in,*out,*in1;
void main()
{
char str[500],*str1="";
int cnt=0,ta=5;
in=fopen("miss.c","r");
in1=fopen("ip2.c","r");
out=fopen("in5.c","w");
while(!feof(in1))
{

if(cnt==ta)
{
fprintf(out,"%s\n",str1);
cnt=0;
}
fscanf(in1,"%s",str);
cnt++;
if(strcmp(str,"*")==0)
{
printf("%s",str);
fscanf(in,"%s",str);
fprintf(out,"%s ",str);
}
else
{
printf("%s ",str);
// fscanf(in1,"%s",str);
fprintf(out,"%s ",str);
```

```
}  
}  
fclose(in);  
fclose(in1);  
fclose(out);  
}
```

```
#include<stdio.h>  
#include<conio.h>  
#include<process.h>  
#include<errno.h>
```

```
main()  
{  
    int s;  
    char *str[10],str1[10]="in4.c";  
    clrscr();  
    strcpy(str[1],str1);  
    printf("SFsf");  
    getch();  
    //execv("tree(2).exe",str);  
    spawnv(P_WAIT,"tree(2).exe", str);  
    printf("gszfg");  
    //printf("%d",s);  
    //perror("exec error");  
    //exit(1);  
    //execlp("tree(2).exe","tree(2).exe","in4.c","in5.c");  
}
```

```
#include<stdio.h>  
#include<conio.h>  
#include<process.h>  
#include<errno.h>
```

```

main()
{
int s;
char *str[10],str1[10]="in4.c";

FILE *in,*out,*po;
void fun();
int ta=0,ca=0,fa=0,i=0,b=0,s=0;
char str1[100];
void main()
{
char str[200];
clrscr();
in=fopen("input.c","r");
po=fopen("in3.c","w");
fgets(str,300,in);
while(str[i]!='\n')
{
// printf("%c",str[i]);
if(str[i]==32)
{
ta++;
while(str[i]==32)
i++;
}
i++;
}
fclose(in);
out=fopen("input.c","r");
fprintf(po,"%d\n",ta);
while(fa<ta)
{
fscanf(out,"%s",str1);
et[s].index=s;

```

```

strcpy(et[s++].atname, str1);
fa++;
}

fscanf(out, "%s", str1);
// printf("%s", str1);
i=0;
s=0;
fun();

}

m=1;
for(k=1; k<=b; k++)
{
p1=cal[k].cy+cal[k].cn;
p=(double)cal[k].cy/p1;
pn=(double)cal[k].cn/p1;
// printf("%d %d %d %f %f", cal[k].cy, cal[k].cn, p1, p, pn);
if(p==0)
p=1;
if(pn==0)
pn=1;
st[m].ig=3.32* ((-1* p *log10 (p))+(-1*pn*log10(pn)));
// printf("--%f %f %f\n", st[m].ig, log10(p), log10(pn));
m++;
}
for(k=1; k<=m; k++)
{
p=(double)(cal[k].cy+cal[k].cn)/14;
f=(double)p*st[k].ig;
// printf("-@--%f %f %f\n", p, st[k].ig, f);
et[s].g+=f;
}
printf("\ninformation gain-%f %d", et[s].g, s);

```

```

getch();
for(k=0;k<=b;k++)
{
    cal[k].cy=cal[k].cn=0;
    strcpy(cal[k].name,"");
}
b=0;i=0;
s++;
ca++;
fclose(out);
out=fopen("input.c","r");
fscanf(out,"%s",str1);
fa=0;
while(fa<ta)
{
    fscanf(out,"%s",str1);
    fa++;
}
//printf("%s",str1);
//exit(0);
}
out=fopen("input.c","r");
fscanf(out,"%s",str1);
fa=0;
while(fa<ta)
{
    fscanf(out,"%s",str1);
    fa++;
}
fa=0;
while(!feof(out))
{
    while(fa<ta)
    {
        fscanf(out,"%s",str1);

```

```

    fa++;
}
fscanf(out,"%s",str1);
if(strcmp(str1,"yes")==0)
    cy++;
else
    cn++;
// printf("%s\n",str1);
fa=0;
}
p=(double)cy/14;
pn=(double)cn/14;
pn=3.32* ((-1* p *log10 (p))+(-1*pn*log10(pn)));
printf("\ntotal entropy value =%f",pn);

for(i=0;i<ta;i++)
{
    et[i].fi=pn-et[i].g;
    printf("\n%-15s %-15s %-15f %15f\n",et[i].atname,et[i].atype[1],et[i].g,et[i].fi);
}
swap();
printf("-----");
for(i=0;i<ta;i++)
    printf("\n%-15s %-15s %-15f %15f\n",et[i].atname,et[i].atype[1],et[i].g,et[i].fi);
fclose(out);
files();
}
int getindex()
{
    int j;
    for(j=0;j<=b;j++)
        if(strcmp(cal[j].name,str1)==0)
            return j;
    b=++i;
    return b;
}

```



```

}

int swap()
{
    fprintf(po,"%s\n","no");
    out=fopen("input.c","r");
    fscanf(out,"%s",str1);
    while(ca<ta)
    {
        fscanf(out,"%s",str1);
        ca++;
    }

    // printf("%s",str1);
    fscanf(out,"%s",str1);
    while(!feof(out))
    {
        ca=0;

        while(ca<=ta)
        {
            strcpy(a[ca],str1);
            fscanf(out,"%s",str1);
            ca++;
        }
        for(m=0;m<ta;m++)
        {
            n=et[m].index;
            fprintf(po,"%s ",a[n]);
        }

        fprintf(po,"%s\n",a[ta]);
    }
}

```

```
return 1;
}
```

```
age income student credit_rating cla
<=30 h no fair no
<=30 h no excellent no
31-40 h no fair yes
>40 m no fair yes
>40 l yes fair yes
>40 l yes excellent no
31-40 l yes excellent yes
<=30 m no fair no
<=30 l yes fair yes
>40 m yes fair yes
<=30 m yes excellent yes
31-40 h yes fair yes
>40 m no excellent no
```

4

age 3 <=30 31-40 >40

student 2 yes no

credit 2 fair excellent

income 3 h m l

cla 2 yes no

```
<=30 no fair h no
<=30 no excellent h no
31-40 no fair h yes
>40 no fair m yes
>40 yes fair l yes
>40 yes excellent l no
31-40 yes excellent l yes
<=30 no fair m no
<=30 yes fair l yes
```

>40	yes	fair	m	yes
<=30	yes	excellent	m	yes
31-40	yes	fair	h	yes
>40	no	excellent	m	no
31-40	no	fair	h	yes
>40	no	fair	m	yes

<=30 yes fair l yes

>40 yes fair m yes

<=30 yes excellent m yes

31-40 no excellent m yes

31-40 yes fair h yes

>40 no excellent m yes

<=30 no fair h no

>40 yes fair h no

<=30 no fair m yes

31-40 no fair m yes

<=30 no fair l yes

>40 yes excellent l no

31-40 no excellent l yes

<=30 no fair m no

<=30 no fair l yes

>40 yes fair h yes

31-40 no excellent h yes

>40 yes excellent h no

>40 yes fair h yes

>40 yes fair m yes

Yes

age income student credit_rating cla

<=30 h no fair no

<=30 h no excellent no

31-40 h no fair yes

>40 m no fair yes

>40	l	yes	fair	yes
>40	l	yes	excellent	no
31-40	l	yes	excellent	yes
<=30	m	no	fair	no
<=30	l	yes	fair	yes
>40	m	yes	fair	yes
<=30	m	yes	excellent	yes
31-40	m	no	excellent	yes
31-40	h	yes	fair	yes
>40	m	no	excellent	no

*	no	fair	h	no
*	no	fair	h	no
*	no	fair	m	yes
*	no	fair	m	yes
*	yes	fair	l	yes
*	yes	excellent	l	no
*	yes	excellent	l	yes
*	yes	fair	m	no
*	yes	fair	l	yes
*	yes	fair	h	yes
*	yes	excellent	h	yes
*	no	excellent	m	no
*	no	fair	h	yes
*	no	fair	m	yes
*	no	fair	h	no

<=30	*	fair	h	no
>40	*	fair	h	no
<=30	*	fair	m	yes
31-40	*	fair	m	yes
<=30	*	fair	l	yes
>40	*	excellent	l	no
31-40	*	excellent	l	yes
<=30	*	fair	m	no

<=30	*	fair	l	yes
31-40	yes	*	h	yes
>40	no	*	m	no
31-40	no	*	h	yes
>40	no	*	m	yes
<=30	no	*	h	no

<=30	yes	fair	*	no
>40	no	fair	*	no
<=30	no	fair	*	yes
31-40	no	fair	*	yes
<=30	yes	fair	*	yes
>40	yes	excellent	*	no
31-40	no	excellent	*	yes
<=30	yes	fair	*	no
<=30	yes	fair	*	yes
>40	yes	fair	*	yes
31-40	yes	fair	*	yes
>40	no	excellent	*	no
>40	no	fair	*	yes
>40	no	fair	*	yes

```

#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<dos.h>
struct sen
{
char atype[40];
int cnt;
}se[30],temp;
int ta=4,b=0,t=0;
char str[500],str1[500];

```

```

FILE *in,*in1,*out,*in3;
char* method2(int ca,int ti);//median
char* method3(int ca,int ti);//last 10
void method4(int ca,int ti);//high ig
char* method5(int ca,int ti);//average
char* method1(int ca,int ti)//average
{
    int ac=0,k,j,i;
    in=fopen("base.c","r");
    while(!feof(in))
    {
        fscanf(in,"%s",str);
        ac++;
        while(ac!=ca)
        {
            if(feof(in))
                goto sd;
            fscanf(in,"%s",str);
            ac++;
        }
    }

    int j;
    for(j=0;j<=b;j++)
        if(strcmp(se[j].atype,str)==0)
            return j;
    b=++t;
    return b;
} int getindex()
{

void main()
char* method3(int ca,int ti)
{
    int cnt=-1,s,t,ac=0,k,j,i;;
    in=fopen("base.c","r");

```

```

out=fopen("tmp.c","w");
while(!feof(in))
{
    cnt++;
    fgets(str,500,in);
}
fclose(in);
s=cnt-10;
in=fopen("base.c","r");
t=0;
while(t<s)
{
    fgets(str,500,in);
    t++;
}
while(!feof(in))
{
    fgets(str,500,in);
    fputs(str,out);
}
fclose(in);
fclose(out);

```

```

in=fopen("tmp.c","r");
while(!feof(in))
{
    fscanf(in,"%s",str);
    ac++;
    while(ac!=ca)
    {
        if(!feof(in))

temp=se[k];
se[k]=se[j];
se[j]=temp;

```

```
}  
}  
}
```

```
import java.io.*;  
import java.util.*;  
import java.util.Calendar;  
class sen
```

```
{  
    String atype;  
    int cnt;  
}
```

```
class module2 {
```

```
    public static void main (String[] args)
```

```
    {  
        int ca=0,ti=1,ta=4,rs=0,me=0;
```

```
        String dbRecord = null;
```

```
        DataInputStream dis = null;
```

```
        try
```

```
        {
```

```
            module2 obj =new module2();
```

```
            File f = new File("ip4.c");
```

```
            FileInputStream fis = new FileInputStream(f);
```

```
            BufferedInputStream bis = new BufferedInputStream(fis);
```

```
            dis = new DataInputStream(bis);
```

```
            Calendar cal = Calendar.getInstance();
```

```
            long tn1=System.nanoTime( );
```

```
            System.out.println("Current milliseconds are :"
```

```
+ cal.getTimeInMillis()+"nanoseconds"+tn1);
```

```
            while ( (dbRecord = dis.readLine()) != null)
```

```
            {
```



```

StringTokenizer st = new StringTokenizer(dbRecord, " ");
while(st.hasMoreTokens())
{
    String str=st.nextToken();
    ca++;
    if(str.compareTo("*")==0)
    {
        System.out.println("missing value is "+obj.method1(ca,ti,str)+" "+ti);
        if(me==0)
        {
            me=1;
            rs=ca;
        }
    }
    if(ca==ta+1)
    {
        ca=0;ti++;
    }
}

long tn2=System.nanoTime( );
    System.out.println("Current milliseconds are :"+
+ cal.getTimeInMillis()+"nanoseconds"+tn2);
long sx=tn2-tn1;
double sum=(double)sx/1000000000;
System.out.println("difference "+(tn2-tn1)+" cost "+((tn2-tn1)/1000000)+" "+sum);
int tmp=obj.costfix(sum);
System.out.println("final testcost "+tmp);
    obj.wr(tmp,"method1",rs);

}
catch (IOException e)
{
    // catch io errors from FileInputStream or readLine()

```

```

// System.out.println("Uh oh, got an IOException error: " + e.getMessage());

}
finally
{
    // if the file opened okay, make sure we close it
    if (dis != null) {
        try {
            dis.close();
        } catch (IOException ioe) {
            //System.out.println("IOException error trying to close the file: " +
                // e.getMessage());
        }

    } // end if

} // end finally
}
public String method1(int ca,int ti,String str)
{
    String rstr=null;
    sen[] se = new sen[20];
    for (int i = 0; i < se.length; i++)
    {
        se[i] = new sen();
    }
    sen temp=new sen();
    //module2 obj =new module2();
    int ac=0,ta=4,k,j,i,jt,flag=0;
    String dbRecord = null;
    DataInputStream dis = null;
    try
    {
        File f = new File("base.c");
        FileInputStream fis = new FileInputStream(f);

```

```

BufferedInputStream bis = new BufferedInputStream(fis);
dis = new DataInputStream(bis);
for(i=0;i<10;i++)
{
    se[i].atype=null;
    se[i].cnt=0;
}

while ( (dbRecord = dis.readLine()) != null)
{

    StringTokenizer st = new StringTokenizer(dbRecord, " ");
    while(st.hasMoreTokens())
    {
        String str1=st.nextToken();
        ac++;
        while(ac!=ca)
        {
            if(st.hasMoreTokens())
            {
                str1=st.nextToken();
                ac++;
            }
            else
                break;
        }

//System.out.println("hai"+str1+" "+se[1].atype);
for(jt=1;jt<10;jt++)
{

if(se[jt].atype==null && flag!=1)
{
//System.out.println("hai");

```

```

i=jt;
flag=1;
}
else if(se[jt].atype!=null)
{
    if((str1.compareTo(se[jt].atype)==0) && flag!=1)
    {
        //System.out.println("hai");
        i=jt;
        flag=1;
    }
}
}

```

```

    flag=0;
    //i=obj.getindex(str1);

```

```

    se[i].atype=str1;
    se[i].cnt++;
    while(ac<=ta)
    {
        if(st.hasMoreTokens())
        {
            str1=st.nextToken();
            ac++;
        }
        else
            break;
    }
    ac=0;
}

```

```

}
for(k=1;k<=10;k++)
for(j=1;j<=10;j++)

```

```

if(se[k].cnt>se[j].cnt)
{
temp=se[k];
se[k]=se[j];
se[j]=temp;
}
rstr=se[1].atype;
for(i=0;i<10;i++)
{
se[i].atype=null;
se[i].cnt=0;
}
return rstr;
}

catch (IOException e)
{
// catch io errors from FileInputStream or readLine()
// System.out.println("Uh oh, got an IOException error: " + e.getMessage());

}

finally
{
// if the file opened okay, make sure we close it
if (dis != null) {
try {
dis.close();
} catch (IOException ioe) {
//System.out.println("IOException error trying to close the file: " +
// e.getMessage());
}
} // end if

} // end finally

```

```

return rstr;
}
int costfix(double sum)
{
if(sum >0 && sum <0.025)
    return 10;
else if(sum> 0.026 && sum < 0.030)
    return 15;
else if(sum >0.031 && sum<0.040)
    return 20;
else if(sum >0.041 && sum<0.050)
    return 25;

else
    return 30;
}
void wr(int mus,String str,int rs)
{
try{
    // Create file
    FileWriter fstream = new FileWriter("out.txt",true);
    BufferedWriter out = new BufferedWriter(fstream);
    System.out.println("----"+mus);
    out.newLine();
    out.write(str+" "+rs+" "+mus);

    out.flush();
    //Close the output stream
    out.close();
}catch (Exception e){//Catch exception if any
    System.err.println("Error: " + e.getMessage());
}
}

```

```
}
```

```
import java.io.*;
import java.util.*;
import java.util.Calendar;
class sen
{
    String atype;
    int cnt;
    public String m22(int ca,int ti,String str)
    {
        String rstr=null;
        String str1=null;
        sen[] se = new sen[20];
        for (int i = 0; i < se.length; i++)
        {
            se[i] = new sen();
        }
        sen temp=new sen();
        int ac=0,ta=4,k,j,i,cnt=0,jt,flag=0;
        String dbRecord = null;
        DataInputStream dis = null;
        try
        {
            File f = new File("base.c");
            FileInputStream fis = new FileInputStream(f);
            BufferedInputStream bis = new BufferedInputStream(fis);
            dis = new DataInputStream(bis);
            while ( (dbRecord = dis.readLine()) != null)
            {

                cnt++;
            }
        }
    }
}
```

```

    }

}

catch (IOException e)
{
    // catch io errors from FileInputStream or readLine()
    // System.out.println("Uh oh, got an IOException error: " + e.getMessage());

}

finally
{
    // if the file opened okay, make sure we close it
    if (dis != null) {
        try {
            dis.close();
        } catch (IOException ioe) {
            //System.out.println("IOException error trying to close the file: " +
                // e.getMessage());
        }

    } // end if

} // end finally

try
{
    File f = new File("base.c");
    FileInputStream fis = new FileInputStream(f);
    BufferedInputStream bis = new BufferedInputStream(fis);
    dis = new DataInputStream(bis);
    for(i=0;i<10;i++)
    {

```



```

        se[i].atype=null;
        se[i].cnt=0;
    }
    System.out.println("count value "+cnt);
    int s=cnt/2,t=1;
    while(t<s)
    {
        dis.readLine();
        t++;
    }
    ac=0;

    dbRecord=dis.readLine();
    StringTokenizer st = new StringTokenizer(dbRecord, " ");
    while (ac<ca && st.hasMoreTokens())
    {

        str1=st.nextToken();
        ac++;

    }
    System.out.println("count value "+s);
    rstr=str1;

    return rstr;
}

catch (IOException e)
{
    // catch io errors from FileInputStream or readLine()

```

```

        // System.out.println("Uh oh, got an IOException error: " + e.getMessage());

    }
finally
{
    // if the file opened okay, make sure we close it
    if (dis != null) {
        try {
            dis.close();
        } catch (IOException ioe) {
            //System.out.println("IOException error trying to close the file: " +
                // e.getMessage());
        }

    } // end if

} // end finally
//-----

```

```

import java.io.*;
import java.util.*;
import java.util.Calendar;
class sen
{
    String atype;
    int cnt;
public String m4(int ca,int ti,String str,String prev)
{
    String rstr=null;
    String str1=null;
    sen[] se = new sen[20];
    for (int i = 0; i < se.length; i++)
    {
        se[i] = new sen();
    }
}

```

```

sen temp=new sen();
int ac=0,ta=4,k,j,i,cnt=0,jt,flag=0,fla=0;
String dbRecord = null;
DataInputStream dis = null;
try
{
File f = new File("base.c");
FileInputStream fis = new FileInputStream(f);
BufferedInputStream bis = new BufferedInputStream(fis);
dis = new DataInputStream(bis);
for(i=0;i<10;i++)
{
se[i].atype=null;
se[i].cnt=0;
}
//System.out.println("count value "+cnt);
if(ca!=1)
{
while ( (dbRecord = dis.readLine()) != null)
{

StringTokenizer st = new StringTokenizer(dbRecord, " ");
while(st.hasMoreTokens())
{
str1=st.nextToken();
if(str1.compareTo(prev)==0)
fla=1;
ac++;
while(ac!=ca)
{
if(st.hasMoreTokens())
{
str1=st.nextToken();
ac++;
}
}
}
}
}
}

```

```
        else
            break;
    }
}
```

```
//System.out.println("hai"+str1+" "+se[1].atype);
```

```
if(fla==1)
```

```
{
    fla=0;
    for(jt=1;jt<10;jt++)
    {
```

```
        flag=0;
```

```
        //i=obj.getindex(str1);
```

```
        se[i].atype=str1;
```

```
        se[i].cnt++;
```

```
    }
```

```
    while(ac<=ta)
```

```
    {
```

```
        if(st.hasMoreTokens())
```

```
        {
```

```
            str1=st.nextToken();
```

```
            ac++;
```

```
        }
```

```
        else
```

```
            break;
```

```
    }
```

```
    ac=0;
```

```
    }
```

```
    }
```

```
    }
```

```
    for(k=1;k<=10;k++)
```

```
    for(j=1;j<=10;j++)
```

```

if(se[k].cnt>se[j].cnt)
{
temp=se[k];
se[k]=se[j];
se[j]=temp;
}
rstr=se[1].atype;
for(i=0;i<10;i++)
{
se[i].atype=null;
se[i].cnt=0;
}

try
{

File f = new File("ip2.c");
FileInputStream fis = new FileInputStream(f);
BufferedInputStream bis = new BufferedInputStream(fis);
dis = new DataInputStream(bis);
Calendar cal = Calendar.getInstance();
long tn1=System.nanoTime( );
System.out.println("-----method 4-----");
System.out.println("Current milliseconds are :"+
+ cal.getTimeInMillis()+"nanoseconds"+tn1);
while ( (dbRecord = dis.readLine()) != null)
{

StringTokenizer st = new StringTokenizer(dbRecord, " ");
while(st.hasMoreTokens())
{

str=st.nextToken();
if(ca==0)

```

```

        prev=str;
        ca++;
        if(str.compareTo("*")==0)
        {
            System.out.println("missing value is "+obj1.m4(ca,ti,str,prev)+" "+ti);
            if(me==0)
            {
                me=1;
                rs=ca;
            }
        }
        if(ca==ta+1)
        {
            ca=0;ti++;
        }
    }
}

long tn2=System.nanoTime();
    System.out.println("Current milliseconds are :"+
+ cal.getTimeInMillis()+"nanoseconds"+tn2);
long sx=tn2-tn1;
double sum=(double)sx/1000000000;
System.out.println("difference "+(tn2-tn1)+" cost "+((tn2-tn1)/1000000)+" "+sum);
int tmp=obj1.costfix(sum);
System.out.println("final testcost "+tmp);
    obj1.wr(tmp,"method4",rs);
}
catch (IOException e)
{
    // catch io errors from FileInputStream or readLine()
    // System.out.println("Uh oh, got an IOException error: " + e.getMessage());

}
finally
{

```

```
// if the file opened okay, make sure we close it
if (dis != null) {
    try {
        dis.close();
    } catch (IOException ioe) {
        //System.out.println("IOException error trying to close the file: " +
            // e.getMessage());
    }

} // end
}
```

SCREEN SHOTS

INFORMATION GAIN:

```
ca:\code> C:\Python>
information gain-0.693134
information gain-0.918285
information gain-0.737830
information gain-0.821641
total entropy value =0.909745
age      <=20      0.693134      1.016117
income  high      0.918285      0.705403
student no       0.737830      1.016117
credit_rating fair   0.821641      1.016117
-----
age      <=30      0.693134      1.016117
student no       0.918285      1.016117
credit_rating fair   0.737830      0.705403
income  high      0.821641      1.016117
```

MISSING VALUE:

```
ca:\code> C:\Python>
method2: missing value is yes for tuple 1
method2: missing value is yes for tuple 2
method2: missing value is yes for tuple 3
method2: missing value is yes for tuple 4
method2: missing value is yes for tuple 5
method2: missing value is yes for tuple 6
method2: missing value is yes for tuple 7
method2: missing value is yes for tuple 8
method2: missing value is yes for tuple 9
method2: missing value is yes for tuple 10
method2: missing value is yes for tuple 11
method2: missing value is yes for tuple 12
```


TEST COST:

```
C:\Windows\system32\CMD.exe
Note: Recompile with -Xlint:deprecation for details.

H:\new\nou1>java module2
Current milliseconds are 11240892493412nanoseconds13772513688589
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
Current milliseconds are 11240892493412nanoseconds13772513688589
difference 284890322
lines 47
-----25
H:\new\nou1>
```

```
C:\Windows\system32\CMD.exe
Note: module2.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

H:\new\nou1>java module2
Current milliseconds are 11240892493412nanoseconds13772513688589
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
missing value 1s ^133
Current milliseconds are 11240892493412nanoseconds13772513688589
difference 284890322
lines 47
-----25
H:\new\nou1>
```

```
missing value is excellent 9
count value 99
count value 49
missing value is excellent 10
count value 99
count value 49
missing value is excellent 11
count value 99
count value 49
missing value is excellent 12
count value 99
count value 49
missing value is excellent 13
count value 99
count value 49
missing value is excellent 14
count value 99
count value 49
missing value is excellent 15
Current milliseconds are 1124339353388;nanoseconds:103370711.47437
difference 136372963 case 133 2.133372963
final test case 25
----25
H:\new\new1>
```

```
excellent
*
count value 99
missing value is 1 10
no
>48
no
fair
*
count value 99
missing value is 1 10
yes
>48
no
fair
*
count value 99
missing value is 1 11
yes
Current milliseconds are 1124339353377;nanoseconds:103370711.47437
difference 136372963 case 133 2.133372963
final test case 25
----25
H:\new\new1>
```

REFERENCES

- [1] Qiang Yang, Charles Ling, Xiaoyong Chai, and Rong Pan, "Test-Cost Sensitive Classification on Data with Missing Values", IEEE Transactions on Knowledge and Data Engineering, Vol. 18, No. 5, May 2006
- [2] P.D. Turney, "Types of Cost in Inductive Concept Learning" Proceeding Workshop Cost-Sensitive Learning at the 17th Int'l Conf. Machine Learning, 2000.
- [3] Jiawei Han, Micheline Kamber. "Data Mining Concepts and Techniques", Morgan Kaufmann Publishers, 2001.
- [4] C. Ling, Q. Yang, J. Wang, and S. Zhang, "Decision Trees with Minimal Costs," Proceedings of the 21st International Conference on Machine Learning (ICML), Banff, Canada, 2004
- [5] P. Domingos, "Metacost: A General Method for Making Classifiers Cost-Sensitive," Knowledge Discovery and Data Mining, pp. 155-164, 1999.
- [6] M.T. Kai, "Inducing Cost-Sensitive Trees Via Instance Weighting" Principles of Data Mining and Knowledge Discovery, Second European Symposium., pp. 139-147, 1998.
- [7] M. Nunez, "The Use of Background Knowledge in Decision Tree Induction" Machine Learning, vol. 6, pp. 231-250, 1991.