Decision Tree Classifier with GA based feature selection

Mini Project Report

Submitted to

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

By

Samir Sheriff

Satvik N

In partial fulfilment of the requirements

for the award of the degree

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

R V College of Engineering

(Autonomous Institute, Affiliated to VTU)

BANGALORE - 560059

May 2012

DECLARATION

We, Samir Sheriff and Satvik N bearing USN number 1RV09CS093 and 1RV09CS095 respectively, hereby declare that the dissertation entitled "**Decision Tree Classifier with GA feature selection**" completed and written by us, has not been previously formed the basis for the award of any degree or diploma or certificate of any other University.

Bangalore

Samir Sheriff
USN:1RV09CS093

Satvik N
USN:1RV09CS095

R V COLLEGE OF ENGINEERING

(Autonomous Institute Affiliated to VTU)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled, "**Decision Tree Classifier with GA based feature selection**", which is being submitted herewith for the award of B.E is the result of the work completed by **Samir Sheriff and Satvik N** under my supervision and guidance.

Signature of Guide
(Name of the Guide)

Signature of Head of Department	Signature of Principal
(Dr. N K Srinath)	(Dr. B.S Sathyanarayana)

Name of Examiner	Signature of Examiner
1:	
2:	

ACKNOWLEDGEMENT

The euphoria and satisfaction of the completion of the project will be incomplete without thanking the persons responsible for this venture.

We acknowledge RVCE (Autonomous under VTU) for providing an opportunity to create a mini-project in the 5th semester. We express our gratitude towards **Prof. B.S. Satyanarayana**, principal, R.V.C.E for constant encouragement and facilitates extended in completion of this project. We would like to thank **Prof. N.K.Srinath**, HOD, CSE Dept. for providing excellent lab facilites for the completion of the project. We would personally like to thank our project guides **Ms. Shantha Rangaswamy and Dr.** and also the lab in charge, for providing timely assistance & guidance at the time.

We are indebted to the co-operation given by the lab administrators and lab assistants, who have played a major role in bringing out the mini-project in the present form. Bangalore

Samir Sheriff
6th semester, CSE
USN:1RV09CS093
Satvik N
6th semester, CSE
USN:1RV09CS095

ABSTRACT

Machine Learning techniques have been applied to the field of classification for more than a decade. Machine Learning techniques can learn normal and anomalous patterns from training data and generate classifiers, which can be used to capture characteristics of interest. In general, the input data to classifiers is an extremely large set of features, but not all of features are relevant to the classes to be classified. Hence, the learner must generalize from the given examples in order to produce a useful output in new cases.

A major focus of machine learning research is the design of algorithms that recognize complex patterns and make intelligent decisions based on input data. Our Project, titled "**Decision Tree Classifier with Genetic Algorithm-based Feature Selection** is aimed at developing a complete program that constructs an optimal decision tree, based on any kind of data set, divided into training and testing examples, by selecting only a subset of features to classify data.

Although our program works with generic data samples, it must be noted that when we started this project, our main intention was to classify ground water samples into two classes, namely Potable and Non-Potable Water. However, thanks to the miracle of Object-Oriented Programming Concepts, we were able to extend this project.

Contents

Chapter 1

INTRODUCTION

Machine learning, a branch of artificial intelligence, is about the construction and study of systems that can learn from data. The core of machine learning deals with representation and generalization. Representation of data instances and functions evaluated on these instances are part of all machine learning systems. There is a wide variety of machine learning tasks and successful applications.

1.1 SCOPE

The machine learning concepts we have used in our project are listed below,

- **Supervised learning** is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which is called a classifier (if the output is discrete; see classification) or a regression function (if the output is continuous; see regression). The inferred function should predict the correct output value for any valid input object. This requires the learning algorithm to generalize from the training data to unseen situations in a ”reasonable” way.

- **Decision tree learning**, used in statistics, data mining and machine learning, uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. The goal is to create a model that predicts the value of a target variable based on several input variables.

- A **Genetic Algorithms** is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

Chapter 2

REQUIREMENT SPECIFICATION

Software Requirement Specification (SRS) is an important part of the software development process. We describe the overall description of the Mini-Project, the specific requirements of the Mini-Project, the software requirements and hardware requirements and the functionality of the system.

Software Requirements

- Front End: Java SWT Application.
- Back End: Java
- Operating System: Windows 7, Ubuntu 12.10.

Hardware Requirements

- Processor: Intel Core 2 Duo or higher version
- RAM: 4GB or more
- Hard disk: 5 GB or less

Chapter 3

Decision Tree Learning

3.1 A Naive Approach

3.2 Definition

Decision tree is the learning of decision tree from class labeled training tuples. A decision tree is a flow chart like structure, where each internal (non-leaf) node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf (or terminal) node holds a class label. The topmost node in tree is the root node.

There are many specific decision-tree algorithms. Notable ones include:

- **ID3** (Iterative Dichotomiser 3)
- **C4.5** algorithm, successor of ID3
- **CART** (Classification And Regression Tree)
- **CHi-squared Automatic Interaction Detector** (CHAID). Performs multi-level splits when computing classification trees.
- **MARS**: extends decision trees to better handle numerical data

3.3 The Basic Idea

Decision tree is a classifier in the form of a tree structure (as shown in Fig. 3.1, where each node is either:

1. A **leaf node** - indicates the value of the target attribute (class) of examples (*In Fig. 3.1, the nodes containing values K=x, K=y)*

2. A **decision node** - specifies some test to be carried out on a single attribute-value, with one branch and sub-tree for each possible outcome of the test.*In Fig. 3.1, the nodes containing attributes A, B and C)*

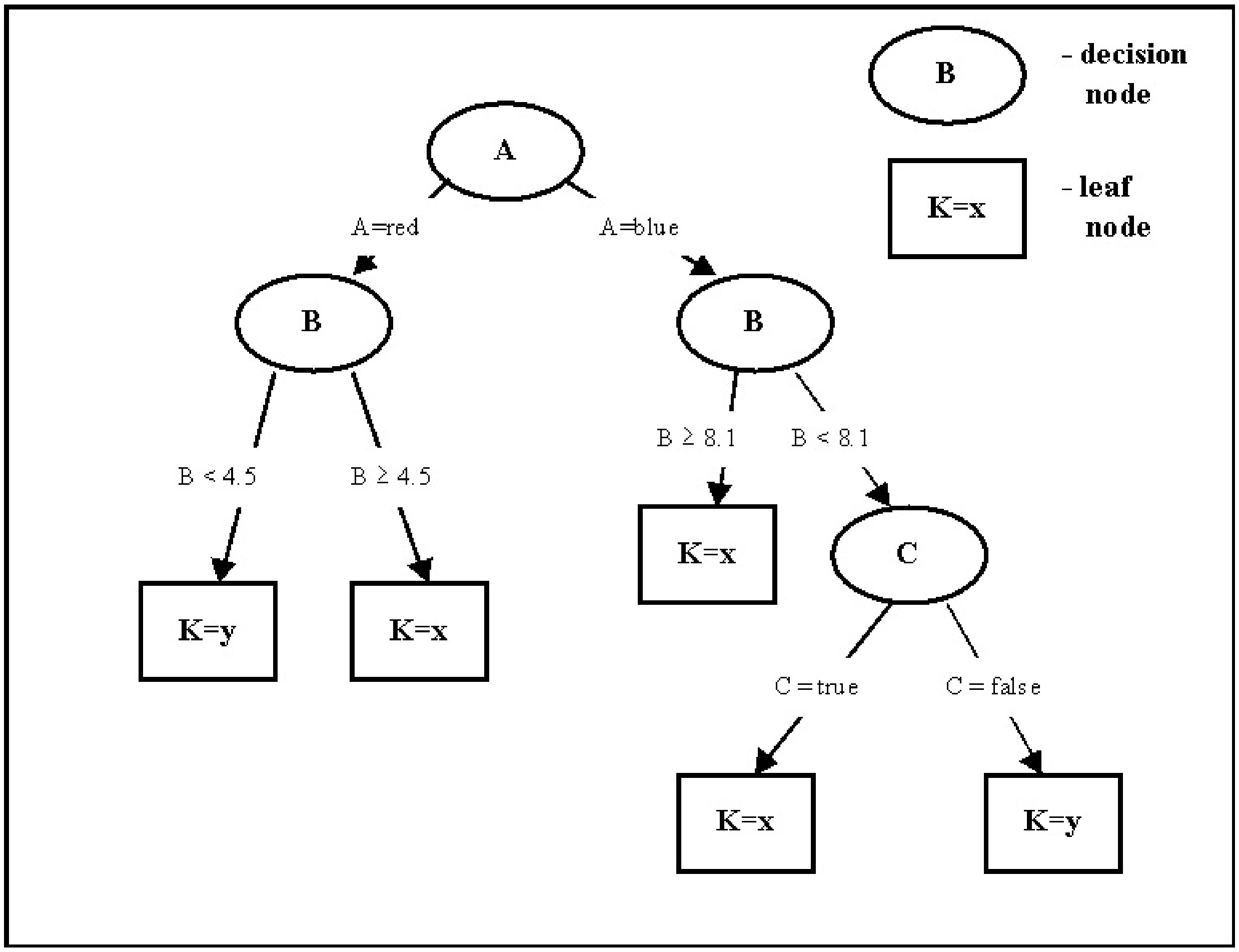


Figure 3.1: Sample Decision Tree

A decision tree can be used to classify an example by starting at the root of the tree and moving through it until a leaf node, which provides the classification of the instance.

Decision tree induction is a typical inductive approach to learn knowledge on classification. The key requirements to do mining with decision trees are:

- **Attribute-value description**: object or case must be expressible in terms of a fixed collection of properties or attributes. This means that we need to discretize continuous attributes, or this must have been provided in the algorithm.*(A, B and C, in Fig. 3.1)*
- **Predefined classes (target attribute values)**: The categories to which examples are to be assigned must have been established beforehand (supervised data) *(Classes X and Y in Fig. 3.1)*.
- **Discrete classes**: A case does or does not belong to a particular class, and there must be more cases than classes.
- **Sufficient data**: Usually hundreds or even thousands of training cases.

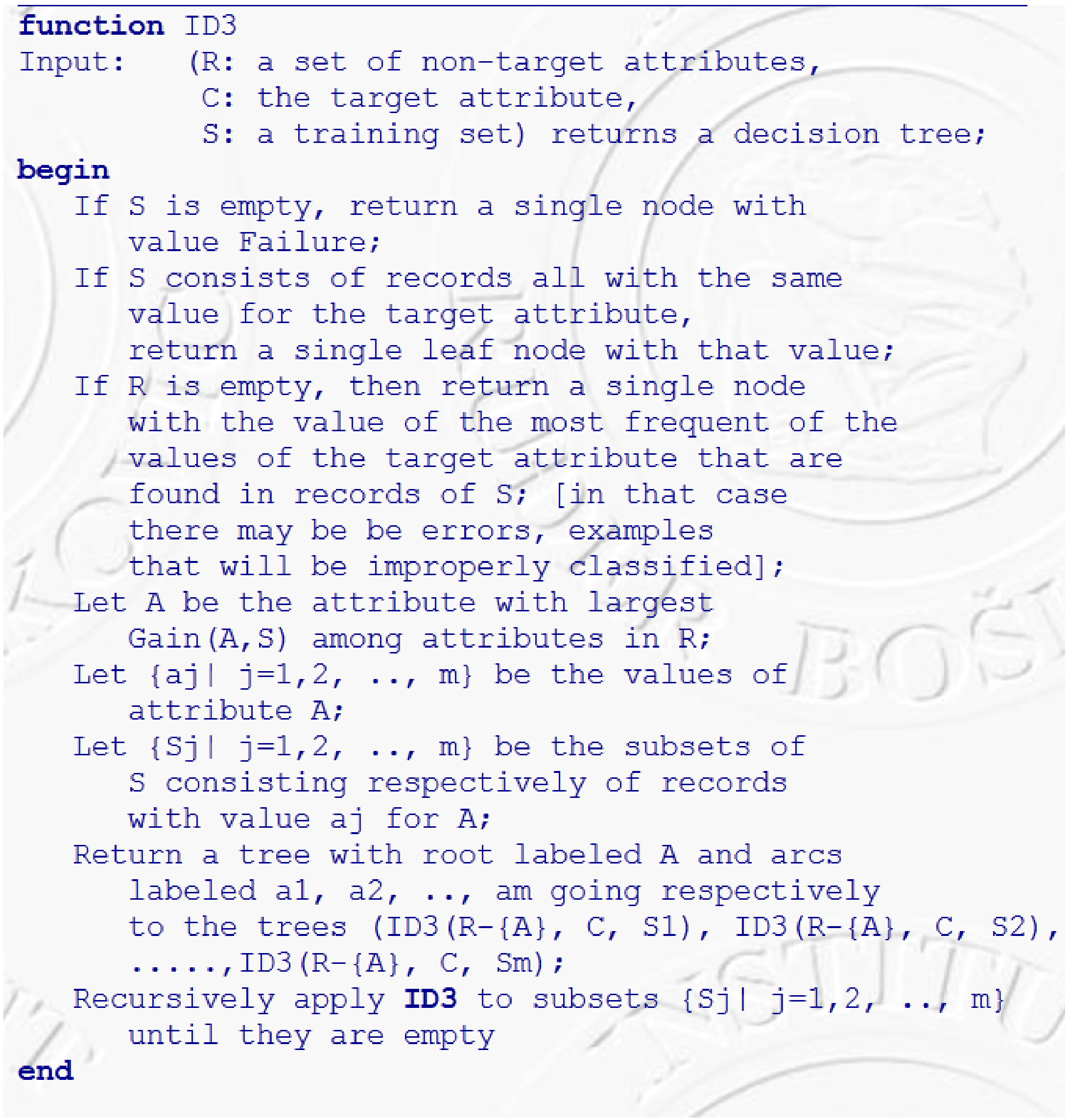
3.4 Building the Decision Tree

Most algorithms that have been developed for learning decision trees are variations on a core algorithm that employs a top-down, greedy search through the space of possible decision trees. Decision tree programs construct a decision tree T from a set of training cases.

3.4.1 ID3 Algorithm

J. Ross Quinlan originally developed ID3 at the University of Sydney. He first presented ID3 in 1975 in a book, Machine Learning, vol. 1, no. 1. ID3 is based on the Concept Learning System (CLS) algorithm. ID3 searches through the attributes of the training instances and extracts the attribute that best separates the given examples. If the attribute perfectly classifies the training sets then ID3 stops; otherwise it recursively operates on the m (where m = number of possible values of an attribute) partitioned subsets to get their

”best” attribute. The algorithm uses a greedy search, that is, it picks the best attribute and never looks back to reconsider earlier choices. Note that ID3 may misclassify data.



```
function ID3
Input:   (R: a set of non-target attributes,
          C: the target attribute,
          S: a training set) returns a decision tree;
begin
   If S is empty, return a single node with
      value Failure;
   If S consists of records all with the same
      value for the target attribute,
      return a single leaf node with that value;
   If R is empty, then return a single node
      with the value of the most frequent of the
      values of the target attribute that are
      found in records of S; [in that case
      there may be be errors, examples
      that will be improperly classified];
   Let A be the attribute with largest
      Gain(A,S) among attributes in R;
   Let {aj| j=1,2, .., m} be the values of
      attribute A;
   Let {Sj| j=1,2, .., m} be the subsets of
      S consisting respectively of records
      with value aj for A;
   Return a tree with root labeled A and arcs
      labeled a1, a2, .., am going respectively
      to the trees (ID3(R-{A}, C, S1), ID3(R-{A}, C, S2),
      .....,ID3(R-{A}, C, Sm);
   Recursively apply ID3 to subsets {Sj| j=1,2, .., m}
      until they are empty
end
```

Figure 3.2: ID3 Algorithm

3.4.2 Choosing the best attribute for a given node

The estimation criterion in the decision tree algorithm is the selection of an attribute to test at each decision node in the tree. The goal is to select the attribute that is most useful for classifying examples. A good quantitative measure of the worth of an attribute is a statistical property called information gain that measures how well a given attribute separates the training examples according to their target classification. This measure is used to select among the candidate attributes at each step while growing the tree.

3.4.3 Entropy - a measure of homogeneity of the set of examples

In order to define information gain precisely, we need to define a measure commonly used in information theory, called entropy, that characterizes the (im)purity of an arbitrary collection of examples. Given a set S, containing only positive and negative examples of some target concept (a 2 class problem), the entropy of set S relative to this simple, binary classification is defined as:

$Entropy(S) = -p_p log_2 p_p - p_n log_2 p_n$

where p_p is the proportion of positive examples in S and p_n is the proportion of negative examples in S. In all calculations involving entropy we define 0 log 0 to be 0.

To illustrate, suppose S is a collection of 25 examples, including 15 positive and 10 negative examples [15+, 10-]. Then the entropy of S relative to this classification is

$Entropy(S) = -(15/25) log_2(15/25) - (10/25) log_2(10/25) = 0.970$

Notice that the entropy is 0 if all members of S belong to the same class. For example, if all members are positive (p_p= 1), then p_n is 0, and:

$Entropy(S) = -1 log_2(1) - 0 log_2 0 = -10 - 0 log_2 0 = 0.$

Note the entropy is 1 (at its maximum!) when the collection contains an equal number of positive and negative examples. If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1. Figure 3.3 shows the form of the entropy function relative to a binary classification, as p+ varies between 0 and 1.

One interpretation of entropy from information theory is that it specifies the minimum number of bits of information needed to encode the classification of an arbitrary member of S (i.e., a member of S drawn at random with uniform probability). For example, if p_p is 1, the receiver knows the drawn example will be positive, so no message need be sent, and the entropy is 0. On the other hand, if p_p is 0.5, one bit is required to indicate whether the drawn example is positive or negative. If p_p is 0.8, then a collection of messages can be encoded using on average less than 1 bit per message by assigning shorter codes to

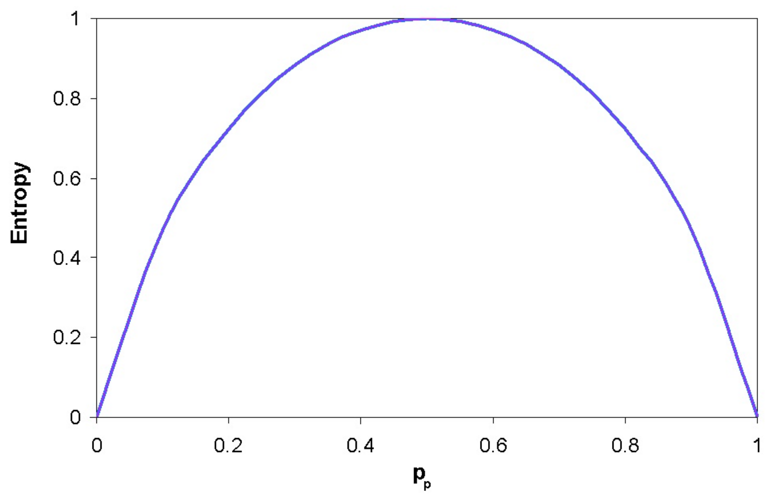


Figure 3.3: The entropy function relative to a binary classification, as the proportion of positive examples p_p varies between 0 and 1.

collections of positive examples and longer codes to less likely negative examples.

Thus far we have discussed entropy in the special case where the target classification is binary. If the target attribute takes on c different values, then the entropy of S relative to this c-wise classification is defined as:

$Entropy(S) = \sum_1^c -p_i log_2 p_i$

where p_i is the proportion of S belonging to class i. Note the logarithm is still base 2 because entropy is a measure of the expected encoding length measured in bits. Note also that if the target attribute can take on c possible values, the maximum possible entropy is $\log_2$c.

3.4.4 Information gain measures the expected reduction in entropy

Given entropy as a measure of the impurity in a collection of training examples, we can now define a measure of the effectiveness of an attribute in classifying the training data. The measure we will use, called information gain, is simply the expected reduction in entropy caused by partitioning the examples according to this attribute. More precisely, the information gain, Gain (S, A) of an attribute A, relative to a collection of examples S, is defined as:

$Gain(S, A) = Entropy(S) - \sum_{v \in Value(A)} \frac{S_v}{S} Entropy(S_v)$

where Values(A) is the set of all possible values for attribute A, and S_v is the subset of S for which attribute A has value v (i.e., S_v = s S — A(s) = v). Note the first term in the equation for Gain is just the entropy of the original collection S and the second term is the expected value of the entropy after S is partitioned using attribute A. The expected entropy described by this second term is simply the sum of the entropies of each subset S_v, weighted by the fraction of examples $|S_v|/|S|$ that belong to S_v. Gain (S,A) is therefore the expected reduction in entropy caused by knowing the value of attribute A. Put another way, Gain(S,A) is the information provided about the target attribute value, given the value of some other attribute A. The value of Gain(S,A) is the number of bits saved when encoding the target value of an arbitrary member of S, by knowing the value of attribute A.

The process of selecting a new attribute and partitioning the training examples is now repeated for each non-terminal descendant node, this time using only the training examples associated with that node. Attributes that have been incorporated higher in the tree are excluded, so that any given attribute can appear at most once along any path through the tree. This process continues for each new leaf node until either of two conditions is met:

1. Every attribute has already been included along this path through the tree

2. The training examples associated with this leaf node all have the same target attribute value (i.e., their entropy is zero).

3.5 An Example

3.5.1 An Example: "go go gophers"

3.5.2 Example Encoding Table

The character encoding induced by the last tree is shown below where again, 0 is used for left edges and 1 for right edges.

3.5.3 Encoded String

The string "go go gophers" would be encoded as shown (with spaces used for easier reading, the spaces wouldn't appear in the real encoding). **00 01 100 00 01 100 00 01 1110 1101 101 1111 1100**

Once again, 37 bits are used to encode "go go gophers". There are several trees that yield an optimal 37-bit encoding of "go go gophers". The tree that actually results from a programmed implementation of Huffman's algorithm will be the same each time the program is run for the same weights (assuming no randomness is used in creating the tree).

Chapter 4

Genetic Algorithms

Generally speaking, the process of decompression is simply a matter of translating the stream of prefix codes to individual byte values, usually by traversing the Huffman tree node by node as each bit is read from the input stream (reaching a leaf node necessarily terminates the search for that particular byte value). Before this can take place, however, the Huffman tree must be somehow reconstructed.

4.1 Storing the Huffman Tree

- In the simplest case, where character frequencies are fairly predictable, the tree can be preconstructed (and even statistically adjusted on each compression cycle) and thus reused every time, at the expense of at least some measure of compression efficiency.

- Otherwise, the information to reconstruct the tree must be sent a priori.

- A naive approach might be to prepend the frequency count of each character to the compression stream. Unfortunately, the overhead in such a case could amount to several kilobytes, so this method has little practical use.

- Another method is to simply prepend the Huffman tree, bit by bit, to the output stream. For example, assuming that the value of 0 represents a parent node and 1 a leaf node, whenever the latter is encountered the tree building routine simply reads the next 8 bits to determine the character value of that particular leaf. The process continues recursively until the last leaf node is reached; at that point, the Huffman tree will thus be faithfully reconstructed. The overhead using such a method ranges from roughly 2 to 320 bytes (assuming an 8-bit alphabet).

Many other techniques are possible as well. In any case, since the compressed data can include unused ”trailing bits” the decompressor must be able to determine when to stop producing output. This can be accomplished by either transmitting the length of the decompressed data along with the compression model or by defining a special code symbol to signify the end of input (the latter method can adversely affect code length optimality, however).

4.2 Creating the Huffman Table

To create a table or map of coded bit values for each character you'll need to traverse the Huffman tree (e.g., inorder, preorder, etc.) making an entry in the table each time you reach a leaf. For example, if you reach a leaf that stores the character 'C', following a path left-left-right-right-left, then an entry in the 'C'-th location of the map should be set to 00110. You'll need to make a decision about how to store the bit patterns in the map. At least two methods are possible for implementing what could be a class/struct BitPattern:

- Use a string. This makes it easy to add a character (using +) to a string during tree traversal and makes it possible to use string as BitPattern. Your program may be slow because appending characters to a string (in creating the bit pattern) and accessing characters in a string (in writing 0's or 1's when compressing) is slower than the next approach.
- Alternatively you can store an integer for the bitwise coding of a character. You need to store the length of the code too to differentiate between 01001 and 00101. However, using an int restricts root-to-leaf paths to be at most 32 edges long since

an int holds 32 bits. In a pathological file, a Huffman tree could have a root-to-leaf path of over 100. Because of this problem, you should use strings to store paths rather than ints. A slow correct program is better than a fast incorrect program.

4.3 Storing Sizes

The operating system will buffer output, i.e., output to disk actually occurs when some internal buffer is full. In particular, it is not possible to write just one single bit to a file, all output is actually done in "chunks", e.g., it might be done in eight-bit chunks. In any case, when you write 3 bits, then 2 bits, then 10 bits, all the bits are eventually written, but you can not be sure precisely when they're written during the execution of your program. Also, because of buffering, if all output is done in eight-bit chunks and your program writes exactly 61 bits explicitly, then 3 extra bits will be written so that the number of bits written is a multiple of eight. Because of the potential for the existence of these "extra" bits when reading one bit at a time, you cannot simply read bits until there are no more left since your program might then read the extra bits written due to buffering. This means that when reading a compressed file, you CANNOT use code like this.

```
int bits;
while (input.readbits(1, bits))
{
// process bits
}
```

To avoid this problem, you can write the size of a data structure before writing the data structure to the file.

Chapter 5

CONCLUSION AND FUTURE WORKS

Summary

Limitations

1. Huffman code is optimal only if exact probability distribution of the source symbols is known.
2. Each symbol is encoded with integer number of bits.
3. Huffman coding is not efficient to adapt with the changing source statistics.
4. The length of the codes of the least probable symbol could be very large to store into a single word or basic storage unit in a computing system.

Further enhancements The huffman coding the we have considered is simple binary Huffman codingbut many variations of Huffman coding exist,

1. **n-ary Huffman coding:** The n-ary Huffman algorithm uses the {0, 1, ... , n 1} alphabet to encode message and build an n-ary tree. This approach was considered by Huffman in his original paper. The same algorithm applies as for binary (n equals 2)codes, except that the n least probable symbols are taken together, instead of just the 2 least probable. Note that for n greater than 2, not all sets of source words

can properly form an n-ary tree for Huffman coding. In this case, additional 0-probability place holders must be added. If the number of source words is congruent to 1 modulo n-1, then the set of source words will form a proper Huffman tree.

2. **Adaptive Huffman coding:** A variation called adaptive Huffman coding calculates the probabilities dynamically based on recent actual frequencies in the source string. This is some what related to the LZ family of algorithms.

3. **Huffman template algorithm:** Most often, the weights used in implementations of Huffman coding represent numeric probabilities, but the algorithm given above does not require this; it requires only a way to order weights and to add them. The Huffman template algorithm enables one to use any kind of weights (costs,frequencies etc)

4. **Length-limited Huffman coding:** Length-limited Huffman coding is a variant where the goal is still to achieve a minimum weighted path length, but there is an additional restriction that the length of each codeword must be less than a given constant. The package-merge algorithm solves this problem with a simple greedy approach very similar to that used by Huffman's algorithm. Its time complexity is O(nL),where L is the maximum length of a codeword. No algorithm is known to solve this problem in linear or linear logarithmic time, unlike the presorted and unsorted conventional Huffman problems, respectively.

Bibliography

[1] Genetic Algorithms -`http://en.wikipedia.org/wiki/Genetic_algorithm`

[2] Genetic Algorithm for constructing DT - `http://www.jprr.org/index.php/jprr/article/viewFile/44/25`

[3] Project brief for the DT using Horse data sets - `https://cs.uwaterloo.ca/~ppoupart/teaching/cs486-spring06/assignments/asst4/asst4.pdf`

[4] Supervised and Unsupervised Discretization of Continous Features - `http://robotics.stanford.edu/users/sahami/papers-dir/disc.pdf`

[5] Hybrid learning using Genetic Algorithms and Decision Trees for pattern sification - `http://cs.gmu.edu/~eclab/papers/ijcai95.pdf`

[6] Kardi Maams' Tutorials - `http://people.revoledu.com/kardi/tutorial/DecisionTree/index.html`

Appendices

Appendix A : Source Code

Appendix B : Screen Shots

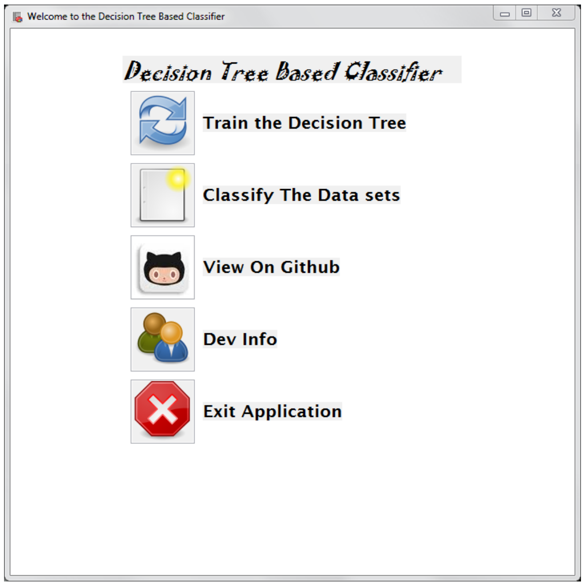
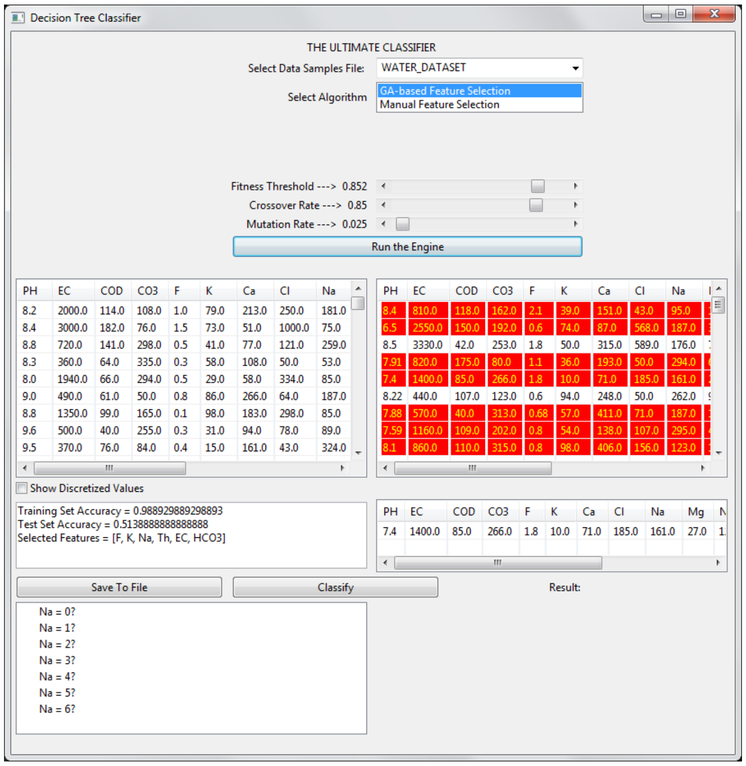


Figure 5.1: Application Window - Welcome Screen



Select Data Samples File: WATER_DATASET

Select Algorithm [✓ GA-based Feature Selection] Manual Feature Selection

Fitness Threshold ---> 0.852
Crossover Rate ---> 0.85
Mutation Rate ---> 0.025

PH	EC	COD	CO3	F	K	Ca	Cl	Na
8.2	2000.0	114.0	108.0	1.0	79.0	213.0	250.0	181.0
8.4	3000.0	182.0	76.0	1.5	73.0	51.0	1000.0	75.0
8.8	720.0	141.0	298.0	0.5	41.0	77.0	121.0	259.0
8.3	360.0	64.0	335.0	0.3	58.0	108.0	50.0	53.0
8.0	1940.0	66.0	294.0	0.5	29.0	58.0	334.0	85.0
9.0	490.0	61.0	50.0	0.8	86.0	266.0	64.0	187.0
8.8	1350.0	99.0	165.0	0.1	98.0	183.0	298.0	85.0
9.6	500.0	40.0	255.0	0.3	31.0	94.0	78.0	89.0
9.5	370.0	76.0	84.0	0.4	15.0	161.0	43.0	324.0

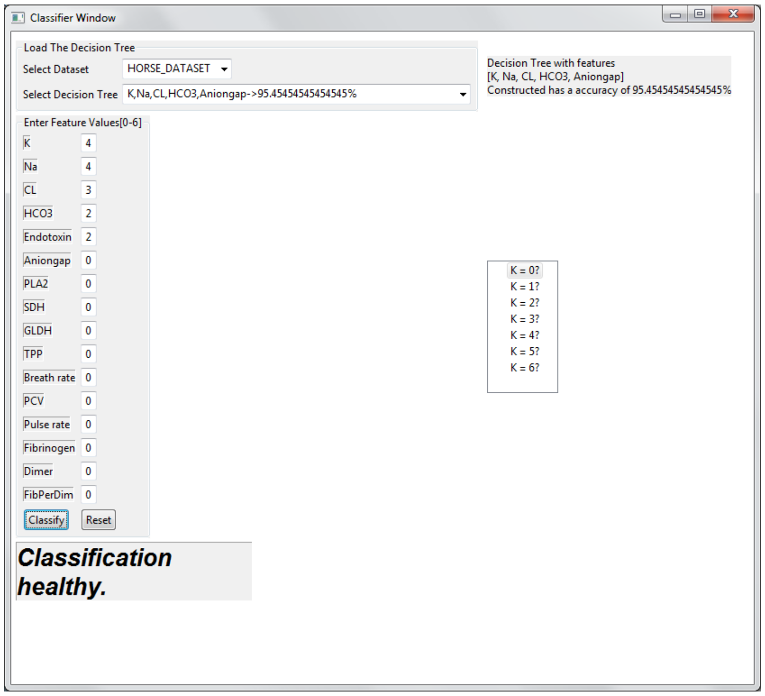
PH	EC	COD	CO3	F	K	Ca	Cl	Na
8.4	810.0	118.0	162.0	2.1	39.0	151.0	43.0	95.0
6.5	2550.0	150.0	192.0	0.6	74.0	87.0	568.0	187.0
8.5	3330.0	42.0	253.0	1.8	50.0	315.0	589.0	176.0
7.91	820.0	175.0	80.0	1.1	36.0	193.0	50.0	294.0
7.4	1400.0	85.0	266.0	1.8	10.0	71.0	185.0	161.0
8.22	440.0	107.0	123.0	0.6	94.0	248.0	50.0	262.0
7.88	570.0	40.0	313.0	0.68	57.0	411.0	71.0	187.0
7.59	1160.0	109.0	202.0	0.8	54.0	138.0	107.0	295.0
8.1	860.0	110.0	315.0	0.8	98.0	406.0	156.0	123.0

```
Training Set Accuracy = 0.988929889298893
Test Set Accuracy = 0.5138888888888888
Selected Features = [F, K, Na, Th, EC, HCO3]
```

PH	EC	COD	CO3	F	K	Ca	Cl	Na	Mg
7.4	1400.0	85.0	266.0	1.8	10.0	71.0	185.0	161.0	27.0

```
    Na = 0?
    Na = 1?
    Na = 2?
    Na = 3?
    Na = 4?
    Na = 5?
    Na = 6?
```

Figure 5.2: Decision Tree Constructor Window.



Select Dataset HORSE_DATASET

Select Decision Tree K,Na,CL,HCO3,Aniongap->95.45454545454545%

Decision Tree with features
[K, Na, CL, HCO3, Aniongap]
Constructed has a accuracy of 95.45454545454545%

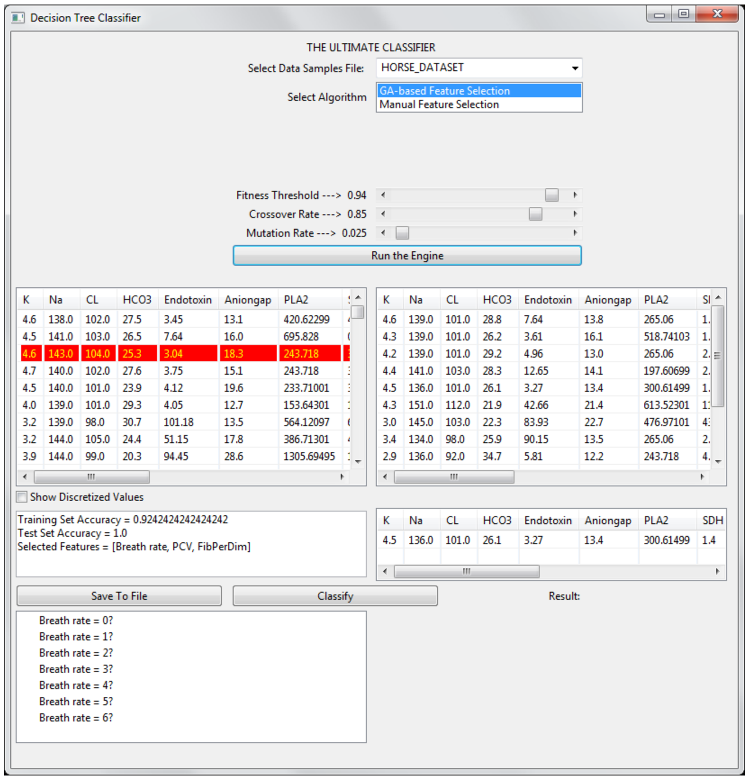
Enter Feature Values[0-6]

K	4
Na	4
CL	3
HCO3	2
Endotoxin	2
Aniongap	0
PLA2	0
SDH	0
GLDH	0
TPP	0
Breath rate	0
PCV	0
Pulse rate	0
Fibrinogen	0
Dimer	0
FibPerDim	0

```
K = 0?
K = 1?
K = 2?
K = 3?
K = 4?
K = 5?
K = 6?
```

Classification healthy.

Figure 5.3: Decision Tree Classifier Window.



Select Data Samples File: HORSE_DATASET

Select Algorithm: GA-based Feature Selection / Manual Feature Selection

Fitness Threshold ---> 0.94

Crossover Rate ---> 0.85

Mutation Rate ---> 0.025

K	Na	CL	HCO3	Endotoxin	Aniongap	PLA2
4.6	138.0	102.0	27.5	3.45	13.1	420.62299
4.5	141.0	103.0	26.5	7.64	16.0	695.828
4.6	143.0	104.0	25.3	3.04	18.3	243.718
4.7	140.0	102.0	27.6	3.75	15.1	243.718
4.5	140.0	101.0	23.9	4.12	19.6	233.71001
4.0	139.0	101.0	29.3	4.05	12.7	153.64301
3.2	139.0	98.0	30.7	101.18	13.5	564.12097
3.2	144.0	105.0	24.4	51.15	17.8	386.71301
3.9	144.0	99.0	20.3	94.45	28.6	1305.69495

K	Na	CL	HCO3	Endotoxin	Aniongap	PLA2	SI
4.6	139.0	101.0	28.8	7.64	13.8	265.06	1.
4.3	139.0	101.0	26.2	3.61	16.1	518.74103	1.
4.2	139.0	101.0	29.2	4.96	13.0	265.06	2.
4.4	141.0	103.0	28.3	12.65	14.1	197.60699	2.
4.5	136.0	101.0	26.1	3.27	13.4	300.61499	1.
4.3	151.0	112.0	21.9	42.66	21.4	613.52301	1
3.0	145.0	103.0	22.3	83.93	22.7	476.97101	4
3.4	134.0	98.0	25.9	90.15	13.5	265.06	2.
2.9	136.0	92.0	34.7	5.81	12.2	243.718	4.

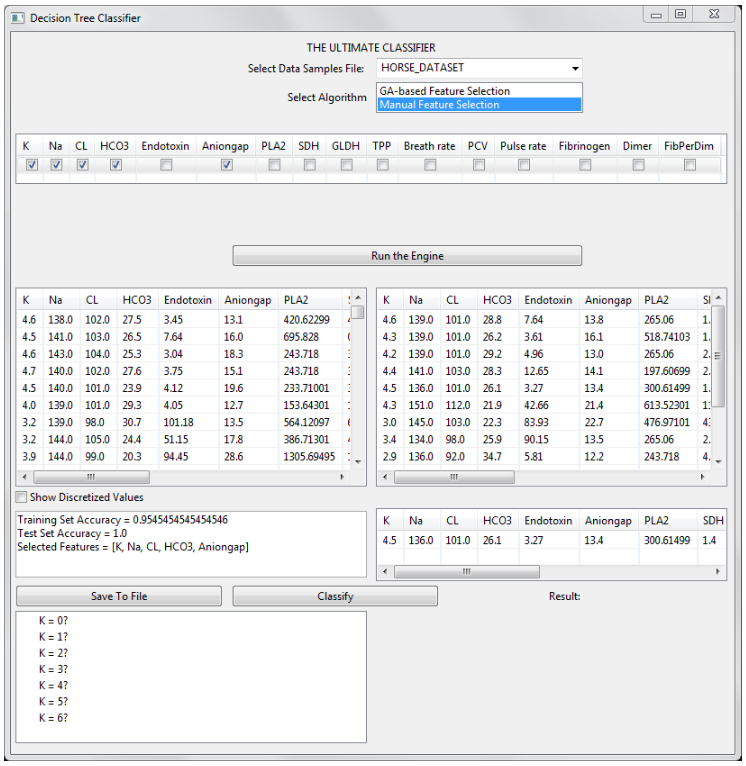
```
Training Set Accuracy = 0.9242424242424242
Test Set Accuracy = 1.0
Selected Features = [Breath rate, PCV, FibPerDim]
```

K	Na	CL	HCO3	Endotoxin	Aniongap	PLA2	SDH
4.5	136.0	101.0	26.1	3.27	13.4	300.61499	1.4

Result:

```
Breath rate = 0?
Breath rate = 1?
Breath rate = 2?
Breath rate = 3?
Breath rate = 4?
Breath rate = 5?
Breath rate = 6?
```

Figure 5.4: Decision Tree Construction with GA based Feature Selector.



K	Na	CL	HCO3	Endotoxin	Aniongap	PLA2	SDH	GLDH	TPP	Breath rate	PCV	Pulse rate	Fibrinogen	Dimer	FibPerDim
✓	✓	✓	✓	☐	✓	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

K	Na	CL	HCO3	Endotoxin	Aniongap	PLA2
4.6	138.0	102.0	27.5	3.45	13.1	420.62299
4.5	141.0	103.0	26.5	7.64	16.0	695.828
4.6	143.0	104.0	25.3	3.04	18.3	243.718
4.7	140.0	102.0	27.6	3.75	15.1	243.718
4.5	140.0	101.0	23.9	4.12	19.6	233.71001
4.0	139.0	101.0	29.3	4.05	12.7	153.64301
3.2	139.0	98.0	30.7	101.18	13.5	564.12097
3.2	144.0	105.0	24.4	51.15	17.8	386.71301
3.9	144.0	99.0	20.3	94.45	28.6	1305.69495

K	Na	CL	HCO3	Endotoxin	Aniongap	PLA2	SI
4.6	139.0	101.0	28.8	7.64	13.8	265.06	1.
4.3	139.0	101.0	26.2	3.61	16.1	518.74103	1.
4.2	139.0	101.0	29.2	4.96	13.0	265.06	2.
4.4	141.0	103.0	28.3	12.65	14.1	197.60699	2.
4.5	136.0	101.0	26.1	3.27	13.4	300.61499	1.
4.3	151.0	112.0	21.9	42.66	21.4	613.52301	1
3.0	145.0	103.0	22.3	83.93	22.7	476.97101	4
3.4	134.0	98.0	25.9	90.15	13.5	265.06	2.
2.9	136.0	92.0	34.7	5.81	12.2	243.718	4.

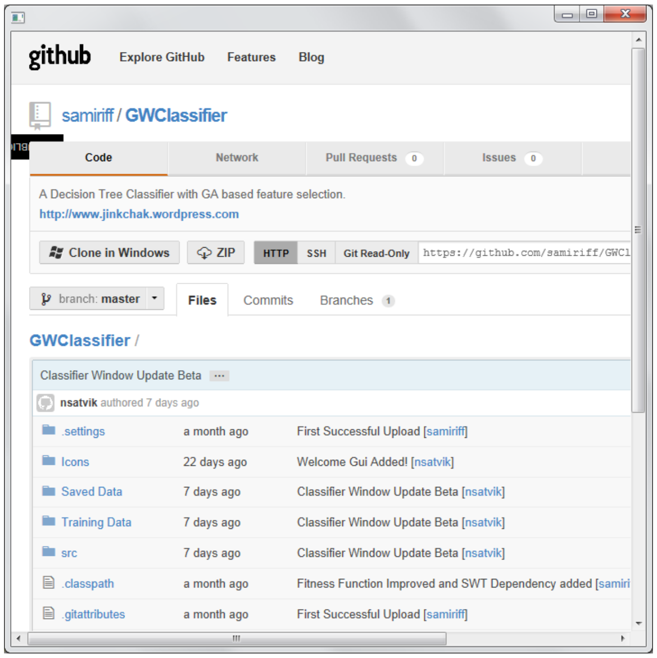
☐ Show Discretized Values

```
Training Set Accuracy = 0.9545454545454546
Test Set Accuracy = 1.0
Selected Features = [K, Na, CL, HCO3, Aniongap]
```

K	Na	CL	HCO3	Endotoxin	Aniongap	PLA2	SDH
4.5	136.0	101.0	26.1	3.27	13.4	300.61499	1.4

```
K = 0?
K = 1?
K = 2?
K = 3?
K = 4?
K = 5?
K = 6?
```

Figure 5.5: Decision Tree Construction with manual feature selection.



A Decision Tree Classifier with GA based feature selection.
http://www.jinkchak.wordpress.com

https://github.com/samiriff/GWCl

Classifier Window Update Beta

nsatvik authored 7 days ago

.settings	a month ago	First Successful Upload [samiriff]
Icons	22 days ago	Welcome Gui Added! [nsatvik]
Saved Data	7 days ago	Classifier Window Update Beta [nsatvik]
Training Data	7 days ago	Classifier Window Update Beta [nsatvik]
src	7 days ago	Classifier Window Update Beta [nsatvik]
.classpath	a month ago	Fitness Function Improved and SWT Dependency added [samiri
.gitattributes	a month ago	First Successful Upload [samiriff]

Figure 5.6: The project source code on github public repository.