Ryan Barker
ECE 4420
11/12/2015

Takehome 3 Proposal

Section I. Project Topic

For Takehome 3, the engineer will develop a complete implementation of ID3 in C. The implementation will come in both a modular and recursive format, and the software package will consist of a makefile and four different C files: *ID3.c*, *ID3.h*, *ID3test.c*, and *ID3debug.c* (See section IV of this report for descriptions of each file in the package). Once ID3 is working, the engineer will expand into analysis of the algorithm by first creating a function to validate an ID3 tree given the training set as discussed in *Intelligent Systems: Principles, Paradigms, and Pragmatics [1]*. He will then address the issues of conflicting data sets in his Takehome 3 report. These problems are described in *Intelligent Systems: Principles, Paradigms, and Pragmatics [1]* section 16.4.10, page 633.

Section II. Resources and References

See section five for a formal list of citations of the research done for this project. The developer began his preparation for this project by reviewing his class notes on ID3 and chapter 16 of the book (specifically, section four) [1]. He used this information to write very broad-level, mostly-text filled pseudo code and information concerning the C code he will develop. To fill in the holes in his pseudo-code and complete his theoretical understanding of ID3, the developer then turned to several internet databases and academic websites and found three addition academic write-ups on ID3 ([2], [3], and [4]). He read through each of these until he was comfortable enough with ID3 to write each function and data structure in his pseudo-code specific enough so he will be able to easily implement it in C. Of the sources, he found Chapter 6, Section 2 of Steven Marsland's *Machine Learning: An Algorithmic Perspective* most useful for preparing for this project, because it did the best job of breaking down how ID3 works in each of the specific cases in the overall algorithm [2].

Section III. Goals

- See section VI for a listing of the pseudo code the engineer developed in preparation for this project. It does a relatively complete job of showing his thought process in developing the algorithm.
    - **Disclaimer:** This pseudo code, though detailed, is only a starting point for the development process. It is not meant to be an absolute solution to the problem. <u>If he finds that any portion of the pseudo-code is impractical or finds a more efficient way to do something in development, the engineer reserves the right to stray from it.</u>
- Chronologically, overall goals for the project include:
    - Fully develop ID3. This means developing each function in section VI alongside *ID3debug.c*. Develop and test modularly on a per function basis to minimize errors and bugs in the development process.
    - Develop ID3_print_tree once the overall ID3 algorithm has been built to show trees as formatted output.
    - Develop ID3_tree_validate to address the validation of the training set for each produced ID3 tree.
    - Address the issues in *Intellegent Systems: Principles, Paradigms, and Pragmatics [1]* chapter 16, section 4, page 633 related to conflicting data sets in *Takehome3_report.pdf.*

- Beyond this, overall strategies for the project include:
    - Reference to the documents in section five for clarifications on the overall theoretical algorithm.
    - Modular, function-by-function development and extensive use of *ID3debug.c* to test each function one at a time and minimize errors in development.
    - When bugs do occur, extensive use of GDB and Valgrind to resolve them.

Section IV. Outcomes and Deliverables

- C source code for the engineer's implementation of ID3. This will consist of:
    - *ID3.c*: File to contain all functions developed related to the ID3 Algorithm itself. It will be fully compatible with any software package that supports C software integration.
    - *ID3.h:* Header file containing all data structures utilized in ID3.c. This file and ID3.c will form an ID3 extension plugin for C.
    - *ID3test*.c: File to contain several test cases from ID3. The scripts in this file will run ID3 on each example utilizing ID3.c. The examples will come from *Intelligent Systems: Principles, Paradigms, and Pragmatics* and will specifically be from:
        - Pg. 617, Table 16.3. C for Three-Input OR (exhaustive set of instances).
        - Pg. 620, Table 16.5. Four Class, Two-Attribute Training Examples.
        - Pg. 623, Revised Training Data.
        - Pg. 635, Case 2: Reduced C.
    - *ID3debug*.c: File to contain any additional test cases necessary in development of the algorithm. It will contain one function call to each developed function to allow modular development of the algorithm.
    - *Makefile*: File for building ID3 files. Capable of producing and destructing the following executables:
        - *ID3:* Executable variant of ID3test.c.
        - *Debug:* Executable variant of ID3debug.c.
- *Output.txt:* A text file showing the output of *ID3test.c.*
- *Debug_Output.txt:* A text file showing the output of *ID3debug.c*.
- *Takehome3_report.pdf*: PDF report discussing Takehome 3 as per the original Takehome3 Project Specification PDF.

Section V. Reference List

[1] R. Schalkoff. Intellegent Systems: Principles, Paradigms, and Pragmatics. ISBN: 978-0-7637-8017-3. Cited: 11/12/2015.

[2] S. Marsland. Machine Learning: An Algorithmic Perspective [Online]. Available: http://www.briolat.org/assets/R/classif/Machine%20learning%20an%20algorithmic%20perspective(2009).pdf. Cited: 11/12/2015.

[3] R. Bhardwaj & S. Vatta. Implementation of the ID3 Algorithm [Online]. Available: http://www.ijarcsse.com/docs/papers/Volume_3/6_June2013/V3I6-0454.pdf. Cited: 11/12/2015.

[4] University of Florida Department of Computer & Information Science & Engineering. The ID3 Algorithm. Available: http://www.cise.ufl.edu/~ddd/cap6635/Fall-97/Short-papers/2.htm. Cited: 11/12/2015.

Section VI. Implementation Information and Pseudo-Code

Input file format:
Number of Inputs (N), Number of Possible Input Values, Possible Input Values
Number of Outputs (M), Number of Possible Output Values, Possible Output Values
Number of Rows of Data
Data by row (I1, I2, …, IN-1, IN, O1, O2, …, OM-1, OM)

Example File:

Title: or.txt
Contents:

```
3 2 0 1
1 2 0 1
8
0 0 0 0
0 0 1 1
0 1 0 1
0 1 1 1
1 0 0 1
1 0 1 1
1 1 0 1
1 1 1 1
```

ID3.h Data Structure Pseudo Code:

```
ID3_metadata_t
{
        int inputs // Number of inputs in data set
        int input_values // Number of possible input values
        int outputs // Number of outputs in data set
        int output_values // Number of possible output values in data set
        int rows // Number of rows of data in data set
}

ID3_tree_node_t
{
        int name //String for name
        char type;
        ID3_tree_node_t *children[] // Vector of children pointers num_inputs long.
        int **data_table // Pointer to data table used at this step in algorithm.
                // Row by Num_inputs + Num_Output dimensions.
}
```

Data_table specifics for both implementations:
1. From data by row section of file format.
2. NumRows by Num_Inputs + Num_Outputs in dimension.
3. Inputs and Outputs by column. Able to keep track of which is which with
   Num_Inputs and Num_Outputs.

ID3.c Psuedo-code:

Global variables: Num_Inputs, Num_Input_Values, Num_Outputs, Num_Output_Values, Num_Rows;

```
ID3_init(metadata)
{
        /* Initialize Global Variables for ID3 algorithm. */
        Num_Inputs = metadata->inputs;
        Num_Input_Values = metadata->input_values;
        Num_Outputs = metadata->outputs;
        Num_Output_Values = metadata->output_values;
        Num_Rows = metadata->rows;
}

ID3_create_node(data_table)
{
        new_node = malloc(sizeof(ID3_tree_node_t));
        new_node->name = -1;
        new_node->type = ‘\0’;
        new_node->data_table = data_table;
        new_node->children = malloc(num_inputs*sizeof(ID3_tree_node_t *));
        for(I = 0; I < Num_Inputs; ++I) new_node->children[i] = NULL;
        return new_node;
}

ID3_modify_node_name(node, new_name)
{
        node->name = name;
}

ID3_link_nodes(parent, child, index);
{
        parent->children[index] = child;
}

ID3_negative_ones(data_table)
{
        Returns TRUE if all inputs in "data table" are -1.

        Else, returns FALSE.
}

ID3_check_outputs(data_table)
{
        Returns FALSE if outputs in “data table” are not the same.
}
```

```
ID3_most_common_output(data_table)
{
        Returns most common output in "data table" for all -1's case.
}

ID3_calculate_entropy(data_table, num_inputs)
{
        Calculate entropies for each input: Ignore -1 columns.

        Return as a 1xN row vector with entropies for each input in their
        corresponding column.
}

ID3_find_input(entropies)
{
        Find min of entropies in “entropies” row vector.

        Return column index of minimum value to represent input to partition by.
}

ID3_input_values(data_table, input);
{
        Returns the number of possible input values for input x in "data table".

        Utilize max function to do this. Since inputs are encoded 0 to N - 1, the
        max(inputs) + 1 will be the number of inputs.
}

ID3_partition_table(data_table, input, input_val)
{
        Returns an array of all rows of data_table with column
        “input” = “input_val”.

        All entries in column “input” are changed to -1.
}
```

```
ID3_compute(node, data_table)
{
    if(Error = 1) return; // Return if invalid data set

    if(ID3_negative_ones(data_table))
    {
        /* Base case 1 (Rare): Output is most common value. */
        most_common = ID3_most_common_output(data_table);
        ID3_modify_node_name(node, most_common);
        node->type = ‘O’;
    }
    else if(check_outputs(data_table))
    {
        /* Base case 2 (Usual): Output is whatever triggered the if statement. */
        output = data_table[1][Num_Inputs + Num_Outputs];
        ID3_modify_node_name(node, output);
        node->type = ‘O’;
    }
    else
    {
         /* Calculate entropy vector: */
        entropies = ID3_calculate_entropy(data_table);

        /* Choose an input column index based on entropy: */
        input = ID3_find_input(entropies);
        num_input_vals = ID3_input_values(data_table, input);

        /* Update node name to the input being considered: */
        ID3_modify_node_name(node, input);
        node->type = ‘I’;

        for(i = 0; i < num_input_vals; ++i)
        {
            /* Create data table partitioned by input for this value: */
            table = ID3_partition_table(data_table, input, i);

            /* Create new decision node (Initially with a blank name: */
            new_node = ID3_create_node(table);

            /* Add decision node to tree: */
            ID3_link_nodes(node, new_node, i);

            /* Recursively call the algorithm on the new node: */
            ID3_compute(new_node, table);
        }
    }
}
```

```
ID3_encode_data(char *fname, ID3_metadata_t *metadata)
{
    Read input data in from file and:
        1. Store metadata in second argument.
        2. Use metadata to build an encoded data table of exclusively positive integers.

    For example, if Num_Inputs = 3 and possible values are A, B, C; A = 0, B = 1, and C = 2.

    Further, if Num_Outputs = 1 and possible values are A, B; A = 0 and B = 1.

    Return encoded data array. Will be of dimensions Num_Rows by Num_Inputs + Num_Outputs.
}


ID3(data_file_ptr)
{
        /* Encode data table. */
        ID3_metadata_t metadata;
        data= ID3_encode_data(data_file_ptr, &metadata);

        /* Create initial root node. */
        root = create_node(data);

        /* Initialize ID3 Algorithm with metadata (for speedup) */
        ID3_init(metadata);

        /* Run ID3 on root node. */
        ID3_compute(root, data);

        Return root;
}

ID3_print_tree(root_node)
{
        Prints tree with root node “root node” to stdout.

        Write this when it is time to test overall ID3 algorithm.
}

ID3_tree_validate(root_node, training_set)
{
        Validates ID3 tree with root node “root node” and training set “training set”.

        Encode table.

        Call auxiliary function to validate tree

        Write after ID3 is working.
}

ID3_tree_destruct(root_node, data_table)
{
        Recursively goes from root node to leaf nodes and frees each node in a
        bottom up style. Use ID3_free_node auxiliary function.

        Write this last in ID3.c.
}

ID3_free_node(node)
{
        Frees name and data_table members of "node".

        Write this with tree_destruct().
}
```