Paper 094-2010

Building Decision Trees from Decision Stumps

Murphy Choy, University College Dublin
Peter Flom, Peter Flom Consulting

ABSTRACTS

Decision stumps are simple one layer decision trees using all the same settings as the typical decision trees. However, most decision stumps are not useful in prediction on their own. To produce more predictive models, a decision tree is better. Building a decision tree in SAS® is much more difficult than a decision stump. By utilizing an innovative approach on macro usage, a full decision tree can be built using a variety of decision stumps. In this paper, we will demonstrate the simplicity of use of the decision stumps.

INTRODUCTION

Decision trees are commonly used in data mining. They are often used for segmentation of population or predictive models. It is also a white box model which expresses the rules in simple Boolean logic. Due to the ease of interpretation, they are extremely popular in helping users to understand various aspects of their data. Decision tree also has an added advantage of handling of missing values. While regression models cannot use observations with missing value, decision trees treat them as an extra value and model around it. Although decision trees are common and widely understood, very few people have heard of decision stumps.

WHAT ARE DECISION STUMPS?

Decision stumps are basically decision trees with a single layer. As opposed to a tree which has multiple layers, a stump basically stops after the first split. Decision stumps are usually used in population segmentation for large data. Occasionally, they are also used to help make simple yes/no decision model for smaller data with little data.

Decision stumps are generally easier to build as compared to decision tree. At the same time, SAS® coding for decision stumps are more manageable compared to CART and CHAID. The reason being that the decision stumps is just one single run of the tree algorithm and thus does not need to prepare data for the subsequent splits. At the same time, there is no need to specify the data for the subsequent split which make renaming of output simpler to manage.

CART OR CHAID SETTINGS?

Different modelers have different preferences. Some believe that one type of decision tree is superior to the others. It is in my opinion that such comparisons are subjective to the data that they are tested on. In this paper, the author has decided to use the CART specifications for the decision stumps for the following reasons.

1.) Easier to understand splits
    a. Binary splits are easier to understand
    b. Can be phrased as an either or statement
2.) Able to handle different data types
    a. CART is able to handle nominal, categorical and missing values simultaneously unlike CHAID.
3.) More robust statistics
    a. CHAID uses chi square test which is size dependent and suffers from multiple comparison test deficiency.
    b. Benferroni adjustment does not fully compensate for the deficiency.
4.) Less dispersion effects
    a. Multiple splits in a single node results in smaller subsequent nodes that may cause severe skewness in validation.

By using the CART specification, we will be using the Gini impurity index to determine the splits.

SPLITTING CRITERION

Gini impurity is a measure of how frequently a randomly chosen element from a set is incorrectly labeled if it were labeled randomly according to the distribution of labels in the subset. Gini impurity is computed by summing the product of the probability of each item selected and the probability of wrong classification for that item. It reaches its minimum (zero) when all cases in the node fall into a single target category. It is a very powerful measure as it allows for very rapid differentiations of good and bad in a node. The Gini impurity index is the default selected criterion by Leo Breiman in his paper on CART (1994).

To compute Gini impurity for a set of items, suppose y takes on values in {1, 2, ..., m}, and let f_i = the fraction of items labelled with value i in the set.

$$I_G(i) = \sum_{i=1}^{m} f_i(1 - f_i) = \sum_{i=1}^{m} f_i - f_i^2 = \sum_{i=1}^{m} f_i - \sum_{i=1}^{m} f_i^2 = 1 - \sum_{j=1}^{m} f_i^2$$

BUILDING THE SAS® DECISION TREE STUMP MACRO

Combining all the algorithms above, we can then put together the fundamentals of a decision tree stump. One of the most important move in the decision tree stump is the pre-summarization of information based on each level of value for each variable. This single move helps to reduce the overall processing time needed to calculate the best split. Bearing in mind that SAS®/IML is not used in this situation, the pre-summarization move is one out of necessity and efficiency. After the pre-summarization, we then calculate the gini for each split and then selecting the split which produces the best results. Even though pre-summarization helps to accelerate the calculation process, if there are too many single unique values in the data, there is a possibility that the macro will take a long time to finish calculations. To overcome this problem, it is suggested that some kind of rounding off maybe desirable in cases of continuous variables.

BUILDING THE LINKAGE FOR A TREE

By repeating the decision stumps on each output data sets, we will be able to build a CART. At the same time, as the tree progressively gets built, the data sets becomes smaller making calculation faster.

CONCLUSION

Decision stumps can be useful for a variety of purpose and by linking them together; we can build a full decision tree for modeling purposes. All this can be done in base SAS® and without external engines.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Murphy Choy
E-mail: goladin@gmail.com

Appendix A: Decision Stump Macro

```
/*************************************************************************
DECISION TREE SPLIT SECTION
*************************************************************************/

%MACRO DTREE(
/*****************************************************************/
INPUT = ,              /*INPUT DATA SET                              */
KEEP_VAR = ,           /*VARIABLE TO USED FOR THIS PARTICULAR SPLIT */
DROP_VAR = ,           /*VARIABLES NOT TO BE USED FOR THIS SPLIT    */
YVAR =  ,              /*DEPENDENT VARIABLE                          */
LEFT = ,               /*LEFT SPLIT                                  */
RIGHT = ,              /*RIGHT SPLIT                                 */
BASE =                 /*BASE MINIMUM POPULATION                     */
/*****************************************************************/
);

/*************************************************************************/

%IF %SYSFUNC(EXIST(&INPUT)) %THEN %DO;

        /*VARIABLES EXTRACTION STEP*/

        %IF &KEEP_VAR ^= %THEN %DO;
                PROC CONTENTS DATA = &INPUT(KEEP = &KEEP_VAR) OUT=CONTENTS;
        %END;
        %ELSE %IF &KEEP_VAR ^= %THEN %DO;
                PROC CONTENTS DATA = &INPUT(DROP = &DROP_VAR &YVAR) OUT=CONTENTS;
        %END;
        %ELSE %DO;
                PROC CONTENTS DATA = &INPUT(DROP = &YVAR) OUT=CONTENTS;
        %END;
        RUN;

        /*************************************************************************/

        /*ASSIGNMENT OF VARIABLE NAMES AND THE OPTIONS FOR THE SUBSEQUENT RUNNING*/

        DATA _NULL_;
        SET CONTENTS;
        CALL SYMPUT('VAR'||TRIM(LEFT(_N_)),NAME);
        CALL SYMPUT('VARCOUNT',_N_);

        RUN;

        /*************************************************************************/

        /*SUMMARIZATION FOR EACH VARIABLE FOR EACH GROUP*/

        PROC SQL NOPRINT;

        %DO I = 1 %TO &VARCOUNT;

                CREATE TABLE &&VAR&I AS SELECT
                &&VAR&I AS &&VAR&I,MIN(&&VAR&I) AS MIN,MAX(&&VAR&I) AS MAX,
                SUM(&YVAR) AS BAD, COUNT(&YVAR) AS TOTAL FROM &INPUT GROUP BY &&VAR&I;

        %END;

        /*ASSIGNMENT OF THE OVERALL VALUES*/

        SELECT SUM(&YVAR),COUNT(&YVAR) INTO :TOTAL_BAD,:TOTAL_POP FROM &INPUT;

        QUIT;

        /*************************************************************************/

        /*CREATION OF NAMES IN EACH DATASET*/

        %DO I = 1 %TO &VARCOUNT;
```

```
        DATA &&VAR&I;
        SET &&VAR&I;

                LENGTH VARNAME $ 155;

                VARNAME = "&&VAR&I";

        RUN;

%END;

/*************************************************************************/

/*GINI CALCULATIONS*/

DATA OVERALL(
/*KEEPING THE CRUCIAL VARIABLES*/
KEEP=VARNAME MAX
CUM_BAD CUM_POP CUM_GOOD
ALT_BAD ALT_POP ALT_GOOD
GINI_POP GINI_ALT GINI
);

/*SETTING EACH DATA SET LEVEL COUNTS*/
SET
%DO I = 1 %TO &VARCOUNT;
        &&VAR&I
%END;
;
/*RETAINING VARIABLES*/
RETAIN CUM_BAD CUM_POP;

/*BEGINNING OR NEW VARIABLE REINITIALIZATION*/
IF _N_ = 1 OR VARNAME ^= LAG(VARNAME) THEN DO;

        /*LEFT HAND SIDE*/
        CUM_BAD = BAD;
        CUM_POP = TOTAL;
        CUM_GOOD = CUM_POP - CUM_BAD;

        /*RIGHT HAND SIDE*/
        ALT_BAD = &TOTAL_BAD - CUM_BAD;
        ALT_POP = &TOTAL_POP - CUM_POP;
        ALT_GOOD = ALT_POP - ALT_BAD;

        /*GINI CALCULATIONS*/
        GINI_POP = 1 - (CUM_BAD/CUM_POP)**2 - (CUM_GOOD/CUM_POP)**2;
        GINI_ALT = 1 - (ALT_BAD/ALT_POP)**2 - (ALT_GOOD/ALT_POP)**2;

        /*MINIMUM POPULATION FILTER*/
        IF CUM_POP < &BASE OR ALT_POP < &BASE THEN DELETE;

        /*OVERALL GINI*/
        GINI = (CUM_POP/&TOTAL_POP)*GINI_POP+(ALT_POP/&TOTAL_POP)*GINI_ALT;


END;
ELSE DO;
        /*LEFT HAND SIDE*/
        CUM_BAD = CUM_BAD + BAD;
        CUM_POP = CUM_POP + TOTAL;
        CUM_GOOD = CUM_POP - CUM_BAD;

        /*RIGHT HAND SIDE*/
        ALT_BAD = &TOTAL_BAD - CUM_BAD;
        ALT_POP = &TOTAL_POP - CUM_POP;
        ALT_GOOD = ALT_POP - ALT_BAD;

        /*GINI CALCULATIONS*/
        GINI_POP = 1 - (CUM_BAD/CUM_POP)**2 - (CUM_GOOD/CUM_POP)**2;
```

```
                IF ALT_POP > 0 THEN GINI_ALT = 1 - (ALT_BAD/ALT_POP)**2 -
(ALT_GOOD/ALT_POP)**2;
                ELSE DELETE;

                /*MINIMUM POPULATION FILTER*/
                IF CUM_POP < &BASE OR ALT_POP < &BASE THEN DELETE;

                /*OVERALL GINI*/
                GINI = (CUM_POP/&TOTAL_POP)*GINI_POP+(ALT_POP/&TOTAL_POP)*GINI_ALT;



        END;
        IF GINI = 0 THEN DELETE;
        RUN;

        /*************************************************************************/

        PROC SQL;

        /*CHECKING FOR AT LEAST 1 RULE THAT CAN BE USED*/
        SELECT COUNT(*) INTO :FLAG  FROM OVERALL;

        %IF &FLAG >= 1 %THEN %DO;

                /*SELECTING THE RULE*/
                SELECT VARNAME,MAX INTO :SELECT,:BREAK FROM OVERALL HAVING GINI =
MIN(GINI);

                /*CREATING A NODE RULE DATA SET*/
                CREATE TABLE &INPUT._RULE AS SELECT * FROM OVERALL HAVING GINI =
MIN(GINI);

                /*LEFT HAND SPLIT*/
                CREATE TABLE &LEFT AS SELECT * FROM &INPUT WHERE &SELECT <= &BREAK;

                /*RIGHT HAND SPLIT*/
                CREATE TABLE &RIGHT AS SELECT * FROM &INPUT WHERE &SELECT > &BREAK;

        %END;

        QUIT;

/*************************************************************************/

%END;

/*************************************************************************/
%MEND;
```

Appendix B: Decision Tree Linkage

```
/*************************************************************************
The decision tree linkage.
*************************************************************************/

%LET DTREE_DATA = GERMAN_CLASS; /*Input data*/
%LET DEP_VAR = Y; /*Dependent Variable*/
%LET BASE = 50; /*Minimum Split Size*/

/*************************************************************************

*************************************************************************/
```

```
%MACRO DTREE_USER();

DATA NODE1;
SET &DTREE_DATA;
RUN;

/*************************************************************************/

/*LEVEL 1 SPLITS*/
%DTREE(INPUT = NODE1, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE2, RIGHT =
NODE3,BASE = &BASE);

/*LEVEL 2 SPLITS*/
%DTREE(INPUT = NODE2, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE4, RIGHT =
NODE5,BASE = &BASE);
%DTREE(INPUT = NODE3, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE6, RIGHT =
NODE7,BASE = &BASE);

/*LEVEL 3 SPLITS*/
%DTREE(INPUT = NODE4, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE8, RIGHT =
NODE9,BASE = &BASE);
%DTREE(INPUT = NODE5, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE10, RIGHT =
NODE11,BASE = &BASE);
%DTREE(INPUT = NODE6, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE12, RIGHT =
NODE13,BASE = &BASE);
%DTREE(INPUT = NODE7, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE14, RIGHT =
NODE15,BASE = &BASE);

/*LEVEL 4 SPLITS*/
%DTREE(INPUT = NODE8, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE16, RIGHT =
NODE17,BASE = &BASE);
%DTREE(INPUT = NODE9, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE18, RIGHT =
NODE19,BASE = &BASE);
%DTREE(INPUT = NODE10, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE20, RIGHT =
NODE21,BASE = &BASE);
%DTREE(INPUT = NODE11, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE22, RIGHT =
NODE23,BASE = &BASE);

%DTREE(INPUT = NODE12, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE24, RIGHT =
NODE25,BASE = &BASE);
%DTREE(INPUT = NODE13, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE26, RIGHT =
NODE27,BASE = &BASE);
%DTREE(INPUT = NODE14, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE28, RIGHT =
NODE29,BASE = &BASE);
%DTREE(INPUT = NODE15, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE30, RIGHT =
NODE31,BASE = &BASE);

/*LEVEL 5 SPLITS*/
%DTREE(INPUT = NODE16, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE32, RIGHT =
NODE33,BASE = &BASE);
%DTREE(INPUT = NODE17, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE34, RIGHT =
NODE35,BASE = &BASE);
%DTREE(INPUT = NODE18, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE36, RIGHT =
NODE37,BASE = &BASE);
%DTREE(INPUT = NODE19, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE38, RIGHT =
NODE39,BASE = &BASE);

%DTREE(INPUT = NODE20, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE40, RIGHT =
NODE41,BASE = &BASE);
%DTREE(INPUT = NODE21, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE42, RIGHT =
NODE43,BASE = &BASE);
%DTREE(INPUT = NODE22, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE44, RIGHT =
NODE45,BASE = &BASE);
%DTREE(INPUT = NODE23, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE46, RIGHT =
NODE47,BASE = &BASE);

%DTREE(INPUT = NODE24, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE48, RIGHT =
NODE49,BASE = &BASE);
%DTREE(INPUT = NODE25, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE50, RIGHT =
NODE51,BASE = &BASE);
```

```
%DTREE(INPUT = NODE26, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE52, RIGHT =
NODE53,BASE = &BASE);
%DTREE(INPUT = NODE27, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE54, RIGHT =
NODE55,BASE = &BASE);

%DTREE(INPUT = NODE28, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE56, RIGHT =
NODE57,BASE = &BASE);
%DTREE(INPUT = NODE29, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE58, RIGHT =
NODE59,BASE = &BASE);
%DTREE(INPUT = NODE30, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE60, RIGHT =
NODE61,BASE = &BASE);
%DTREE(INPUT = NODE31, KEEP_VAR = , DROP_VAR = , YVAR =  &DEP_VAR,LEFT = NODE62, RIGHT =
NODE63,BASE = &BASE);

/*****************************************************************************/
%MEND;

/*****************************************************************************

*****************************************************************************/
%DTREE_USER();
```