help chaid

---------------------------------------------------------------------------------

Title

Chi-square automated interaction detection (CHAID)

Syntax

chaid depvar [if] [in] [weight] [, minnode(integer) minsplit(integer)

unordered(varlist) ordered(varlist) noisily missing mergalpha(pvalue)

respalpha(pvalue) spltalpha(pvalue) maxbranch(integer) dvordered noadj

nodisp predicted importance xtile(varlist, xtile\_opt) permute svy

exhaust]

fweights are allowed; see weight

Description

chaid is a recursive partitioning algorithm that searches for an optimal

decision tree structure based on the correspondence between the

dependent/response variable and a set of independent/splitting variables.

chaid is part prediction, part clustering estimation command that seeks to

reduce uncertainty about the values of/predict a response variable but

simultaneously partitions the dataset into clusters of observations based

on the set of splitting variables.

The chaid algorithm cycles through 2 (or, optionally, 3) processes

recursively. First, chaid seeks to reduce overfitting the data by

optimally merging together categories/levels of a splitting variable i.

Without the optimal merging step, the splits chaid uncovers tend to be

biased toward predictors with more categories. In the merging step, chaid

chooses a splitting variable i of the p total splitting variables and

deciphers the total number of possible combinations k of 2 levels j of

splitting variable i that are "mergable" according to the splitting

variable type (for unordered splitting variables, all combinations of two

levles are possible; for ordered splitting variables, only adjacent level

value combinations are possible). chaid then takes combination j and

conducts a statistical association test (i.e., chi-square) using

combination j (i.e., omitting all other combinations, using combination j

as a dummy code/indicator) on the response variable to determine the

probability (i.e., p-value) of the association between the response

variable and the categories separated by combination j being 0. The process

of obtaining p-values from a chi-square test continues for all k

combinations of splitting variable i. After all k combinations have

obtained a p-value, chaid finds the combination j that has the highest

p-value/weakest association with the response variable. If the magnitude

of the p-value is higher than a user-defined threshold, combination j are

merged into a single category. To add a little more detail, high p-values

in the merging step indicate a high probability that the categories

separated by combination j are independent of the response variable, would

not likely produce a meaningful split, and can effectively be merged

together. After merging the categories with the highest p-value, chaid

then begins again, deciphering, again for splitting variable i, the number

of valid combinations k with the formerly separate categories now treated

as a single category. The merging process continues for splitting variable

i until the combination j with the highest p-value does not pass the user

defined threshold for merging or there are 2 levels remaining in the

optimally merged splitting variable. When the p-value for combination j

does not pass the thnreshold for merging, chaid remembers the optimal

merging of levels associated with splitting variable i and moves to the

next splitting variable. The optimal merging stage continues until all p

splitting variables have an optimally merged structure. chaid then moves

to step 2.

Following the merging step, chaid moves to the second, splitting step.

chaid uses the optimally merged structure to decide which variable, if any,

should be used to split the data into clusters/nodes. For all p optimally

merged splitting variables, chaid conducts a chi-square association test of

that splitting variable i on the response variable and records the p-value.

Owing to the multiple comparisons, the p-value in the splitting step is

Bonferroni adjusted according to splitting variable type. After optaining

all p adjusted p-values, chaid finds the smallest and, if the p-value

threshold and other splitting criteria are met, the data are split into

clusters/nodes based on the splitting variable with the smallest adjusted

p-value. Besides the p-value threshold, chaid imposes several other

criteria on splits. In particular, the clusters/nodes must be of a certian

size. For example, if the minimum cluster/node size is 100, and chaid

finds a split in the data that would produce 3 clusters of size 175, 120,

and 80 - the split will be prevented as one of the possible splits is

smaller than the minimum. When chaid finds a split that will be smaller

than the minimum size, all the observations in the data associated with

that potential split are removed from consideration of any remaining splits

and the algorithm moves to other observations that are still "splitable".

If chaid produces a split in the data, or decides to move along to other

splitable observations, it begins again at Step 1 and forgets the optimal

merging it produced in the previous step. Moreover, chaid now searches

within each cluster for another optimal merger set and potential split.

Thus, when a split occurs, all subsequent merging and splitting are

contingent on previous splits and the dataset is "partitioned" conditional

on the previous splits.

The chaid algorithm stops when a] the minimum cluster/node size is not met

for potential splits across all clusters, b] the minimum cluster/node size

is not met for splitting across all cluters (i.e., when chaid moves from

Step 2 back to Step 1, the size of the cluster is below the minimum size

allowed to attempt a split, c] the cluster/node is pure (i.e., composed of

single value of response variable), d] the maximum number of

branches/contingencies is met, and e] a cluster has a single value for each

splitting variable (i.e., no splits possible). When the 5 criteria are met

across all observations/clusters/nodes in the data, chaid stops and reports

results.

Traditionally, a third step occurring during Step 1 above involving

attempting to "re-split" a merged combination of the levels of a splitting

variable is attempted. Specifically, after a merger between combination j

occurs, if combination j contains 3 or more levels of splitting variable i,

chaid attempts a "Step 2-like" split in which chaid searches through all

potential binary splits (i.e., splits into 2 categories) of the 3 or more

optimally merged categories. If the used-defined threshold is met, the

split is carried out and the newly split set of levels is substituted into

the merging levels set in Step 1. The "re-split" step is implemented in

chaid as an option (when respalpha() receives a positive probability

value). The re-splitting step is necessary to reach near optimal results,

but can greatly increase run time and as Kass (1980) notes "In practice a

merger is rarely split,..." (p. 121). Hence, the user can choose to invoke

or ignore this step.

The current implementation of chaid differs from the traditional use of

contingency tables in that it uses logistic models to estimate chi-square

values and, as such, may require somewhat larger sample sizes than do other

implementations of chaid for the ml algorithm to converge. The default

estimation method for chaid is mlogit. The use of Stata's ml based

commands greatly increases the flexibility of the kinds of data chaid can

accomodate. chaid also uses levelsof to separate out levels of splitting

and response variables. Consequently, all splitting variables and the

response variable must non-negative integers and, for the current

implementation, must have fewer than 21 levels or unique values.

chaid generates, by default, a variable named \_CHAID after finishing

execution that indicates each observation's membership in a cluster as

defined by chaid. If you have a variable named \_CHAID already in your

dataset, chaid will overwrite it with the new value of \_CHAID based on the

current cluster membership.

Display

The decision tree structure is returned in two forms 1) as a matrix that is

read from top-down, and 2) as a graph which shows the hierarchical

structure of the chaid tree including the split contingencies. For example

consider the results from example #1 below (i.e., the below matrix is

exactly what chaid reports):

Chi-Square Automated Interaction Detection (CHAID) Tree Branching Results

--------------------------------------------------------------------------------

1 2 3 4

+---------------------------------------------------------+

1 | xtlength@1 xtlength@2 xtlength@2 xtlength@3 |

2 | rep78@1 3 2 rep78@4 5 |

3 | Cluster #1 Cluster #2 Cluster #4 Cluster #3 |

+---------------------------------------------------------+

The results show that 4 clusters were uncovered from the data (the clusters

are out of order due to the way in which the algorithm searches for splits)

- each column represents a distinct cluster uncovered by chaid. The first

split is represented in the first row. Each column in the first row has a

variable name, an ampersat sign, and some valid values of the variable.

The variable name represents the variable on which the first split occurred

for the chaid algorithm - in this case the xtiled length variable from the

auto dataset. The values following the ampersat signs refer to the values

chosen through the optimal merging steps described above of the variable.

In this case, all valid values were deemed by chaid to be different from

one another and were not merged. Thus, split #1 was on the lentile

variable into levels 1, 2, and 3. The dataset was then partitioned based

on that split and the remaining splits are then conditional on the first

split. All successful chaid runs will have a populated first row.

Notice that Clusters #2 and #4 have repeated values of xtlength@2. This is

because another split occurred within xtlength@2, but not for xtlength@1 or

xtlength@3 - which is why each only has an entry at the first row. The

split on xtlength@2 is represented in the second row for the rep78

variable, splitting into optimally merged 1 3 2 and 4 5 groups. Thus,

given a observation was in xtlength category 2, one additional split was

possible putting observations into rep78 1 3 and 2 versus 4 and 5. The

rows "lower" on the matrix are then response on the rows "higher" in the

matrix. Furthermore, the number of rows represents the number of

"branches" the decision tree has. Finally, each column represents each

cluster's unique "path" leading from the first split in row 1 to the final

populated row of the column/cluster in question.

The graph that is returned by chaid can be cluttered with value labels. If

unreadable, consider using Stata's graph editor to alter the size, angle,

and location of the labels. Currently, graph options are not available to

alter the appearence of the graph using Stata syntax directly.

Options

minnode() specifies the minimum number of observations allowed in a

terminal cluster or "node." If an optimally merged splitting varible

passes to the point of splitting, but one or more of the clusters that

would be created by the optimally merged splitting variable is below

the minnode() value, the split will not be carried out. Thus, minnode()

prevents chaid from carrying out Step 2. The default value for

minnode() is 100.

minsplit() specifies the minimum number of observations across all levels

of an optimally merged splitting variable to allow any split to occur,

irrespective of how many observations would be in each of the final

clusters. Hence, previous to clustering, if a the number of

observations does not meet the minsplit() minimum, no clustering will

occur. minsplit() prevents chaid from executing Step 1. The default

value for minsplit() is 200.

unordered() treats the variable list included as "unordered." The unordered

treatment affects both how categories of the splitting variables are

combined (i.e., any categories can be combined), and how final

significance of the optimally merged splitting variable is computed

(i.e., unordered Bonferroni adjustment). Unordered predictors are the

most time intensive splitting variables for which to discern an optimal

merging, as all categories are potentially allowed to be merged.

ordered() treats the variable list included as "ordered." The ordered

treatment affects both how categories of the splitting variables are

combined (i.e., only adjacent categories can be combined), and how

final significance of the optimally merged splitting variable is

computed (i.e., ordered Bonferroni adjustment). Ordered variables with

missing invoked, when missing data are present, treat the missing

category as a "floating" option and, consequently, adjusts the p-value

for splitting based on the floating category as opposed to the usual

ordered adjustment.

noisily turns on the tracing of the command through split-related data

management and estimation runs. noisly can reassure the user that

chaid is running or, can show the user the entire process of

determining the splits arrived at by chaid.

missing allows missing data in the response and splitting variables to be

treated as another category. The missing option does not affect

unordered splitting variables, but does result in all ordered indepdent

variables with missing data, to have a "floating" missing data

category. Note that with the dvordered option that missing data on the

response variable is not treated as another category but, rather, is

treated as missing and marked out of the sample. Missing data with

ordered response variables must be imputed through some other means to

be included in chaid. The missing option can also result in missing

(i.e., ".") as a predicted value when combined with predicted.

mergalpha() sets the alpha level at which an allowable pair of categories

in an splitting variable can be merged in the merging step.

mergalpha() refers to the probability values the chi-square would take

on that would allow a merge. The default value is .95. Thus, p-values

that are among the middle 95% of the probability density for the

chi-squqare distribution are merged. In other words, predictors that

pass the p > .05 threshold are allowed to be merged.

respalpha() sets the alpha level at which an already optimally merged set

of 3 or more of an independnet variable's original categories will be

allowed to de-couple into a binary split. How the splitting variables

are split depends, of course, on the splitting variable's type and

whether the missing option is invoked. respalpha() refers to the

probability values the chi-square would take on that would allow a

split. The default setting for this option is to a negative value,

which shuts this option off. If the user chooses to invoke the

resplapha() option, the value for this option must be smaller than the

value of 1 - mergalpha().

spltalpha() sets the adjusted alpha level at which an optimally merged

predictor will be split following the merging step. As compared to

mergalpha(), spltalpha() works in the same way, save that small

p-values will produce a split - just as small p-values are deemed to be

statistically significant. The default value is .05. Thus, predictors

that pass the p < .05 threshold are allowed to be split.

maxbranch() sets the maximum number of "branches" on which the chaid tree

is allowed to take. In other words, maxbranch() sets a maximum in the

number of splits on which a particular terminal cluster can be based.

The default value is for there to be no maximum in terms of the number

of splits. Any negative value will turn off maxbranch().

dvordered changes the estimation procedure underlying chaid to be ologit

instead of the default mlogit. Hence, dvordered treats the response

variable as being ordered.

noadj prevents chaid from Bonferroni adjusting the spltalpha() p-values to

prevent potential Type I/"false positive" inferential error. noadj is

useful in cases where the user desires to be more lenient in terms of

uncovering relationships in the data. noadj is turned on by default

when using permute as the Bonferroni adjustment is intended to

approximate exact/permutation p-values (see Kass, 1980).

nodisp prevents chaid from displaying the decision tree structure and graph

produced by the algorithm.

predicted produces a predicted value for the response variable for each

cluster. The predicted value is the mode of the response variable for

that cluster. When predicted is invoked, chaid produces a variable

called CHAID\_predict containing the predicted value by default. Any

variable named CHAID\_predict in the dataset, including those from

previous runs of chaid, will be overwritten.

importance produces a permutation importance matrix for rank ordering the

splitting variables. The permutation importance approach derives from

the idea that if one randomly permutes the values of any one splitting

variable there should be a decrease in fit if that splitting variable

predicts the response variable. Larger decreases in fit based on

permutations of a splitting variable indicate that a splitting variable

is better at predicting the response variable than smaller decreases in

fit. All fit scores are computed based on Cramér's V (see tabulate

twoway), which obtains perfect fit when all the clusters have a single

value (i.e., are pure) or a similar metric when combined with svy. The

decreases in fit owing to permutation are normed out of 1 and displayed

in the "raw" row of e(importance). Because it is possible to obtain

negative decrements to fit (that is, increases in fit) due to permuting

the values of a splitting variable, and due to the random nature of the

permutation, the values in the "raw" row are not particularly

meaningful and only the the second row, "rank", of e(importance) should

be interpreted. Note that importance creates a Mata matrix called

"ifstmnts" that it uses to conduct the permutation importance

computations that the user can access.

xtile() is a convenience command to take variables that are otherwise

continuous, or ordered with many categories (i.e., counts), and

generate a set of ordered categorical quantiles from those variables.

The variables created by xtile() are automatically considered as

ordered() and are added to the dataset with the name "xtvarname". Any

variables called "xtvarname" already in the dataset will be overwritten

when using xtile(). The xtile() option allows the user specify the

number of quantiles created as an option (i.e., nquantiles(#))

following a comma. Not specifying the number of quantiles will result

in 2 quantiles (i.e., the default for xtile).

permute changes the way in which p-values are computed for splitting and

merging steps from the traditional large-sample approximation to a

monte-carlo permutation test-based approach (see permute for details).

Permutation tests are more approproate for smaller sample sizes (i.e.,

conditions under which large sample approximations will not hold) and

tend to be more conservative than the large sample approximation and,

thus, tend to produce fewer splits than the same model not using

permute. permute, owing to its monte carlo-based approach, greatly

increases chaid's run time. Permuation tests also do not require

Bonferroni adjustment and thus imply noadj. permute does not work with

svy or weights.

svy incorporates svyset complex survey design characterisitics into the

p-value computation. As with other svy commands, the data must be

svyset previous to running chaid. Option svy and weights cannot be

used together. Additionally, option svy and permute cannot be used

together.

exhaust implements the exhaustive CHAID algorithm described by Biggs, de

Ville, and Suen (1991). The only difference between exhaustive and

traditional CHAID is that exhaustive CHAID continues the merging step

until only a 2 categories/a binary split remains. Thus, exhaustive

CHAID ignores mergalpha() and will continue to merge categories until 2

optimally merged categioies of the original set if categories of

variable i remain. Option exhaust also changes the way the Bonferroni

adjustment is computed as the way it merges categories differs from the

traditonal method.

Remarks

Because there is a component of randomness to the chaid results, the user

should set seed prior to using chaid. Additionally, chaid utilizes

Mata and saves a string matrix there that the user can access post

estimation. One Mata matrix, called "CHAIDsplit", contains information

used to create the e(split#) and e(path#) macros. In some instances,

most especially with complicated and large CHAID trees, the strings

contained in some of the e(path#) macros will be truncated. Users

knowlegable of Mata can obtain such information from the "CHAIDsplit"

Mata matrix. To be specific, "CHAIDsplit" includes, in the first row,

information regarding all the splits in the data. The first column is

a label, "splits", and each column thereafter corresponds to a split

made in the data. The labels are represented as "varname value" with

no space between them. Thus, the rep78 variable from the auto dataset

would have 5 labels: rep781, rep782, rep783, rep784, and rep785 -

corresponding to all 5 levels of the rep78 variable. Merged categories

are separated in the "splits" row by commas. Thus, any labels not

separated by a comma have been "optimally merged" by chaid. The

remaining rows correspond to the "path" represented by each cluster.

Thus "path1" is the set of contingencies (or "and" logical statements

that resulted in "Cluster #1", "path2" is the set of contingencies that

resulted in "Cluster #2", etc... using the labels as outlined above

(i.e., with "varname value" format).

As with other computationally intense programs (e.g., findit gllamm),

collapsing the data over identical observations and using a fweight is

a way to speed up estimation time for larger datasets. Given that

chaid requires categorical data, using the collapse command is a

particularly useful approach - but, again, will not work with svy or

permute.

Due to numerical issues related to storage of very small p-values, chaid

uses the Akaike information criterion (AIC; see estat) to decide on

splits when p-values are identical (i.e., when both are effectively 0

at ~10^-300).

Introductory examples

#1: Basic CHAID analysis with altered minsplit() and minnode()

set seed 1234567

webuse auto

chaid foreign, unordered(rep78) minnode(4) minsplit(10) xtile(length, n(3))

#2: Basic CHAID analysis as in #1 with permutation tests

chaid foreign, unordered(rep78) minnode(4) minsplit(10) xtile(length, n(3))

permute

#3: Larger-scale CHAID with ordered response variable and permutation importance

webuse nhanes2f, clear

chaid health, dvordered unordered(region race) ordered(houssiz sizplace

diabetes sex smsa heartatk) importance

#4: Larger-scale CHAID with ordered response variable; collapsed using fweight

preserve

generate byte fwgt = 1

collapse (sum) fwgt, by(health region race houssiz sizplace diabetes sex

smsa heartatk)

chaid health [fweight = fwgt], dvordered unordered(region race)

ordered(houssiz sizplace diabetes sex smsa heartatk)

restore

#5: Exhaustive CHAID with complex survey design

svyset psuid [pweight=finalwgt], strata(stratid)

chaid health, dvordered unordered(region race) ordered(houssiz sizplace

diabetes sex smsa heartatk) svy exhaust

Saved results

chaid saves the following results to e():

scalars

e(N) number of observations

e(N\_clusters) number of clusters created by chaid and returned in

\_CHAID variable

e(fit) purity of the clusters (extent to which each cluster has

only a single value of the response variable); based on

Cramér's V

macros

e(cmdline) command as typed

e(cmd) chaid

e(title) title in estimation output

e(path#) displays the levels of each split leading to cluster #.

Each split is separated by semicolons. The splitting

variable is first followed by an "at" sign (i.e., @)

followed by the levels of the splitting variable that

describes cluster # at that split

e(split#) displays the #th split by chaid. Splitting variable is

displayed first followed by the optimal mergers of

levels of the splitting variable. Each merged set of

splitting variable levels is separated by parentheses

e(depvar) name of dependent/response variable

matrices

e(importance) permutation importance matrix

e(sizes) sample size of each cluster

e(branches) number of branches from the root node for each cluster

functions

e(sample) marks estimation sample

References

Kass, G. V. (1980). An exploratory technique for investigating large

quantities of categorical data. Applied Statistics, 29, 2, 119-127.

Biggs, D., de Ville, B., and Suen, E. (1991). A method of choosing multiway

partitions for classification and decision trees. Journal of Applied

Statistics, 18, 49-62.

Author

Joseph N. Luchman

Behavioral Statistics Lead

Fors Marsh Group LLC

Arlington, VA

jluchman@forsmarshgroup.com

Acknowledgements

Thanks to Florian Zettelmeyer, Paul Bergmann, and Kimberly Wilson for bug

reporting. Thanks also to Jonathan Mendelson and Luke Viera for

comments on the functionality of and suggestions about features for

chaid.

Also see

[R] mlogit, [R] ologit, [R] levelsof, [R] tabulate twoway, [R] svy, [R]

permute, [R] xtile.