README regtreeseg

regtreeseg package

The goal of this package is to segment CNV data based on the log2r between the sample genome and the reference genome. A value above zero in log2r means that there is a amplification in copy number in the sample genome compared to the reference genome. On the other hand, if the log2r is below zero then there is a deletion in the copy number in the sample genome compared to the refernce genome.

Using a regression tree approach set up in the 'rpart' package, CNV data can be segmented with multiple approaches.

There are 8 functions in this package. 6 of the functions segment the data in someway, and the other functions- WholeGenome.Plot and cpopt - accompany those.

Dataframe input specification

The inputs of these functions require a dataframe that has column names specifically labeled as "Start.Pos", "log2r", and "Chr". The "Start.Pos" column is the bin index or genomic location which the log2ratio comes from. The "log2r" column is the log2ratio that between the sample genome ad the reference genome. The "Chr" column is the chromosome of interest. The chromosomes should be labeled exactly as "chr1", "chr2", "chr22", "chrX", "chrY".

How to use each function:

cpopt:

Inputs: dataframe(required), conserve (optional)

Output: the optimal cp value as a numeric value

The cpopt takes a dataframe that has columns Start.Pos and log2r and returns the optimal cp for the data to reduce the cross validation error. The conserve option allows the user to make the optimal cp choice more conservative. The lower the cp value, the regression tree will have more splits in the regression tree. The cp value is the minimum benefit that a split in the tree must improve the R^2 value by to be included in the tree. Optimizing the cp value is a game between underfitting and overfitting. The default for the function is CONSERVE = FALSE. When this is set the minimum cp value is around .001. If overfitting is a worry CONSERVE = TRUE will make so the minimum cp value is around .01.

WholeGenome.Plot:

The whole genome data plots data for the whole genome. The whole genome also has a smoothing option. This option is used to show the segmentations.

Segmenting using optimal cp:

seg.genome:

Inputs: dataframe(required), png_filename(required), upper.y.lim (optional), lower.y.lim (optional), cpvalue (optional), conserve (optional)

Outputs: .png file with the whole genome plot and segmentation, a list containing a dataframe with all of the segmentation data(segments), and a dataframe with the predictions from the regression tree(regtreepred), and a dataframe with the optimal cp used for each chromosome (cpdf)

The seg.genome function segments all the data from the dataframe genome. The dataframe must have columns named specifically "Start.Pos", "log2r", and "Chr" (more about the specifications can be found in the section 'Data input specifications'). The function splits the data by chromosome and the segments that subset of the data using the cpopt for that chromosome. Then all the chromosome segmentations are binded together. The first part of the segmentation is getting the predictions from the regression tree approach. These can be found in the 'regtreepred' dataframe.Then the second part is the predictions being translated into Start, End, Chr, ChrN, log2r, location, and width segments that can be found in the 'segments' dataframe. Since each chromosome is segmented individually with its optimal cp value, the cp values used for each of the chromosomes are reported.

The seg.genome function returns a png file at the png_filename that the user provides. The WholeGenome.Plot function is used to create the plot. The optional inputs of upper.y.lim and lower.y.lim control the range of the y values on the plot.

A cpvalue can be specified in the input. By specifying a cp value, this cp value will be used to segment each chromosome. If no cp value is specified the optimal cp value will be used. The conservative optimal cp value can be specified in the input.

seg.chr

Inputs: dataframe(required), chromid (required), cpvalue (optional), conserve (optional)

Outputs: A list containing a dataframe with all of the segmentation data (segments),and a dataframe with the predictions from the regression tree (regtreepred), a plot of the segmentation and chromosome data (chrplot), and a dataframe with the optimal cp used for each chromosome on the first round of iteration(cpdf)

The seg.chr function segments all the data from the dataframe. The dataframe must have columns named specifically "Start.Pos", "log2r", and "Chr" (more about the specifications can be found in the section 'Data input specifications'). The dataframe can include any number of chromosomes. All will be segmented but only information on the chromosome of interest will be returned. The "chromid" indicates the chromosome of interest and it should be entered in the same format as the Chr column in the dataframe. The The first part of the segmentation is getting the predictions from the regression tree approach. These can be found in the 'regtreepred' dataframe. The second part is the predictions being translated into Start, End, Chr, ChrN, log2r, location, and information for each of those segments. This can be found in the 'segments' dataframe.

A cpvalue can be specified in the input. By specifying a cp value, this cp value will be used to segment each chromosome. If no cp value is specified the optimal cp value will be used. The conservative optimal cp value can be specified in the input.

Segmenting using optimal cp and regression tree iteration:

iterseg.genome

Inputs: dataframe(required), png_filename(required), upper.y.lim (optional), lower.y.lim (optional), cpvalue (optional), conserve (optional)

Outputs: A list containing a dataframe with all of the segmentation data (segments), and a dataframe with the predictions from the regression tree (regtreepred), a plot of the chromosome data and final predictions (chrplot), a list 5 plots of each step in iteration (plots), and the cp values used for each chromosome on the first iteration (cpdf)

The iterseg.genome function segments all the data from the dataframe. The dataframe must have columns named specifically "Start.Pos", "log2r", and "Chr" (more about the specifications can be found in the section 'Data input specifications'). The function splits the data by chromsome and segments that subset of data using the cpopt for that chromosome. Then all the predictions are binded together for a full dataframe with the regression tree predictions for the whole genome. Then the residual error is calculated between the predictions and the actual points. Following this, using the cpopt for this residual data a regression tree is fitted to the data. Then the predictions from the initial regression tree (pred1) and the second regression tree (pred2)are added together. Then a third iteration follows. The residuals are calculated from the pred1+pred2 and using the cpopt another regression tree is fitted to the data (pred3). Then the predictions from each iteration are added together (pred1+pred2+pred3) for the final prediction. These can be found at "regtreepred'. These predictions are then translated in to Start, End, Chr, ChrN, log2r, location, and information for each segment in the genome. This can be found in the 'segments' dataframe.

The iterseg.genome function returns a png file at the png_filename that the user provides. The WholeGenome.Plot function is used to create the plot. The optional inputs of upper.y.lim and lower.y.lim control the range of the y values on the plot.

A cpvalue can be specified in the input. By specifying a cp value, this cp value will be used to segment each chromosome. If no cp value is specified the optimal cp value will be used. The conservative optimal cp value can be specified in the input.

iterseg.chr

Inputs: dataframe(required), chromid (required), cpvalue (optional), conserve (optional)

Outputs: A list containing a dataframe with all of the segmentation data (segments), and a dataframe with the predictions from the regression tree (regtreepred), a plot of the chromosome data and final predictions (chrplot), a list 5 plots of each step in iteration (plots), and the cp values used for each chromosome on the first iteration (cpdf)

The iterseg.chr function segments all the data from the dataframe. The dataframe must have columns named specifically "Start.Pos", "log2r", and "Chr" (more about the specifications can be found in the section 'Data input specifications'). The dataframe can include any number of chromosomes. All will be segmented but only information on the chromosome of interest will be returned. The "chromid" indicates the chromosome of interest and it should be entered in the same format as the Chr column in the dataframe. The data is segmented using a regression tree with the optimal cp for the specific chromosome. The predictions from this iteration are the first iteration (pred1). Then the residual error is calculated between the predictions and actual data points. using the cpopt for this residual data a regression tree is fitted to the data. Then the predictions from the initial regression tree (pred1) and the second regression tree (pred2)are added together. Then a third iteration follows. The residuals are calculated from the pred1+pred2 and using the cpopt another regression tree is fitted to the data (pred3). Then the predictions from each iteration are added together (pred1+pred2+pred3) for the final prediction. These can be found at "regtreepred'. These predictions are

then translated in to Start, End, Chr, ChrN, log2r, location, and information for each segment in the genome. This can be found in the 'segments' dataframe.

The final predictions can be found plotted in 'chrplot'. Seeing the step by step for the 3 iterations can be seen in 'plots'.

A cpvalue can be specified in the input. By specifying a cp value, this cp value will be used to segment each chromosome. If no cp value is specified the optimal cp value will be used. The conservative optimal cp value can be specified in the input.

Segmenting using optimal cp, regression tree iteration, and weighiting:

iterseg.genome.weighted

Inputs: dataframe(required), png_filename(required), upper.y.lim (optional), lower.y.lim (optional), cpvalue (optional), conserve (optional)

Outputs: A list containing a dataframe with all of the segmentation data (segments), and a dataframe with the predictions from the regression tree (regtreepred), a plot of the chromosome data and final predictions (chrplot), a list 5 plots of each step in iteration (plots), and the cp values used for each chromosome on the first iteration (cpdf)

The iterseg.genome.weighted function works the same as iterseg.genome, but it weights the points. The weighting works to incentivize the regression tree to reach for the points that are farther from a log2ratio of 0. If the log2ratio has a magnitude less than or equal to the mad.diff of all the log2ratio points, then the weight of the point is 1 (it is a "regular" point). This also avoids decimal weights, which seemed to cause issues. If the log2ratio has a magnitude greater than the mad.diff, then the weight is the magnitude of the log2ratio/mad.diff. Thus, a larger log2r will have a larger weight to act as incentive for the regression tree.

iterseg.chr

Inputs: dataframe(required), chromid (required), cpvalue (optional), conserve (optional)

Outputs: A list containing a dataframe with all of the segmentation data (segments), and a dataframe with the predictions from the regression tree (regtreepred), a plot of the chromosome data and final predictions (chrplot), a list 5 plots of each step in iteration (plots), and the cp values used for each chromosome on the first iteration (cpdf)

The iterseg.chr.weighted function works the same as the iterseg.chr.weighted function, but it weights the points. The iterseg.genome.weighted function works the same as iterseg.genome, but it weights the points. The weighting works to incentivize the regression tree to reach for the points that are farther from a log2ratio of 0. If the log2ratio has a magnitude less than or equal to the mad.diff of all the log2ratio points, then the weight of the point is 1 (it is a "regular" point). This also avoids decimal weights, which seemed to cause issues. If the log2ratio has a magnitude greater than the mad.diff, then the weight is the magnitude of the log2ratio/mad.diff. Thus, a larger log2r will have a larger weight to act as incentive for the regression tree.