RankNormalize Documentation

Description: Ranks the intensity values for genes by sample within a dataset (columns in the file) and then normalizes the ranks by a common multiplier.

Author: Pablo Tamayo, David Eby, gp-help@broadinstitute.org

Introduction

Rank normalization allows comparison of datasets that were created on different platforms. For microarray data, RankNormalize replaces measurements (i.e., gene expression abundance) with their rank within each sample, and allows the user to normalize, or scale, those ranks to facilitate comparison with other datasets. This approach preserves the ordering of genes in a dataset while removing the incompatibilities arising from the differences in platform technologies.

Algorithm

Note: A pseudocode representation of this algorithm is provided at the end of this document.

Step 1: Ranking the dataset

In the ranking step, each data point is replaced by its rank within its column. That is, for a dataset with N genes (rows), the data points will be replaced by values ranging from 1 (lowest) to N (highest). Each value corresponds to the rank of the data point within that sample (column). If two or more genes have the same data point in a sample, their data points are replaced by a value that is the average of their ranked positions. For example, if the data points in positions 4 and 5 of the ranked ordering are equal, they will be replaced by 4.5 (see Examples, below).

Step 2: Scaling the dataset

The maximum value of values in a column is determined by the *scale to value* parameter. For a *scale to value* of V and dataset containing N rows, the final results will therefore range from V/N to V. If you do not enter a value for the *scale to value* parameter, RankNormalize will not scale the ranked results.

Examples

Ranking:

- Given a microarray dataset consisting of 5 genes and 2 samples:

Name	SampleA	Sample B
gene1	1.2	7.0
gene2	0.8	4.4
gene3	3.5	3.7
gene4	8.4	9.0
gene5	0.5	0.4

- For each sample, the algorithm replaces the value of each gene with its rank in that sample, in ascending order, i.e., the lowest value for each sample is replaced with 1:

Name	SampleA	Sample B
gene1	3	4
gene2	2	3
gene3	4	2
gene4	5	5
gene5	1	1

Where two or more genes have the same value in a sample, they are given the same rank. This rank is decided by averaging the ranks occupied by all of the similar values.

For example:

- Given a microarray dataset consisting of 7 genes and 2 samples:

Name	Sample A	Sample B
gene1	1.2	7.0
gene2	0.8	4.4
gene3	3.5	3.7
gene4	8.4	9.0
gene5	0.5	0.4
gene6	1.2	7.0
gene7	1.2	6.5

In Sample A, gene5 is ranked 1, gene2 is ranked 2, but gene1, gene6, and gene7 all have the same value. These genes occupy ranks 3, 4, and 5. RankNormalize averages these ranks and therefore gives gene1, gene6, and gene7 all their average rank of 4, as shown:

Name	Sample A	Sample B
gene1	4	5.5
gene2	2	3
gene3	6	2
gene4	7	7
gene5	1	1
gene6	4	5.5
gene7	4	4

Adjusting the Dataset Before Ranking

There are three additional parameters that can be used to constrain or adjust the dataset before ranking:

- The *threshold* parameter sets the minimum for values in the dataset. Any value below this will be increased to *threshold* before ranking.
- The *ceiling* parameter sets the maximum for values in the dataset. Any value below this will be decreased to *ceiling* before ranking.
- The *shift* parameter provides an amount to adjust values in the dataset. This *shift* adjustment value will be added to every value in the dataset before ranking.

Note: The order in which these parameters are applied is *threshold*, then *ceiling*, then *shift*.

These parameters are used to restrict or adjust the dynamic range of the dataset, repositioning the data points (*shift*) or enforcing minimum (*threshold*) or maximum (*ceiling*) boundaries on their values. This might be necessary, for instance, with legacy microarray datasets, or if you suspect or know that your data is noisy in the high or low ranges. In the past, it was possible for certain microarray instruments to report negative intensity values and, in general, the scale of values for these was found to be noisy at the bottom of the range, causing issues in downstream analyses. Similar issues may also be seen at the upper end of the range.

Parameters

Name	Description
input file (required)	The dataset to be normalized in GCT or RES format.
output file name (required)	The name to be given to the output file (defaults to <input.file_basename>.NORM.<input.file_extension>)
scale to value (optional)	Result values will be scaled to this value by multiplication after normalization. Leaving this blank will give results scaled from 1 to N (where N is the number of rows).
threshold (optional)	Any value below this will be set to the threshold value before normalization.
ceiling (optional)	Any value above this will be set to the ceiling value before normalization.
shift (optional)	The shift value will be added to all values before normalization.

Input Files

1. <input file>
   A dataset in GCT or RES file format to be normalized by rank.

Output Files

1. <output file name>
   The resulting normalized ranked dataset. The output format will match the input format, either GCT or RES. Any calls in a RES file will be maintained unchanged.

Requirements

If you want to install this module on your own GenePattern server, the RankNormalize module requires R2.15.2 with the following packages:

- getopt_1.17
- optparse_0.9.5

The RankNormalize module uses R's built-in "rank" function.
These R packages will be automatically downloaded and installed when the module is installed. R2.15.2 must be installed and configured independently; for more information, see the GenePattern Administrator's Guide: http://www.broadinstitute.org/cancer/software/genepattern/gp_guides/administrators-guide/sections/r-versions.

Platform Dependencies

Module type:	Statistical Methods
CPU type:	Any
OS:	Any
Language:	R2.15.2

GenePattern Module Version Notes

Date	Version	Description
3/15/13	1	Initial version.

Pseudocode for RankNormalization

This pseudocode is included because it may be of interest to some users.

The rank normalization algorithm requires a Rank function that takes a list of values and returns in their place a list of their rank positions. This works as follows:

```
L := [v1, v2, …, vN]    # a list of values to be ranked
Rank(V):
     # Get a copy of the list, with values sorted from least to
     # greatest.
     # That is:
     #    s[1] ≤ s[2] ≤ …  ≤ s[i] ≤ s[i+1] ≤ …  ≤ s[N]
     # Thus, the index of each list item equates to its ranking
     position.
     S := sort(L)

     # Create a lookup table from value to ranking position
     # Correct for ties in the sorted list by average as we go.
     tieCount := 0; tieTotal := 0; tieValue := NULL
     T := new table()
     for i := 1 to N:
        if (S[i] = tieValue)    # we have a tie
           tieTotal := tieTotal + i
           tieCount := tieCount + 1
        else
           T(S[i]) := i
           # If we are coming off a run of ties, find the avg
           # and use that as the ranking position for that
           # value.
           if (tieCount > 1)  T(tieValue) := tieTotal/tieCount

           tieCount := 1; tieTotal := i; tieValue := S[i]
     # end for

     # Check if S ended in a run of ties and treat as above
     if (tieCount > 1) T(tieValue) := tieTotal/tieCount

     # Now, create a new list of ranking positions corresponding
     # to the order of the original list of values.
     R = new list(size=N)
     for i  := 1 to N:
        R[i] := T(L[i])
     # end for

     Return R
```

Note that Rank is an R built-in function, so the actual implementation will vary from the above for efficiency purposes. It is conceptually the same, however. For more details on sorting algorithms, see this Wikipedia article.

With the Rank function in place, we can now describe Rank Normalize. Given a dataset of M samples over N genes organized as a GCT or RES file representing an MxN matrix of intensity values IV and using a scale.to.value of V, Rank Normalize works as follows:

```
RankNormalize(IV):
      # If specified, apply threshold, ceiling and shift
      if (threshold) IV := threshold(IV)
      if (ceiling) IV := ceiling(IV)
      if (shift) IV := shift(IV)

      Norm = new matrix(size[M][N])
      for i := 1 to M:
         # Get the intensity values for sample i as a list of
         # values.
         L := column(IV, i)

         # Find the ranking positions of these values
         R := Rank(L)

         # Adjust each ranking using the scale.to.value
         for j := 1 to N:
            Norm[i][j] := R[j] x (V/N)
         # end for
      # end for

      Return Norm
```

Here again, though conceptually equivalent, the actual implementation is different for efficiency purposes. Those interested in further details can review the rank_normalize.R file within the module.