Function:Troilkatt/Troilkatt Stages

From FunctionWiki

This page describes the general *Troilkatt pipeline* that provides data conversion stages to convert data downloaded from GEO to a format that can be used by the data exploration tools.

See also https://bitbucket.org/peak/spell-tools

Contents

Overview

The pipeline stages are as follows:

1. convertSoft2Pcl or seriesFamilyParser or cel2pcl: convert respectively a GEO GSD dataset, GEO GSE series, or GEO supplementary CEL files to the PCL format.
2. MissingValues: Set some low-quality values as missing and filter out genes and datasets with too many missing values.
3. runKnnImpute: Fill in missing values with their most likely values.
4. mapGeneNames: Map all gene names to the Entrez Gene ID.
5. MeanGenesThatAgree; Average together duplicate genes.
6. collectFinalData: Final numeric cleanup and consolidation.
7. Distancer: Calculate gene-gene correlation.
8. Dat2QDab: Convert .dat files to .qdab files.

File formats

The main file formats used by the pipeline are the GEO input files:

- GDS SOFT file
- GSE SOFT family file
- CEL raw data file

The internal and output file formats are:

- PCL expression table files
- INFO meta data files
- QDAB correlation files

All except QDAB are tab delimited text files. A detailed description follows.

GDS soft

GDS soft files are well defined, well structured files created by GEO curators. The file format and content is described here: http://www.ncbi.nlm.nih.gov/geo/info/soft2.html. For additional information refer to the convertSoft2Pcl source code that implements a GDS soft file parser

GSE soft

The GSE (family) soft files are in the same format as the GDS soft files. However, these files are submitted by the GEO contributors, and the content such as table header names may differ between datasets and organisms. Some files also do not contain any expression values. These files are therefore much harder to parse than the GDS soft files, as demonstrated by the seriesFamilyParser (described below).

PCL

See also: [1] (http://qed.princeton.edu/main/PCL_Format)

The "Pre-CLustered" file format was developed by the Stanford Microarray Database for storing data about genes and experiments. It is a tab-delimited text file that in addition to header rows and columns, there is one gene per row, with one column per sample.

The **first row** gives column headings for the subsequent rows:

- **Column 1** (required): the name of a unique identifier (required in each row); the name is usually UID, GID, ORF, or simply ID. However, most processing steps ignore the actual name used in the cell.
- **Column 2**: the name, usually NAME, of an alternative name. This column is typically ignored, or set to the same values as Column 1.
- **Column 3**: GWEIGHT (optional but highly recommended). This is a per-row analog of the per-column EWEIGHT. It provides the option of weighting samples differently. However, it is usually set to *1* for all samples.
- **Column 4..**: description of the conditions under which the values in this column (in rows 3 and up) were obtained. This names may be used as column identifiers by data exploration tools.

The **second row** specifies the EWEIGHTS for the samples:

- **Column 1**: is EWEIGHT.
- **Column 2**: is blank.
- **Column 3**: is blank.
- **Column 4..**: the weights for this sample. Typically set to 1.

The **subsequent rows** provide expression values for a gene. The columns correspond to the descriptions given in Row 1:

- **Column 1** (required): a unique gene IDENTIFIER for a gene. Before mapping gene identifiers to a common namespace this value is organism and instrument specific. After mapping it is typically the Entrez Gene ID.
- **Column 2** (optional): the NAME of the gene. Typically set to the same value as Column 1

- **Column 3** (optional): the GWEIGHT value. Typically set to 1.
- **Column 4...**: numerical values (generally representing microarray intensity log-ratios). Missing values can be blank, or set to zero.

INFO

The info file contains meta-data used later in the pipeline (in bold) and various other meta-data that is useful for data presentation tools.

Each row in the info file contains the following 24 fields separated by a tab character.

The meta-data extracted from the SOFT file is:

- Source filename
- **Dataset ID** (used as key for lookup in the info file)
- **Organism** (mapGeneNames, collectFinalData)
- Platform
- ValueType
- Number of channels
- Title
- Description
- PubMedID
- Number of features/columns
- Number of samples/rows
- Date

The meta-data calculated for the expression values is:

- Min value
- Max value
- Mean value
- Number of negative values
- Number of positive values
- Number of zero values
- Number of missing values
- Number of total values
- Number of calculated channels
- '*Is dataset log transformed* (runKnnImpute, collectFinalData)
- **Are zeros treated as missing values** (used in insertMissingValues)
- **Missing value cut-off value** (used in insertMissingValues)

convertSoft2Pcl

Short description: This stage converts a GEO GDS soft file to the PCL format, and extracts meta-data information saved in an info file.

Source code: troilkatt/scripts/array_utils/spell/convertSoft2Pcl.rb

Input: GEO GSD dataset .soft file.

Output: .pcl file and .info file.

Expected failures: none. All GDS files should successfully be converted.

Description. The .soft file is parsed to find the dataset table with the expression values to be written to the .pcl file, and to

find meta-data tags written to the .info file. In addition some additional meta-data information is calculated for the expression values and written to the .info file.

The *dataset table* format in a GSD soft file is well defined such that the PCL data table headers and expression values are found by parsing this format (refer to the source code for details). The EWEIGHT and GWEIGHT values are always set to *1*.

The .info file is created by parsing the well defined GSD meta-data values, and by doing calculations for the missing values (refer to the INFO subsection above for details).

seriesFamilyParser

Short description: this stage parses and converts a GEO GSE soft file to the PCL format and extracts meta-data information that is stored in an info file.

Source code: troilkatt/src/edu/princeton/function/troilkatt/tools/SeriesFamilyParser.java

Input: GEO GSE series .soft file.

Output: .pcl file and .info file.

Expected failures: non-microarray data will often fail since the gene name and/or expression value columns cannot be found. There is no solution for this failure since these files typically do not contains expression values.

Description. The SeriesFamilyParser creates a .pcl and .info file similar to the convertSoft2Pcl script. However, the GSE series .soft file format is not well structured so it is necessary to guess which columns in each *sample table* contains the platform probe ID, gene names/IDs, sample probe ID's, and the expression values. The *guessing* is done by searching for keywords in the table headers. For the platform probe ID, sample probe ID, and, expression values the keywords are respectively: "ID", "ID_REF" and "VALUE". For the gene names the keywords are: "GENE_SYMBOL", "GENE SYMBOL", "GENE_NAME", "GENE NAME", "GENE ID", "ORF", "DDB". This list may need to be expanded when parsing a new organism since the keyword may be organism specific.

The info file calculation is the same as for GSD files.

cel2pcl

Short description: a set of GEO AffyMetrix CEL files are normalized and converted to a PCL file.

Source code: troilkatt/scripts/cel2pcl.py

Input: GEO series supplementary data in a tar archive.

Output: one or more .pcl files.

Important arguments:

- Maximum virtual memory size: for big datasets the R script run as part of the conversion may use a lot of memory and may crash the machine or take forever to run.
- BrainArray CDF files are required for the specific platform used to create these files.
- gse.info file that contains the meta-data for this series (extracted earlier from the .soft file).

Expected failures:

- If the R calculation uses more than the specified maximum virtual memory size (8-24GB) the conversion will fail.

memory size can be increased.

- If there is no CDF file from BrainArray for the specific platform the program will exit.

Description. The cel2pcl scripts first unpacks the archive, uncompresses all CEL files, splits the CEL files into groups if necessary, and then runs the 'troilkatt/scripts/R/ProcessCEL.R" R script on each group of CEL files. It adds the necessary header rows and columns to the R script output file, and finally creates the info file by running RawPclParser.

The R script does the necessary normalization in order to create the PCL file. The RawPclParser (troilkatt/src /edu/princeton/function/troilkatt/tools/RawPclParser.java) calculates statistics for the expression values, and then merges the calculated values with meta-data extracted from the gse.info file.

It may be necessary to split a set of CEL files into groups if the memory requirements of the R script are above the maximum allowed virtual memory size on the system. The split attempts to: average together *replicas* (**TODO**), and to split *time series* by condition (**TODO**). Otherwise the CEL files are split into groups randomly. (Another possibility may be to modify the R script to do the normalization in multiple steps). In case of a split one PCL file is created for each group. These should not be concated.

RemoveOverlap

Short description: Overlapping samples from a PCL file from either seriesFamilyParser or cel2pcl are removed.

Source code: troilkatt/src/edu/princeton/function/troilkatt/tools/RemoveOverlap.java

Input: .pcl file

Output: zero, one, or multiple .pcl files

Important arguments:

- The gse.overlap file that contains information about duplicate series, and series with overlapping samples.

Expected failures:

Description.

There can be four types of overlapping series, that are dealt differently:

1. Duplicate: series A and B has identical set of samples.
2. Superset: superset series A contains all samples in subsets series B, C,…N.
3. Proper subset: all samples of dataset A are in dataset B, but B is not a superset.
4. Subset: some, but not all, samples in dataset A are in dataset B.

Ideally the datasets would be split such that similar experiments are grouped together. This can be done manually by examining the column labels that contain the experiment description. However, it is not practical to automate the label parsing so instead the series files are split or deleted according to the following rules.

For *duplicates* the newest series is kept, the older series is discarded.

Supersets are deleted.

For *proper subsets* the rules are:

1. Keep all subsets with at least 3 samples.
2. Keep the unique samples and samples from subsets with fewer than 3 samples in the inproper superset.

For *subsets* the rules are:

1. Keep all subsets with at least 3 samples.

2. Keep the unique samples and samples from subsets with fewer than 3 samples in the inproper superset.
3. (may also be possible to make decision based on Jaccard index).

The motivation for keeping the subsets and deleting the supersets is that the data provides more power to the prediction models if the data is separated. Also, for these models it is not a problem if a dataset has few samples. It is also not a problem if there is a small 1-3 sample overlap between series. For all cases except duplicates, the creation date for the series is ignored.

The gse.overlap file contains the information about duplicates, supersets, and subsets. This file is created by: **...TODO...** The output file contains:

- List of GSE IDs for series that have a duplicate with a newer creation date-
- List of superset GSE IDs.
- Set of trees built bottom-up by connecting all (proper and non-proper) subsets to their supersets. Each node in the tree contains the GSE ID of the series represented by the node, the number of samples in the series, and the GSM IDs of its samples.

The RemoveOverlap java program loads the two lists and the set of trees from the gse.overlap file. It then uses the GSE ID found in the input filename, to do the following checks:

1. If the ID is in the duplicates list: return without creating an output file.
2. If the ID is in the supersets list: return without creating an output file.
3. Find the node for the series in one of the subset trees. If the node is a leaf node with more than 3 samples; output the input file content. Otherwise output all samples not in a child node with 3 or more samples.

Note! this step should not be run for cel2pcl data intended for Spell since it may loose data.

MissingValues

Short description: Fill in missing values and remove genes and datasets with too many missing values.

Source code: troilkatt/src/edu/princeton/function/troilkatt/tools/MissingValues.java

Input: .pcl and .info file

Output: converted .pcl file (or no output if too many missing values)

Important arguments:

- Zeros as missing values from info file.
- Missing value cut-off value form info file. Expression values less than the cut-off are set as missing.
- Gene discard ratio: genes with more than 50% missing values are discarded.
- Dataset discard ratio: datasets with more than 50% missing values are discarded. The total values count does not include values in discarded genes.
- Minimum samples: datasets with fewer than 3 samples are discarded.

Description. This program sets a value in the output PCL file if:

- The zeros as missing values field in the info file is true and the expression value is zero.
- If the missing value cut-off is a valid number and the expression value is below the cut-off value.
- If the expression value is not a valid floating point number.

The program prune bad data by:

- Discarding all genes with more than 100% missing values.
- Discard all datasets with more than 50% missing values in the included genes.

runKnnImpute

Short description: estimate missing values.

Source code: troilkatt/scripts/array_utils/spell/runKnnImpute.rb

Input: .pcl file and .info file

Output: imputed .pcl file

Important arguments:

- Number of channels from info file.
- Log transform from info file.
- Maximum virtual memory size (default 8GB): for big datasets the KNNImpute program may use a lot of memory and may crash the machine or take forever to run.
- KNNImpute arguments: -k 10 -m 0.7 -d euclidean

Expected failures:

- If KNNimpute uses more than available (8-24GB) it will crash.

Description. If the log transform field in the info file is false the expression values in the input value are log transformed (Math.log(val)). Then the Sleipnir KNNImpute program is run on either the input file or the log transformed input file using the above arguments. Finally, if the log transform field in the info file is false the imputed values are expoeniated (Math.exp(val)).

mapGeneNames

Short description: Convert gene names to a common namespace.

Source code: troilkatt/scripts/array_utils/spell/mapGeneNames.rb

Input: imputed .pcl file, info file, and gene mapping file

Output: mapped .pcl file

Important arguments:

- Organism name from info file
- Gene map file

Expected failures:

- If the input file has invalid or unknown gene names this script will produce an empty file that will crash the next stage.

Description. The gene map file contains mappings of all possible gene names to the Entrez ID as follows:

```
if row[gene ID] in geneMap:
   row[gene ID] = geneMap[gene ID]   # Map gene ID to Entrez ID
   row[gene Name] = geneMap[gene ID] # and also map the Gene Name to Entrez ID
   write(row)
else if row[gene Name] in geneMap:
   row[gene ID] = geneMap[gene Name]   # Map gene ID to Entrez ID
   row[gene Name] = geneMap[gene Name] # and also map the Gene Name to Entrez ID
   write(row)
```

```
#else: discard row
```

Note that the gene IDs and names read from the PCL and map file are converted to uppercase before doing the mapping.

The gene map file is generated by the troilkatt/scripts/parseEntrezGeneInfo that parses the Enztrez gene_info file for an organism to create the gene mappings using the following rules:

1. The taxonomy ID for the row should match the taxonomy ID for the organism
2. The type of gene is "protein-coding".
3. For rows with duplicate gene ID, the gene ID of the latter is changed to the gene ID of the first
4. Add all synonyms that map to a single gene ID.
5. Add all database synonyms that map to a single gene ID.
6. Add the symbol systematic name to gene ID mapping.
7. Add the systematic name to gene ID mapping.
8. Add the identity gene ID to gene ID mapping.
9. Add additional synonym to gene ID mappings read from additional files, but filter out all that do not map to a valid gene ID.

MeanGenesThatAgree

Short description: Average together the expression values duplicate genes.

Source code: troilkatt/src/edu/princeton/function/troilkatt/tools/MeanGenesThatAgree.java

Input: mapped .pcl file

Output: averaged .pcl file

Description. Average together duplicate genes. Pseudo code is as follows

```
# geneSets is a set of genes in a pcl file with the same gene ID/Name
```

```
for gene in geneSets.keys:
  if geneSets[gene].len == 1: # Only one row with this gene name in file
    write(gene, geneSets[gene].expressionValues)
  else:
    # Count number of genes that agree
    geneRows = geneSets[gene]
    for i in range(0, geneRows.size):            # for each gene pair
      for j in range(i + 1; geneRows .size):
        if geneRows [i] agree with geneRows [j]: # count number of genes that "agree"
          numAgree[i]++
          numAgree[j]++
```

```
    # Include a gene if it agrees with at least half of the others in the set
    for i in range(0, geneRows .size):
      if numAgree[i] >= ((numAgree.size - 1) / 2.0):
        numGood++
        update mean expression values with geneRows[i] values
```

```
    # If at least half of the genes are included, use the mean
    if (numGood > (numAgree.size / 2.0)):
      write(gene, meanExpressionValues)
    # else: discard the gene
```

To determine if two genes *g1* and *g2* agree the following formula is used:

```
if normalizedDistance(g1, g2) > fCutoff: true
```

where fCutoff is

```
bgMean + 0.5 * (mapMean - bgMean)
```

where *bgMean* is then mean calculated for an array distances created by:

```
for i in range(0, rows.length):
  for j in range(0, rows.length):
    if randomValue() < 0.02 and rows[i].name != rows[j].name:
      distances.add( normalizedDistance(rows[i].expressionValues, rows[j].expressionValues)
```

and *mapMean* is the mean for the array distances created by:

```
for set in geneSets where set.size > 1:
  for i in range(0, set.size):
    for j in range(i + 1, set.size):
      distances.add( normalizedDistance(set[i].expressionValues, set[j].expressionValues)
```

collectFinalData

Short description: do final log transformation if needed.

Source code: troilkatt/scripts/array_utils/spell/collectFinalData.rb

The ruby script runs the Java program in: *troilkatt/scripts/array_utils/spell/DivLogNorm.java*

Input: averaged .pcl file and .info file

Output: final .pcl file

Important arguments

Arguments of collectFinalData.rb:

- the filepath of the pcl file
- the filepath of the .info file
- directory containing DivLogNorm.jar
- output directory

Values for the following variables are taken from the .info file:

- Organism from the info file.
- Is log transformed from the info file.

In addition, the DivLogNorm arguments are statically set to **0 1 0** (do not divide by individual medians, log transform if needed, do not normalize).

Description. If the log transformed field in the info file is false, the DivLogNorm program is run with the "0 1 0" arguments which will:

- Not divide all genes by individual medians.
- Log transform all expression values.

- Not normalize all genes.

The log transform is implemented as follows:

```
if (val < 0)
  val = -Math.log(-val) / Math.log(2);
else if (val > 0)
  val = Math.log(val) / Math.log(2);
else
  val = Math.log(0.001) / Math.log(2);
```

pcl2qdab

Short description: convert a .pcl file to a .qdab (pairwise distances) file.

Source code: troilkatt/scripts/pcl2qdab.py

In addition the scripts runs Distancer and Dat2Dab from Sleipnir.

Input: final .pcl file

Output: qdab file (compressed quantified pairwise gene distances).

Important arguments:

- quant.file: quantifier file. A file with the following content: "-1.5\t-0.5\t0.5\t1.5\t2.5\t3.5\t4.5\n"
- Maximum virtual memory size (default 8GB)

Expected failures:

- If more than maximum virtual memory is used the program will crash.

Description. The pcl2qdab will run the Sleipnir Distancer program on the final .pcl file to create a .dat file. This file is then used as input to the Dat2Dab program. Dat2Dab reorders the pairwise distnaces in the .dat file such that these can be read efficiently from disk, and then compresses the content using the quantifies in quant.file. The final file is a .qdab file.

PclCombiner

Short description: combine .pcl files into one big .pcl file.

Source code: troilkatt/src/edu/princeton/function/troilkatt/mapreduce/PclCombiner.java

Input files:' .pcl files to be combined

Output files: one big output file

Important arguments:

- Minimum samples: minimum samples for files to be combined (default 1)
- Maximum samples: maximum samples for files to be combined (default -1: no maximum limit)

Description. Start a MapReduce job, where mappers check if number of samples in a .pcl file are withing the minimum and maximum limit, and output the content of files to be combined to a reducer task that creates the final combined file. This stage implements a MapReduce version of the Sleipnir combiner

normalize

Short description: do normalization on PCL files.

Source code: troilkatt/scripts/normalize.py

Also uses Normalizer from Sleipnir.

Input files: .pcl file to be normalized

Output files: normalized pcl file

Important arguments:

Expected failures:

Description. For each dataset the Sleipnir Normalizer tool is run to transform each gene's expression vector to have mean zero and standard deviation.