LeMoNe tutorial: module network inference from mRNA and microRNA expression data

Eric Bonnet

December 7, 2010

1 Introduction

In this tutorial, we will use two different expression data sets, corresponding to mRNA and microRNA expression data measured using specific microarray platforms.

The first step (clustering) is done on the mRNA expression data only. We did a selection of genes having non-flat profiles, keeping genes having a standard deviation above a certain value. We usually use 0.5 as the cutoff score, but this might depend on the dataset (having a look at the distribution is always a good idea). In this example, we have an input matrix of 499 genes and 90 samples. The data is scaled (by row) to have a mean of 0 and a standard deviation of 1.

No selection is done for microRNA data, just scaling the matrix to have mean 0 and standard deviation of 1.

2 Clustering genes: ganesh task

The LeMoNe package has been installed (unzipped) in the LeMoNe_v2.5 directory.

For this step, we use the filtered gene matrix (mRNA.txt) located in the data/ directory.

The command to generate one clustering solution is (should be on one line):

```
java -Xmx1000m -jar LeMoNe_v2.5/lemone.jar -task ganesh -data_dir data/
-data_file mRNA.txt -output_file net0 -init_num_clust 250
```

For the initial number of clusters, the rule of thumb is to take half the size of the matrix. In this case, we have around 500 genes, so we choose 250 for the parameter init_num_clust.

This step will be repeated multiple times, using the same command, only changing the name of the output file (usually ten times, this can be easily parallelized on a computer cluster). Here we have generated 5 runs, prefixed net0, 1, 2, 3, 4. The files are in the ganesh_xml directory.

3 Printing the fuzzy matrix: fuzzy task

The fuzzy matrix is just a matrix indicating which cluster each gene belongs to, for all the clustering solutions generated. The command line is as follow:

```
java -Xmx1000m -jar LeMoNe_v2.5/lemone.jar -task fuzzy
-data_dir ganesh_xml/ -output_file fuzzy
```

Now we have the fuzzy.txt matrix in the current directory, that will be used as input for the next step.

It is very important to put **only** lemone files (i.e. .xml.gz files) in the directory indicated by the data_dir options. The program is not very smart and will try to open any file in the directory, provoking an error if the file does not correspond to the lemone format.

4 Extracting tight clusters: matlab procedure

The extraction of tight clusters from the fuzzy matrix is done by using matlab scripts (for efficiency). Here, we assume that matlab is installed on your computer.

After you've launched a matlab session, you need to add the path to the matlab code.

```
addpath('LeMoNe_v2.5/matlab/')
```

The next step converts the fuzzy matrix to a probability matrix (s).

```
[s,data,idx] = makepairwiseprob('fuzzy.txt');
```

Here we filter out (set to 0) values below 0.2, for a better clustering.

```
[s,n1,n2,nc] = pcutoff(s,0.2);
```

Iterative clustering of the adjacency matrix to produce fuzzy clusters.

```
p = matrixClustSym(s);
```

Define optimal probability cutoff for fuzzy clusters.

```
[xopt, qopt,n1,n2,nc] = optCutoff(p,s);
```

Extract tight clusters according to the cutoff probability and write the results to an output file.

```
writeclusters(qopt,idx,'tight_clusters.txt')
```

Here, we select only clusters having at least 4 genes, using a simple perl script (the clusters numbers are also changed, to start from 0).

```
perl lemone_select_clusters 4 < tight_clusters.txt > tc4.txt
```

We have a final set of 19 clusters, for a total of 361 genes.

5 Assigning regulators: regulators task

For this task, we have selected 10 transcription factors (based on their GO annotation) and 30 microRNAs as candidate regulators. The transcription factors were not selected in the previous matrix, because they have a relatively flat expression profile. Nevertheless, it is well known that transcription factors frequently have such flat profiles, so we want to add them as candidate regulators anyway for this step of the procedure. A new data matrix is build with all the cluster genes, all the transcription factors and all the microRNA expression profiles (all.txt). A global list of all the candidate regulators ID codes is also build (reg_mir_list.txt).

The first step of the regulators' assignment is to build hierarchical trees grouping conditions (samples) having similar gene expression levels. The same clustering algorithm as in the ganesh step is actually used here. The clusters of conditions are then linked hierarchically to produce the trees, and regulators are assigned to each node of each tree, depending on how well their relative expression data match the split defined by the node. Finally, a global score for each regulator assigned is calculated, taking into account the score of the regulator in all the trees. The global score is used to rank (or prioritize) all the regulators for a given module.

The following command is building 10 conditions clusters for each cluster of co-expressed genes, thus defining 10 hierarchical trees, and assigning 10 regulators per node of each hierarchical tree (those are the default values that can be changed at will).

```
java -Xmx1000m -jar LeMoNe_v2.5/lemone.jar -task regulators -data_dir data/
-data_file all.txt -reg_file reg_mir_list.txt -cluster_file tc4.txt
-output_file reg -num_clust 19
```

The output file is reg.xml.gz. It is moved to a new directory named reg_xml.

6 Printing the list of regulators: print_regulators task

For each cluster of co-expressed genes, we have now a list of regulators with their score. As we want to retain only the best ones, we usually take the top 1% of the complete list, which is the default behavior of this command:

```
java -Xmx1000m -jar LeMoNe_v2.5/lemone.jar -task print_regulators
-data_dir reg_xml/  -output_file reg_1p.txt
```

The output file has three columns: the gene ID, module ID and the global score of the regulator. If we want to have the complete list of regulators, for example to select regulators with another threshold, we can use the all_regulators flag.

```
java -Xmx1000m -jar LeMoNe_v2.5/lemone.jar -task print_regulators
-data_dir reg_xml/  -output_file reg_all.txt -all_regulators
```

7 Creating figures: figures task

This task is creating one figure per module. The figure represent the expression values color-coded with a gradient from dark blue (low expression values) to bright yellow (high expression values). All the module genes are in the lower panel while the top regulators (if any) are displayed in the upper panel. One of the regulation trees is represented on top of the figure, with the different split points highlighted on the figure as vertical red lines.

The name of each gene is displayed on the left and the name of each condition (or experiments, or samples) is displayed at the bottom of the figure. In this example, the genes name in the data matrix is in fact the chip probe name. We want to replace them on figure by the gene symbols and for that we use a *map file*, which is a simple two columns text file where for each probe name we have also the corresponding symbol (if any).

The figures are created in encapsulated postscript format (eps). If a module does not have any high scoring regulator selected, the figure will not draw the hierarchical tree ("leaf" type, opposed to the "tree" type, as indicated in the file name). In this toy example, as we have a few high-scoring regulators, only a subset of the modules will be drawn with the tree (module 9, for example).

```
java -Xmx1000m -jar LeMoNe_v2.5/lemone.jar -task figures -data_dir reg_xml/
-top_regulators reg_1p.txt -map_file all_genes_map
```

8 Calculating GO enrichment: task go_annotation

The goal of this task is to calculate the GO category enrichment for each module. We use the BiNGO library to calculate the statistics. We have to specify a GO annotation file, in this case we have human genes, so we choose the default file H_sapiens_default, provided with the BiNGO library. We also specify the set of genes that should be used as the reference for the calculation of the statistics, in this case the list of all the genes that are present on the microarray chip (file go_ref.txt). The results are stored in the output file go.txt. The GO annotation file is using the HUGO gene symbols, while the data matrix we used has the microarray chip probe names, so we provide a *map file* from probe names to HUGO gene symbols (all_genes_map). The probe codes will be replaced by the gene symbols according to the map file before the GO calculations.

```
java -Xmx1000m -jar LeMoNe_v2.5/lemone.jar -task go_annotation
-data_dir reg_xml/ -output_file go.txt
-go_annot_file H_sapiens_default
-map_file all_genes_map
-go_ref_file go_ref.txt
```