Genomics - Problem Set 2

Part 1 due Friday, 1/24/2014 by 9:00am
Part 2 due Friday, 1/31/14 by 9:00am

One major aspect of functional genomics is measuring the expression of all genes simultaneously. This was initially done using DNA microarrays, although RNA sequencing (RNA-seq) and its variants are becoming increasingly common due to their greater reproducibility and utility in exploring other biological questions. For this problem set, you will use existing tools as well as some custom scripts that you will write to extract biological information from a raw mRNA expression dataset.

Problem: You are studying a yeast gene, YPL267W. You decide to explore published datasets to see how the relative abundance of YPL267W changes, and to see what genes show similar patterns. You hope to gain insight into the role of YPL267W.

A standard method of defining groups of co-expressed genes in a gene expression dataset is hierarchical clustering. We have not covered clustering in class yet, but in general, clustering constructs a tree in which genes with similar expression patterns are correlated (or "clustered together") using either a *neighbor joining* or *UPGMA*-type algorithm applied to a distance matrix. If you would like more information, a good starting point is a paper by Eisen et al.: http://www.pnas.org/content/95/25/14863.full.

There are several software products that perform clustering; we will use a program written by Gavin Sherlock, called *XCluster*, which does centroid-based clustering (to be discussed in lecture). We will work with a data file that is a subset of the data described in:

> Spellman P.T., Sherlock, G., et al. (1998). Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. *Mol Biol Cell* **9**(12):3273-3297.

To do these problems, you will need to use a Stanford FarmShare server (e.g. corn.stanford.edu). The plain text input data file to use can be found at:

```
/usr/class/gene211/misc/cellcycle.pcl
```

Start by making a local copy of this file in your own directory and examine it using any of the UNIX utilities we have worked with (e.g. more or less). The first row of this file (the header) contains the experiment names (here representing a single experimental replicate). The experiments described in this file compared yeast cells synchronized in different phases of the cell cycle with asynchronous yeast cells. The first two columns of the file contain the gene's unique ID and name, and the third column contains a gene weight that can be ignored for this exercise. The remaining columns contain the $\log_2$-transformed expression ratios (that is, 2-fold differences in expression) for that gene in the corresponding experiment.

Part 1:

Question 1: Why are the data represented as $\log_2$ ratios for each gene in each experiment?

Cluster this dataset by executing the XCluster command (you must be in the directory containing your local copy of the `cellcycle.pcl` file, and you *must* be logged onto `corn.stanford.edu`.

```
/usr/class/gene211/bin/cluster -f cellcycle.pcl
```

If you want more information about the clustering software or the file formats it uses, see the online documentation at: http://www.stanford.edu/group/sherlocklab/cluster.html

Question 2: What metrics does XCluster allow you to use to determine the 'distances' between genes? Which is used by default? Does it matter which one you use?

Executing the clustering command above produces two output files: `cellcycle.cdt` and `cellcycle.gtr` (both explained below).

To view the data, download `cellcycle.cdt` and `cellcycle.gtr` to your desktop computer and open the `cellcycle.cdt` file with MapleTree. MapleTree is available from http://sourceforge.net/projects/mapletree/. For MacOSX, click on the MapleView.jar file while holding down CONTROL, select Open, and then in the resulting dialog box click Open. To load the 'cdt' file use MapleView, Load->Hierachical Clustering Data -> Open.

Looking at the data, you should see that plenty of genes show similar expression to your gene of interest, YPL267W. Remembering the analysis we performed in Problem Set 1, you are to download `/usr/class/gene211/misc/SGD_features.tab` to your computer and use Python to search for genes with similar expression to YPL267W.

[FYI: in previous years we used Java TreeView, http://sourceforge.net/projects/jtreeview/, but it does not work on recent versions of MacOSX.]

Programming Exercise: *Write a Python script, `extract_info_better.py`, that will accept as input any number of yeast gene names, feature names, gene aliases, or primary/secondary SGDIDs (or a mixture thereof) on the command line and will extract information about those genes from `SGD_features.tab`. The output of your script should include the following information for each gene in a tab-delimited format:*

1. Primary SGDID (SGD's database identifier)
2. Feature Type
3. Feature Qualifier
4. Feature Name
5. Standard Gene Name
6. Aliases (when more than one, delimited by a '|' character)
7. Parent feature name

8 Secondary SGDID
9 Chromosome
10 Start coordinate
11 Stop coordinate
12 Strand (W or C)
13 Description

As a reminder of the format for SGD_features.tab, it contains the following tab-delimited information:

1 Primary SGDID
2 Feature Type
3 Feature Qualifier*
4 Feature Name*
5 Standard Gene Name*
6 Alias* (when more than one, delimited by the '|' character)
7 Parent Feature Name*
8 Secondary SGDID* (when more than one, delimited by the '|' character)
9 Chromosome*
10 Start coordinate*
11 Stop coordinate*
12 Strand*
13 Genetic position*
14 Coordinate version*
15 Sequence version*
16 Description*

* These fields are optional

Algorithms and Methods:

Your program should be executable as follows:

```
python extract_info_better.py gene1 gene2 … geneN < SGD_features.tab
```

The < operator sends the contents of SGD_features.tab to your Python script via the standard input (sys.stdin). Within your script, you can access each line of SGD_features.tab within a for loop as follows:

```
import sys

for line in sys.stdin:
      # process str variable line
```

In this for loop, your program will read through SGD_features.tab one line at a time, and the loop will exit when there are no more lines to be read. We covered how to split based on the tab

character in Problem Set 1, which you can re-use. You will also need to further split the alias and secondary SGDID fields based on the vertical bar character ('|'), which you do as follows:

```
# convert str alias into list
aliases = aliases.split('|')
```

You can now check each line for a full, *case-insensitive* match to one of the genes in your gene list, which is found in the list `sys.ARGV[1:]`:

```
gene_names = sys.ARGV[1:]
```

Note that in `SGD_features.tab` for each open-reading frame there are a minimum of 2 lines, one for the ORF (analogous to a gene in the yeast world), and one for the coding sequence (CDS, a.k.a. the exonic portion). In the case that the gene is spliced, there will be more than one CDS, as well as one or more introns. It is most useful to print out only the 'top-level' part of the feature – these have a parent feature name that beings with "chromosome" such as "chromosome 1". To test whether a string begins with "chromosome", you can use the startswith method of the string object in this fashion:

```
if parent_feature.startswith("chromosome"):
      # something useful
```

Finally, to make the printed output of your program be saved to a file, you can use command line redirection as covered in Problem Set 0 and Problem Set 1.

Additionally, your script should be *case-insensitive* – this means that the terms 'cdc6' and 'CDC6' should be treated identically so that the user doesn't have to worry about the correct casing of any gene identifier. Finally, your code should be well commented.

Submission of Part 1:

Write your answers to the questions above in a file called `README` and submit this with your script, `extract_info_better.py`. You should also submit a file, `output.txt`, which is generated when you run your script with the following input identifiers:

```
YAL030W TRN1 bin2 PAT1 YRF1 S000003551 ssu21 CEP2 YPK1 ZRG11 MLP1 SUT1 CBP80 CSN11 YIH1
```

These files can be submitted by putting them all in a temporary directory and running the following script, then following the instructions listed:

```
/usr/class/gene211/bin/submit.pl
```

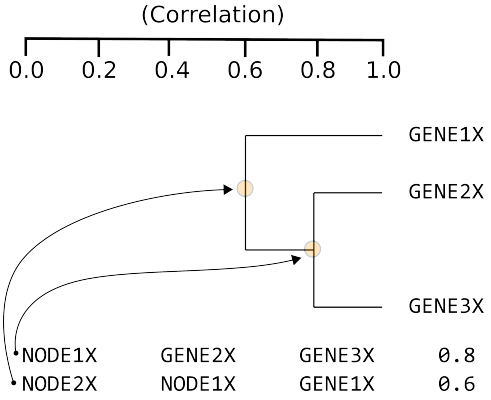
Part 2: The program you wrote in Part 1 proves useful, but it needs to be enhanced. You will be looking at lots of datasets, then cluster after cluster in Java TreeView, and typing many gene names on the command line. Apart from being error prone this process will become quite tedious. What's more, it is important to be consistent from one dataset to the next, so that your definition of 'similar expression' doesn't change over time due to fatigue, boredom, or the caffeine that you just drank in preparation for the task ahead. It occurs to you that you can take advantage of the structure of the cluster itself – what you're really interested in is extracting sub-clusters that include your gene of interest.

Examine the output files that we mentioned earlier, the cluster data file `cellcycle.cdt` and the gene tree file `cellcycle.gtr`. The `.cdt` file is in the same format as the `.pcl` file except:

(1) In the `.cdt` file, the genes (rows) are ordered such that genes that clustered together are adjacent (note that at the boundary of two clusters, genes are adjacent in the list although they did not cluster together)

(2) There is an additional column at the beginning with a new (unique) gene identifier.

The `.gtr` file consists of a number of rows, each of which describes a node. The following is an example tree and its corresponding `.gtr` file.



The first column is the name of a node. The next two columns are the children (branches) of that node. In this case, NODE2X has two branches, one leading to the gene (leaf) GENE1X and the other leading to another node, NODE1X. The last column is the level of correlation between the data of GENE1X and the (average) data of NODE1X, 0.6. The columns are tab delimited.

Goal: You resolve to write a script named `parse_cluster.py` that takes the output from XCluster, the gene (YPL267W) and a correlation cutoff (0.85) as input and prints out all genes (i.e., their ORF name) that clustered with the input gene up to the input correlation cutoff.

Algorithms and Methods:

Your script should be designed to execute like this:

```
python parse_cluster.py cellcycle.cdt cellcycle.gtr gene_name cutoff_value
```

In this program, you will have to open two files. To do that, we cannot read in from standard in like we did in Part 1. In order to manage this, we will have to use some file I/O to access the data found in these files. An example of this is shown below:

```
with open('cellcycle.cdt', 'r') as cdt_file:
      for line in cdt_file:
            # process line
```

NOTE: The problem outlined in this exercise can be solved using a variety of algorithms and implementations of those algorithms – feel free to explore any possibilities that you like, with the only **requirements being:**

- Your program must use `sys.ARGV[1:]` to read in all command-line parameters
- Your script must include at least one function; this function must be passed at least one argument, return at least one value, and must be used directly by your program to find the genes that cluster with the gene of interest

We supply one possible algorithm at the end of the problem set, which uses two functions. You may ignore it and design your own algorithm.

Finally, in addition to answering the questions below, you are expected to use your script to find all of the genes in the cluster that are co-expressed with ORF YPL267W *with a correlation greater than 0.85*. **Note:** You are to consider the highest order branches that have correlations greater than 0.85; i.e. if a higher order node has a correlation > .85 and a lower order node has a correlation < .85, still consider the higher order node and its sub nodes to be clustered together. There should be about 40 genes that should be listed in an output file called `coexpressed_genes.txt`.

Question 3: Use the GO::TermFinder at SGD to get information about the biological roles that these genes may play. What cellular component is enriched for these genes? What do the different-colored boxes displayed by the GO::TermFinder represent? The URL at SGD is:

http://www.yeastgenome.org/cgi-bin/GO/goTermFinder.pl.

For more information on GO::TermFinder check out:

http://www.ncbi.nlm.nih.gov/pubmed/15297299

Question 4: Now that you know where the gene products tend to be located in the cell, repeat the search using the Biological Process Ontology. Why are the p-values so different for the two ontologies? How are the p-values calculated?

Now that you have a list of co-expressed genes, with some clues as to the biological processes in which these genes participate, you would like to learn more about the regulation of these genes.

Q5: Consider the type of data that was used to generate your list of genes. With that in mind, what type of regulation would you expect to underlie these data? Other than this, for what biological reasons might these genes be clustered together?

Submission of Part 2:

Submit your answer to the questions in a file called `README`. Also submit `parse_cluster.py` and `coexpressed_genes.txt`.

These files can be submitted by putting them all in a temporary directory and running the following script, then following the instructions listed:

```
/usr/class/gene211/bin/submit.pl
```

Algorithm Hints For Traversing the Gene Cluster Tree

1. Read in the .cdt file. Make a dictionary indexing the ORF identifiers (e.g. YOL053W) to the gene ID. Record the gene identifier that corresponds to the gene of interest.

2. Read in the .gtr file. For each node and gene in the file make an entry into a 'parent' dictionary with the node ID or gene ID as the key and the node ID of its parent node as the value. Create two 'children' dictionaries, where each node ID is entered as a key in both children dictionaries and indexed to a different child node in each dictionary (every node should have exactly two children, which are either another node or a gene). You will also want to store the correlations in a similar way.

3. To find the co-expressed genes, you should now be able to write two simple functions. One will find the node ID for the highest node on the tree above a gene below any arbitrary correlation cutoff. The second will find all gene IDs for the constituents of a given node. You must then find a way to combine the use of these functions to solve the problem.

NOTE: This is a naturally recursive problem, but you may also be able to solve it without recursion if that's more intuitive to you. Wikipedia has some good information about recursion and tree traversal:

http://en.wikipedia.org/wiki/Recursion_(computer_science)#Recursive_algorithms
http://en.wikipedia.org/wiki/Recursion_(computer_science)#Recursive_programs
http://en.wikipedia.org/wiki/Tree_traversal#Traversal