Take-Home Final Exam: Mining Regulatory Modules from Gene Expression Data

36-350, Data Mining; Fall 2009[*]

Due at 5 pm on Tuesday, 15 December 2009

There are three problems, each with several parts. All three problems are worth 100 points. Please read the background material and the problem statements carefully before you begin. Include all your code.

Each problem will be curved, and there will be a curve for the over-all test as well.

This is a take-home examination. I am trusting you to abide by the university's policy on cheating and plagiarism while working on it (see `http://www.cmu.edu/policies/documents/Cheating.html`).

1 Introduction: Biological Background

1.1 Proteins, Gene Expression, Transcription

The crucial molecules in cells are **proteins**. Cells use them to build structure, to run the chemical reactions which give a cell its energy, to carry signals across the cell, to respond to signals with physical or chemical changes, to regulate what passes through the cell's membranes (including its surface) and to control what the cell sticks to. Specialized cells use proteins to do things like carry oxygen to other cells, produce hair or silk (both proteins themselves), etc., etc.

Proteins are themselves made up of smaller molecules called **amino acids** strung together in a chain. (The chain folds up on itself in complicated ways, depending on the amino acid sequence, giving proteins distinct shapes.) The central insight of molecular biology is that the information which tells an organism how to string amino acids together to make proteins comes from another kind of molecule called DNA. DNA is assembled from strings of smaller molecules called **bases**, and the sequence of amino acids in a protein mirrors the sequence of bases in a particular piece of DNA, the **gene** which **codes for** that protein. DNA actually comes in large molecules called **chromosomes**, each of which contains many genes. **Gene expression** is the cellular process by which

[*] Based on material prepared by Dr. Timothy Danford

genes are decoded into their proteins. The process of expression has two distinct steps. First, genes are **transcribed** into an intermediate form (**mRNA**[1]), the carbon copying sheet to DNA's plain paper). Second, the intermediate mRNA is **translated** into the corresponding protein.

Ideally, a biologist interested in measuring the structure and function of a cell (or population of cells) would measure the varying amounts of different proteins present in those cells at different points of the cell's lifecycle, or when the cells were exposed to different stimuli or stresses. However, for technical reasons, direct measurements of proteins is difficult — instead, biologists resort to measuring the process of gene expression as a proxy. In particular, if a large amount of the intermediate mRNA form of a gene is present in the cells then this is a clue that (perhaps) the corresponding protein is abundant as well.

Because biologists are so interested in measuring the expression of genes (again, as a proxy for proteins), they have developed experimental techniques by which the expression of *all* genes in a genome can be measured simultaneously[2]. For each gene in the genome, the biologist measures the "intensity" of that gene's expression — a real number whose value is related to the abundance of the gene's transcript in a population of cells. A collection of expression experiments forms a data matrix: the rows are *genes*, with one row for each gene in an organism, and the columns are the *experiments*.

1.2 Co-expression and Gene Modules

The first thing biologists want to know is which genes have similar patterns of expression across experiments — which genes are **co-expressed**. This usually indicates deeper similarities between the genes or their proteins. For instance they might be on the same **pathway**, a set of proteins which interact in sequential or regular ways, in order to communicate a message or perform a function within the cell[3]. Since pathways function as a unit, it doesn't no good to make only one of the proteins in the pathway — all of them are needed. Thus we expect all the genes coding for proteins in a common pathway to be co-expressed.

Groups of co-expressed genes are sometimes referred to as **modules** in the biology and bioinformatics literature. A module is a collection of genes which share a common pattern of expression in a common set of experimental conditions. Notice that the genes in a module do not have to be co-expressed in *all* experimental conditions — the function of the module may only be active in some experiments or some kinds of cells. The genes in a module coding for a pathway which responds to arsenic, for instance, should be co-expressed in

[1] RNA is a class of molecules very similar to, but slightly different than, DNA. The "m" stands for "messenger" and indicates the functional role of these RNA molecules.

[2] Exactly *how* these experiments are performed isn't relevant to this exercise; if you're interested, try searching for the phrases "gene expression microarrays" or "transcript sequencing" in your favorite search engine.

[3] An example of such a pathway is a **metabolic pathway**, where a sequence of chemical reactions to breakdown a chemical and provide energy for the cell are catalyzed by a set of proteins, or conversely in which some useful chemical is built up from other molecules at a cost of energy.

experiments where we expose the cells to arsenic, but not necessarily in other experiments.

1.3 Transcriptional Regulation

But why should we expect to see genes co-expressed at all? How does the cell actually effect a common expression pattern for a set of functionally related genes? The answer lies in the process of **transcriptional regulation**.

Transcription, the key first step in gene expression, doesn't just happen randomly or haphazardly. Each gene is "turned on" by a complex transcriptional machinery which interacts with signals in the DNA sequence of the gene itself. Genes which share a common function (and are part of the same module) will have the same, or similar, versions of these DNA signals — and so they will be affected by the transcriptional machinery in the same way, and be expressed in the same experiments.

The transcriptional machinery is itself composed of proteins, which are themselves coded for by genes and must be expressed. Genes coding for components of transcription mechanisms are called **transcription factors**, and are often co-expressed with the genes whose transcript they regulate. Fr example, if gene A is a transcription factor which activates Gene B, then when Gene A is expressed we should see a corresponding increase in the expression of Gene B.[4] There can be more complicated set-ups as well, like a gene C whose expression requires *both* genes A and B, but not either in isolation. In that case, if gene A is pretty much always (or "constitutively") expressed, we'd tend to find gene C is more co-expressed with gene B than with gene A.

A tighter classification of genes beyond co-expression is **co-regulation** — a co-regulated set of genes is co-expressed because they share a common set of regulatory DNA signals to attract a common set of transcription factors. A module is co-regulated if all the genes in the module share a common set of regulatory transcription factors which together control the module. The discovery of co-regulated modules, and the transcription factors which regulate them, is a fundamental goal of the field of functional genomics.

Co-expression is about an association between the expression levels of genes; it's just about probability distributions. Co-regulation on the other hand is a *causal* idea — it says there are shared causes for the expression levels of the co-regulated genes. Co-regulation is usually investigated from experimental rather than just observational data. That is, biologists experimentally *change* the expression of selected transcription factors, and see which genes' expression levels change in response. The crude but reliable way of doing this is to engineer **knock-out strains** of organisms, which are simply missing the gene for a given transcription factor. Because the gene is completely missing, it can't be

[4] One way this can work: gene A codes for a protein which attaches itself to the chromosome near gene B, and which preferentially binds to the proteins which actually transcribe DNA into mRNA. Thus, expressing gene A increases the rate at which gene B gets expressed. If by contrast gene A's protein likes to attach itself *directly* to gene B on the chromosome, this tends to interfere with transcribing gene B.

expressed, and (assuming the organism lives to express any genes at all!) biologists can them compare the expression levels of other genes in the knock-outs to the expression of those same genes in normal control organisms.

1.4 Data and Objectives

In this exam, we will walk you through a few of the basic tasks involved in identifying gene modules and their common regulators. As you will see, much of what you have learned applies immediately to discovering functional modules.

All the data comes from expression measurements in **yeast**, which are well-studied organisms both because they are economically important[5] and because they are easy to experiment on (grow rapidly in vast quantities, easy to manipulate, etc.). The standard strain of yeast used for these experiments descends from one use by brewers. Its genome is well-studied, containing about 6000 genes with a standardized naming scheme[6]. The set of transcription factor genes has been known for at least ten years, and there are a wealth of whole-genome datasets available from publicly-funded research. We will work with two of them, which you can find on Blackboard.

Both data sets are tab-delimited text files, `expressdb_cleaned.txt` and `HuIyer_TFKO_expression.txt`, which you can read in as follows:

```
edb <- read.table("expressdb_cleaned.txt",header=TRUE,row.names=1,sep="\t")
```

This tells R where to find the file, that the first line of the file contains column names, that the first column contains row names, and that entries are separated by tabs. In each file, the columns (features) correspond to experiments, and the rows to genes. The actual entries in the data frame show how much of each gene was expressed in each experiment.

`expressdb_cleaned.txt` compiles data from 439 observational expression experiments done by different labs. (The complete dataset is called "ExpressDB," and was independently curated and redistributed by a lab at Harvard University.) Each of the 5468 rows corresponds to a different gene (with its standardized name), and each column to an experiment (with a name and, here, a number). These experiments were all done on yeast which are, for our purposes, genetically unmanipulated, so differences in expression levels between experiments are due to differences in the experimental conditions that the cells were grown under (nutrients, chemical or physical stresses), or noise. The data set is *not* normalized, so the numbers in different columns are not (necessarily)

[5] Beer! Beer! Beer! Beer!

[6] In this scheme, the names have the format `Yx{RL}###{CW}` . The `x` is a single letter that indicates the chromosome of the gene (chromosome 1 is 'A', chromosome 2 is 'B', etc.), followed by a letter (either `R` or `L`) which indicates which *side* of the chromosome the gene is on. The next three digits indicate the *order* of the gene on that side of the chromosome, and the final letter (either `C` or `W`) indicates the *direction* that the gene is pointing ("C" stands for Crick and "W" for Watson, the scientists credited with the discovery of the structure of DNA). `YAL010C` is the 10^{th} gene in on the left arm of the first chromosome, which points in the "Crick" direction.

directly comparable — a value of 1.0 in one experiment may be equivalent to a value of 4.5 in another.

The second data set comes from a 2007 paper by Zhanzhi Hu and his collaborators, and is a "knockout" expression atlas for all of the 269 transcription factors in the yeast genome. That means that, for each transcription factor, they engineered a strain of yeast which knocked out *just* that gene, and then measured the resulting expression of all the remaining genes (6429 of them)[7]. Here the values are *relative* expression levels — the logarithm of the ratio of the expression in the knockout strain to that in an unmodified strain measured under the same conditions. (So negative values mean *lower* expression in the knockouts than in the controls, and positive values mean higher expression levels[8].) As with the first experiments, however, you cannot assume that the values are *normalized* or comparable across experiments.

Both data sets have missing values (marked `NA`). That is, some genes' expression levels were not recorded for some experiments. Missing values are a fact of life in expression analysis (and real data generally). It is *not* the case that the gene wasn't expressed, just that the gene's expression wasn't measured correctly. Your code needs to be able to cope with this.

If you want to learn more about proteins, genes, and gene regulation, an excellent place to start is Gonick and Wheelis (1991). Baldi and Brunak (2001) has a *lot* of material about applying data-mining techniques to problems in bioinformatics. There is an especially large literature on discovering modules; Bar-Joseph *et al.* (2003) was an early entry, and one of the data sets comes (ultimately) from that paper.

2 Problems

1. *Calculating Gene Expression Similarity* The first task is to develop a measure of **gene expression similarity**. A gene expression similarity function is a method which takes three inputs: two genes, and a set of experiments (possibly the complete set of experiments). It returns a numerical indicating the similarity or difference in expression between the two genes *across the indicated experiments only.*

   One of the simplest tasks you can use a gene expression similarity function for is to rank genes by their similarity with respect to a reference gene, from most similar to least similar. Even before the identification of modules and

[7] Every gene in the first data set also has a row in the second. The second data set also has rows for some genes omitted from the first for various reasons (e.g., too many missing values). You can ignore the extra rows if you want.

[8] Higher expression levels are perfectly reasonable, since some transcription factors regulate their target genes by *repressing* them — like putting a light on a dimmer. If gene A represses gene B, and you remove A, you would expect, all else being equal, to see the expression levels of B go up. One way things could not be equal was if gene C is an even stronger repressor of B than A is, but A also repressed C. Plainly, this sort of thing gets very complicated very fast.

regulators, biologists will often wish to ask the question "what genes are most similar to Gene X?"

(a) Write an R function for the calculation of a gene similarity function. If you need to do any pre-processing of the data beforehand, describe it, and include the code for it.

(b) Discuss what normalization, if any, was necessary in order to create your similarity function.

(c) What other properties does your similarity function possess? Is it symmetric (i.e. does it produce the same result when you switch the gene arguments)? What value does it return if you pass it two identical rows (genes) as arguments?

(d) Use your similarity function to produce a ranking of genes by their similarity to gene `YPR035W`.

(e) Describe a method for determining a ranking *cutoff* — that is, a level of similarity so low that we can reasonably discount the idea that two such dissimilar genes are co-expressed.

(f) Extra credit: Implement the method you just described. Which genes have above-threshold similarity to `YPR035W`?

2. *Discovering Co-expressed Modules* For the next task, you will use your gene similarity function to discover co-expressed sets of genes in the ExpressDB dataset. As discussed above, a co-expressed module is a collection of genes whose expression is similar across a collection of experiments. In our case, we will ask you to focus (again) on `YPR035W` — we are interested in the module or modules which contain this gene in different subsets of experiments.

   (a) Write an R function which takes a gene and a set of experiments as input, and produces the module (set of genes) as output which are "co-expressed" with the given gene in the indicated experiments. Note that you do not *have* to use the similarity measure from the previous problem, but you can.

   (b) What properties does your module discovery algorithm have? If you ran it on gene A (producing as output module A), and then chose a gene B not in module A as second input — would the output of the second run, module B, overlap with module A? That is, are the modules you produce **disjoint**?

   (c) Use the function to find the module containing `YPR035W` in the group of experiments numbered 23–30, 61–65, 103–112 and 287–291 (this is a single group of related experiments, not four separate ones). Repeat this using experiments 359–388. Are the genes within the two modules the same? Overlapping? Completely different (aside from `YPR035W` itself)?

(d) Discuss the design of a "module discovery" method — this is a method which takes only a set of experiments as input, and produces *all* the relevant sets of genes which form modules with respect to these experiments.

(e) Extra credit: Implement and run your module discovery method. Include the results.

3. *Discovering Co-regulated Modules*

   (a) Write an R function to discover the regulators of a module — this is a method which takes as input a module (a set of genes), and produces a list of all regulators (these can be experiment names from the Hu/Iyer dataset) which are regulators of the genes in that module.

   (b) Discuss the characteristics of this function — are any of the regulators discovered for the module part of the module itself? (Note that *self*-regulation, where a transcription factor regulates both a set of target genes and itself, is a well-known biological phenomenon.)

   (c) Discover the regulators of one of the modules that you discovered for gene `YPR035W` in problem 2.

   (d) Given that regulators can regulate each other as well as other genes, describe a method by which you might try to discover the **regulatory network** of the cell — this is the complete set of regulatory interactions between transcription factor genes and their targets.

   (e) Extra credit: Implement and run your regulatory network discovery method.

References

Baldi, Pierre and Søren Brunak (2001). *Bioinformatics: The Machine Learning Approach*. Cambridge, Massachusetts: MIT Press, 2nd edn.

Bar-Joseph, Ziv, Georg K Gerber, Tong Ihn Lee, Nicola J. Rinaldi, Jane Y. Yoo, François Robert, D. Benjamin Gordon, Ernest Fraenkel, Tommi S. Jaakkola, Richard A. Young and David K. Gifford (2003). "Computational discovery of gene modules and regulatory networks." *Nature Biotechnology*, **21**: 1337–1342. doi:10.1038/nbt890.

Gonick, Larry and Mark Wheelis (1991). *The Cartoon Guide to Genetics*. New York: HarperCollins, 2nd edn.