The Jackson Laboratory

A Bioinformatics Approach to Develop an Application for an Interactive Gene Co-Expression Network for Breast Cancer

Catherine Ding

The Jackson Laboratory

Academic Year Internship

June 6, 2018

Catherine Ding Catherine Ding

Student

Krishna Karuturi ____________________________________

Mentor

Abstract

A valuable tool that may be used to aid many biologists are gene co-expression networks (GCN). These networks can be used for differential co-expression analysis, where the differences in co-expression profiles of various sets of genes viewed in multiple contexts. In the case of various diseases, these analyses may aid in developing potential treatments as they can be used to infer potential diseases genes, which can then later be targeted. With the coding language R, both the Shiny and igraph packages will be used to build an interactive web application that will allow for network analysis based upon user input of total number of co-expression relationships, gene name, and context (estrogen receptors, p53, and grade). We were able to carry out original goals, however with a few issues such as efficiency in loading time.

Introduction

The tumor suppressor gene TP53, which encodes the protein p53, regulates biological processes in order to maintain genome stability. In 80% of all cancer cases, a mutation in this gene occurs. While the frequency of this mutation occurring in breast cancer is 20%, they are most common in the most aggressive breast cancer subgroup: triple negative breast cancer. Estrogen receptors serve as a useful prognostic marker to predict the course of breast cancer or any treatment targeting the disease.

Biological systems respond to either internal or external stimuli, which control regulation or expression pathways in genes. In one of the earliest works performed by Butte and Kohane (1999), they proposed that highly mutual information between two genes suggests that they are non-randomly associated and are thus biologically related.

The idea of gene co-expression networks was also first introduced by the work of Butte and Kohane as “relevance networks,” where they took medical laboratory test data from a number of patients and then calculated the Pearson correlation between results from different pairs of tests.

Gene co-expression networks today are often constructed from datasets obtained from microarrays or RNA sequencing (RNA-seq). In these networks, there are nodes or vertices that correspond to a specific gene, and edges that represent a certain correlation or relationship in co-expression profiles. Though edges are undirected and thus do not prove causality, they are valuable in elucidating whether clusters of genes with similar co-expression patterns are controlled by the same transcriptional regulatory program, functionally related, or members of the same pathway or protein complex. Using the guilt-by-association (GBA) approach, potential disease genes may be identified if they are that are co-expressed with other disease genes. By identifying the genes that play an important role in breast cancer, an effective treatment may be more efficiently developed.

Methods

R packages Shiny and igraph will be used to build an interactive web application of a gene co-expression network. Using differentially co-expressed gene sets, the network will contain a node for each unique gene and distinct edge colors that correspond to positive or negative coefficients of estrogen receptors, the p53 protein, and tumor grade.

Debugging the current code was first priority to make sure everything worked properly before I added new features. Next, in the server and user interface scripts, I will code for a slider that takes an integer as user input. This number will determine the genes displayed at one time based upon whether the total number of relationships one gene has with other genes is above or below this integer. Another feature will be a search bar for a gene name that will display all the relationships of that one gene. Lastly I will create tick boxes for each one of the three contexts (ER, p53, grade).

To aid in this process of debugging and the addition of new features, it would be wise to make a smaller data set to test code, due to the data input being very large (184 genes, 59 different sets, and over five thousand different connections). Taking the first two lines from the original data, it is much easier to trace the original code to debug and also see immediate effects of any changes to the code when adding new code.

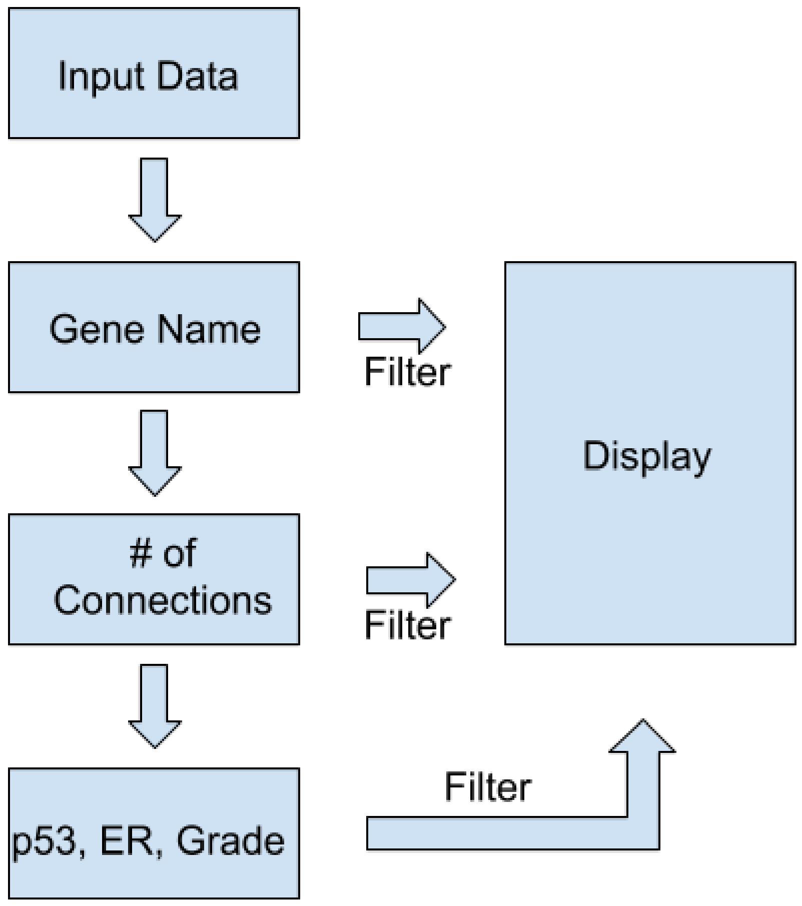
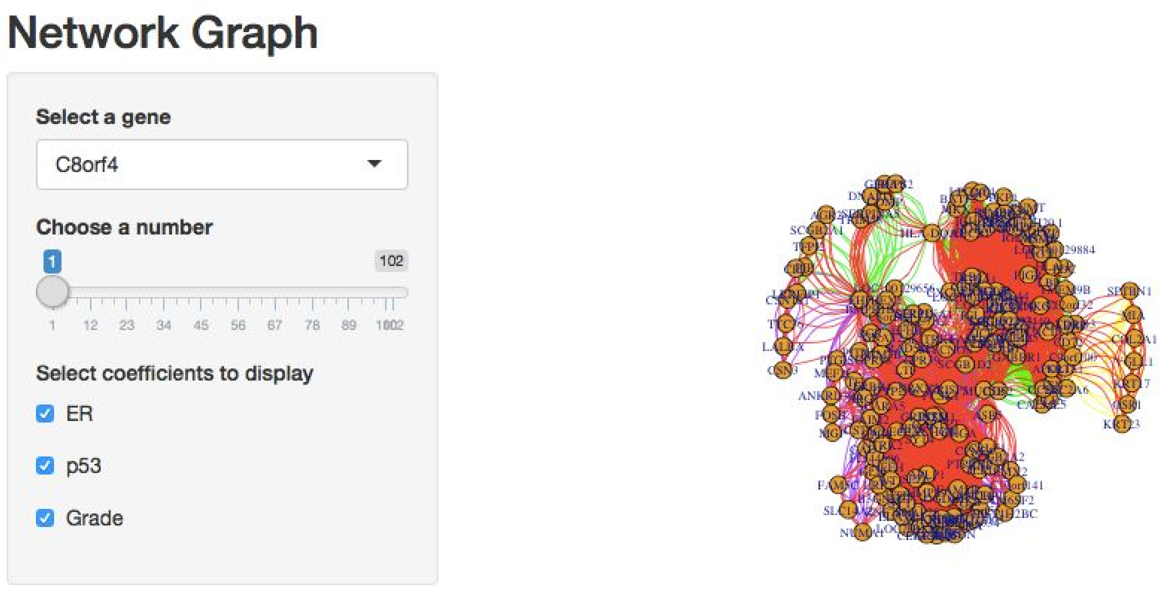


Figure 1. The theoretical flow of how data will be processed. After input data is present, and the application is running, one must first manipulate the display by selecting a specific gene. This filters all the data to only show nodes that have connections to this one gene. Next, using the number slider, the user will be able to further narrow down the already smaller data set used to display the network by selecting a minimum for the number of total connections each node has. For example, if the gene "SLC7A2" has 17 total connections and the slider is set to 20, it will be filtered from the current data and removed from the network display. Lastly, the user will have the option to select one or a combination of the three contexts to further narrow down and filter the data.

Results

Current example outputs:



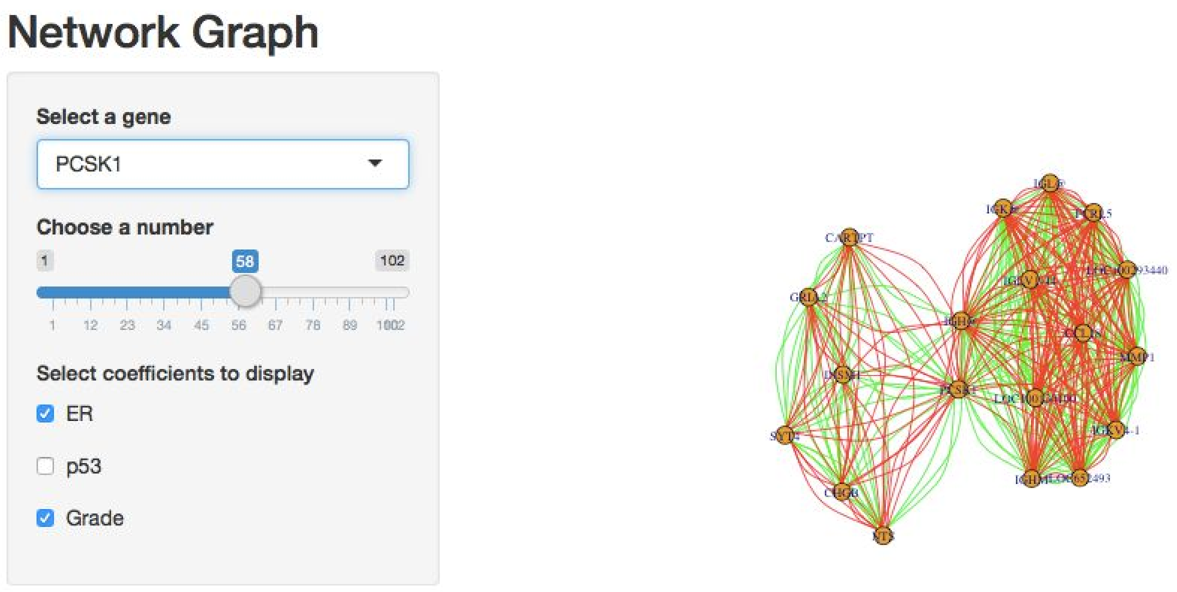
Network Graph

Select a gene: C8orf4

Choose a number: 1 (range 1–102)

Select coefficients to display

- [x] ER
- [x] p53
- [x] Grade



Network Graph

Select a gene: PCSK1

Choose a number: 58 (range 1–102)

Select coefficients to display

- [x] ER
- [ ] p53
- [x] Grade

[Code is attached (server.R, ui.R, Plot.R, Variables.R).]

The time needed to display the new network after changing any of the widget values is significantly long. This is mainly due to the inefficient method for the filtering process using the slider input. While there were no apparent effects on load time in my “mini-application” with the smaller data, after copying the code to the actual data, the effects were much clearer as each change in slider input or any other widget, would warrant a full 1-2 minutes of waiting time. To improve the efficiency, I cleaned up the edge list to remove any repeat connections (originally the edge list contained over 12 thousand, rather than the current 5). However even then, the load time was still significantly slow, which inhibits practical use of the application. This suggests that the problem is within the code for the slider, more specifically, it is a result of the overabundance of nested for loops, which requires the computer to cycle through every one of the over five thousand values many times in order to eliminate certain values. To eliminate this problem, one must find use of other R functions to subset data, perhaps the %in% function, to use in place of the nested for loops.

Conclusion

I’ve learned through this experience that computer science projects or projects in general do not always have an official end point. There is always room to reflect, redesign, and improve what you already have done. Despite practically finishing what was originally planned for this project, such as the addition of the new features such as a search bar and tick boxes, as well as debugging, I came to the realization that one must always keep the critical mind open for these new enhancements. Even when a task was finished, such as the filtering of data to display the slider input, we realized there were obvious improvements to be made. that my method for the filtering process was inefficient due to the overabundance of nested for loops. As a result of this use of for loops, the load time for the display was significantly negatively impacted.

Further features and improvements that may be added to this code in the future would be to fix the inefficiency in using for loops to process the slider input, the addition of weighted edges, in which the thickness of an edge corresponds to however many occurances there are of a particular connection between two genes or nodes, and a zooming feature that allows for the user to zoom in or out to look more closely at the co-expression network and node labels. Furthermore, to make the code more neat and efficient, it may be valuable to break the original code into functions then call the functions from the server or user interface, instead of having one large chunk of continuous code.

References

1. Herty Liany, Jagath C. Rajapakse, and R. Krishna Murthy Karuturi. "MultiDCoX: Multi-factor analysis of differential co-expression." *BMC Bioinformatics*. The Author(s). 2017, published 28 December 2017.
2. Sipko van Dam, Urmo Võsa, Adriaan van der Graaf, Lude Franke, João Pedro de Magalhães. "Gene co-expression analysis for functional classification and gene–disease predictions." *Oxford Academic: Briefing in Bioinformatics.* N.p. Published 10 January 2017.
3. A. J. Butte, and I. S. Kohane. "Mutual Information Relevance Networks: Functional Genomic Clustering Using Pairwise Entropy Measurements." *Pacific Symposium on Biocomputing*., U.S. National Library of Medicine, published 2000.

4. C. Berger, Y. Qian, and X. Chen. “The p53-Estrogen Receptor Loop in Cancer.” *Current Molecular Medicine*, U.S. National Library of Medicine, published Sept. 2013.
5. Sommer, S, and S A Fuqua. “Estrogen Receptor and Breast Cancer.” *Seminars in Cancer Biology.*, U.S. National Library of Medicine, published Oct. 2001.