Introduction to Scientific Computing for Biologists: Final Project

Julie Szymaszek

21 March 2015

1 Introduction to the Problem

In the age of cheaply available sequencing, there is a wealth of data available through databases like NCBI's SRA. This is useful for exploratory projects, like rotations, that want to look at data without spending too much money on generating data. For my rotation last quarter, I was interested in parsing through publicly available RNAseq data in NCBI's DRA database in order to test hypotheses about the evolution of cell types.

However, all of these RNAseq data files have a lot of information. Simply put, the problem is that a lot of this information was not useful to the project I was working on. I obtained a set of adipose tissue RNAseq files that were processed using an open, web-based platform called Galaxy. The formatted RNAseq files were of varying kinds of adipose tissue from a variety of animals, though mostly from mammals.One of the first things I wanted to do with the files was find out what genes were "expressed" and compile a list of all of those in a single file for downstream applications.

The easiest way to do this would be to use the column in the file called "TPM" as a measure for expression of the transcripts reported. As a general rule for my project, I was using any TPM value of 2 or higher as a criteria for expression. I wanted to pull out every transcript that had a TPM value of 2 or higher. From there, I needed to know what the associated gene name is for that transcript ID, because the only things in that column are the names for transcripts presented in how the Ensembl database reports it. After getting the gene name, I'd like to know what genes are commonly expressed in all of these tissues.

Doing all of this by hand would take a long time because I'd have to first sort through one file by TPM number and only pull out transcripts. Then I'd have to look up the transcript names in order to figure out what the associated gene name for each one is. After I did that for every single, I'd have to use a venn diagram calculator to find out what genes all of these samples had in common. The quarter is only 10 weeks long, and this was only the first step in a series of things I was supposed to accomplish. Once I know what genes I'm interested in - I can look up whether they are of interest to me, and start planning experiments with cell cultures in the lab.

2 One Possible Solution

Since I have a file that contains every single possible transcript and their associated names from every single species I am working with, I want to use this as a dictionary to lookup the names of genes that are expressed, and use that output to find all of the genes in common.

I wrote two separate scripts which are both fold in the main directory called Final Project. The first script entitled main.py extracts ENSMBL Transcript ID for TPM values greater than 2 from all of the aligned RNAseq files present in the specified subdirectory called transcript reads. After extracting all of the transcript read file, the script identifies the "locus" in the files, which is the transcript ID. Those entries with a TPM value greater than 2 (an arbitrarily selected cutoff value that would mean this transcript is "expressed" in the tissue) are exported into a list. Should I decide that there is reason to use another TPM cutoff value, I can update the script (for example, Kin et al, 2015 used a TPM cutoff of 3 [1]). The locus is compared to the giant file I have of every possible transcript and its gene name, and pulls out the gene name. The gene name output is saved into individual files for each tissue sample as "output original file name.txt." These are all saved into the subdirectory entitled "associated genes."

The second script I wrote called associated genes.py can then be called from the command line. This script takes all of the files in the subdirectory associated genes and finds the intersection of its contents in sequence. It saves its output into the main folder in a file called intersection genes.txt.

3 Code

Before I can access the code, I have to clone my directory from github, using the following command in terminal:

```
git clone https://github.com/jszymaszek/scicom-project.git
```

This directory contains the two necessary scripts, and the subdirectory that contains all of the transcript read data files that will be necessary to complete the analyses and run the two scripts successfully.

The first script, used to lookup the gene names of all of the transcripts that are expressed in the different samples, is presented below:

```
# !/usr/bin/python

import os
#this imports a module that lets me read all of the files within a specific directory later in this code

def build_dict():
    dict = {}
    with open("associated_gene_names.txt") as file:
  #build a dictionary using the associated_gene_names.txt file as its input
        for line in file:
            arr = line.split("\t")
            if len(arr) > 1:
                locus = arr[0].lower()
                associated_gene = arr[1].lower()
                dict[locus] = associated_gene
    return dict
#whatever values i'm going to want to call from this dictionary will look up in the locus column
#then it will just spit out whatever value is associated with that locus column entry from the associated_gene
```

```
    ↩ column
19
20 for root, dirs, files in ↪
    ↩ os.walk("./transcript_reads",topdown=True):
21 #reading everything inside the folder transcript_reads
22     dict = build_dict()
23 #creates a dictionary that is comprised of the build_dict() ↪
    ↩ from above
24     with open("garbage.txt","w") as garbagefile:
25         garbagefile.truncate()
26 #creates a file called garbage.txt where some errors will go
27 #truncate clears out the file every time it's running so ↪
    ↩ that way there aren't duplicates
28     for name in files:
29         filename = os.path.join(root, name)
30         with open(filename) as file:
31          print "------"+filename+"-------"
32          lines = file.readlines()
33          column_names = lines[0].strip().split("\t")
34          locus_index = column_names.index("locus")
35          TPM_index = column_names.index("TPM")
36          outfilename = ↪
                ↩ os.path.join("./associated_genes","output_"+name)
37 #within each file that is opened it takes the locus and the ↪
    ↩ TPM columns from the files within transcript_reads
38 #outfilename section is specifying that the output files ↪
    ↩ from this script will go here with each file being ↪
    ↩ called
39 #output_name of the original file.txt
40         with open(outfilename,"w") as outfile:
41             for i,line in enumerate(lines):
42              if i !=0 and len(lines[i].strip().split("\t")) ↪
                    ↩ >= max(locus_index,TPM_index):
43 #ignore line 0 (column names), makes sure that the index ↪
    ↩ being accessed is within the range of the line
44                 locus = lines[i].strip() ↪
                        ↩ .split("\t")[locus_index].lower()
45                 TPM = float(lines[i].strip() ↪
                        ↩ .split("\t")[TPM_index])
```

```
#parse the locus and the TPM columns
                if locus in dict:
                    if TPM > 2 and dict[locus].strip() != "":
                        associated_gene = dict[locus]
                        outfile.write(associated_gene)
#look at all of the corresponding locus entries and pull out the locus name is the TPM value is greater than 2
#then use that locus name to lookup the associated gene name in the dictionary function written above
#print out the gene name of that locus with a value into the file output_name of the file.txt
                else:
                    with open("garbage.txt","a") as garbagefile:
                        garbagefile.write("~~filename:")
                        garbagefile.write(filename)
                        garbagefile.write(" ~~locus:")
                        garbagefile.write(locus)
                        garbagefile.write("\n")
#however, sometimes the locus in the single files might not have a corresponding gene name in the dictionary
#if this is the case, print them out into this file called garbage.txt
#this file also includes the name of the file and the locus name so i can later see what "failed"


    '''print "---dirs---"
    for name in dirs:
      print root
      print name
        print(os.path.join(root, name))'''
print "END"
```

Now, it's time to use the second script to find what genes all of these samples have in common:

```
#!/usr/bin/python -tt
# -*- coding: utf-8 -*-

import os
#imports operating system module to be able to read the list of files within a directory

def build_list(filename):
    list = []
    file = open(filename,'r')
    for line in file:
        line = line.strip()
        list.append(line)
    file.close()
    return list

#this builds a function called build_list that takes the input from the directory associated_genes
#it builds a set with them

def get_intersect(listoffilenames):
   flag = 1
   for filename in listoffilenames:
      print filename
      if flag == 1:
         s = set(build_list(filename))
         flag = 0
      s &= set(build_list(filename))
      print '\n'.join(s)
   return s

#this builds a function called get_intersect of all of the filenames set
#starts with a file and it takes the intersection of it with the next file and so forth
#basically takes the intersection of the files in sequence, and joins them together

```

```
def build_output(listoffilenames,outputfilename):
    file = open(outputfilename,'w')
    intersect = get_intersect(listoffilenames)
    for i in intersect:
        file.write('%s\n' % i)
    file.close
#taking the output of the get_intersect function and writes it to file

if __name__ == "__main__":
   listoffilenames = []
   for root, dirs, files in os.walk("./associated_genes",topdown=True):
      for name in files:
         filename = os.path.join(root, name)
         listoffilenames.append(filename)
   build_output(listoffilenames,'intersection_genes.txt')
#this uses the main function and os to take all of the files in the folder "associated_genes"
#and uses the list of filenames as the input for this script
#then it builds the output of this script into a file called "intersection of expressed genes
```

4 Future improvements

As I worked through this problem, I thought of a couple of ways in which this process could be streamlined further. For example, instead of executing two scripts, I could "join" them together using the main function in order to use execute the second script as a module of the first one. In the future, I may want to run both simultaneously, so this would be useful to do.

Another possibility area for improvement would be to use a package like Mechanize in order to obtain all of the transcript/associated gene name information I have contained in my dictionary file. I already had made this file a while back, so I decided to use it here, but all of the data comes from Ensembl's BioMart website. I could use a package like Mechanize to write a script that would access the information I needed, so that I wouldn't be limited to just the species I have in my dictionary.

References

[1] Koryu Kin, Mauris C Nnamani, Vincent J Lynch, Elias Michaelides, and Gunter P Wagner. Cell-type phylogenetics and the origin of endometrial stromal cells. *Cell reports*, 10(8):1398–1409, 2015.