CB2-101: Final Problem Set*

Feb 28, 2014

Contents

1 How to submit you answers **1**

2 Problem 1 **2**

3 Problem 2 **3**

4 Problem 3 **3**

5 Problem 4 **3**

1 How to submit you answers

1. First create a directory with your name, then create separate subdirectory for each problem. An `ls()` inside the first directory should return something like this:

```
problem1/
problem2/
problem3/
data/
...
```

2. The data files required for the problems should reside in the `data` directory. For e.g., the domain file, `"9660.tsv.gz"` will be inside the data directory. Do not uncompress any files. Keep the files as you have downloaded from the Internet.

3. In each `problem` directory put your scripts. You *should not* put any data files in there. Write only scripts that will transform each data file and create intermediate files if needed. But do not submit intermediate files with your answer.

4. Assume that your input data file will be always in the data directory. That means, if a program asked for input of `"9660.tsv.gz"` and you are inside problem directory you should pass the argument as `../data/9660.tsv.gz`.

*If you can solve all of the problems in this problem set, you're ready for research in bioinformatics. Contact us. There is a job waiting for you.

5. In each problem directory, in addition to your other scripts, you will have a special `bash` script called, `run.sh`. There should be only one `run.sh` in each problem directory. The script will not take any argument. If I run the script as `sh run.sh` it should perform the analysis and create the answer. Put all calling argument in this file. For e.g., if your program name is "problem1.py" and you normally run it like this:

```
        problem1.py ../data/9660.tsv.gz BRCA1

Your `run.sh` should look like this:

        #!/bin/bash
        problem1.py ../9660.tsv.gz BRCA1

Simply running the `run.sh` should give me the answer to the problem 1.
```

   NOTE: If you program does not take any argument, you may not write a `run.sh`.

6. You need not finish all the problems to submit your answer. But make sure that you submit only the directories that you have completed. If you have completed only the problem 1, your directory structure should look like this:

```
        problem1/
        data/
```

7. Now create a tarball of the whole directory structure. So my directory tree looks like this,

```
        malay/
            problem1/
            problem2/
            problem3/
            problem4/
            data/

Go to one more directory up. So my `ls()` look like this:

    malay/

Now create a tarball like this:

    tar -cvzf malay.tar.gz malay/
```

8. Send the tar ball to malay@uab.edu or emidio@uab.edu

2 Problem 1

The PFAM domain distribution for human proteome can be found at ftp://ftp.sanger.ac.uk/pub/databases/Pfam/releases/Pfam27.0/proteomes/9606.tsv.gz. The first column of this file is the protein accession number. The location of the domain hit for each gene is given by the columns 2-5. Columns 2-3 are alignment start and end. Columns 4-5 are envelope start and end. Envelopes are generally considered the location of a domain on a gene. Write a python program that takes 9606.tsv.gz file as a first argument, a protein accession number as a second argument, and a location (integer) as a third argument. The program should print the domain name (hmm_name), if the location falls within a domain for a given protein accession. The program should return nothing if the position is outside the boundaries of domains. We should be able to run the program like this

```
> problem1.py ../data/9606.tsv.gz O95931 20
> Chromo
```

Note: Remember to populate your `run.sh` file in the directory with the command example given above.

Hint: You should create a dictionary using the protein accession as key and location start and end as values.

3 Problem 2

Swissvar is a database of human gene, their variations, and disease associations. The file can be downloaded from here: ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/variants/humsavar.txt. The 2nd column of the file is the protein accession numbers. These are the same accession numbers used in the domain file in Problem 1. The 6th column is `dbSNP` and reports the variation at a particular location. Using these two files, create a sorted list of domains according to the total number of their variations. The domains with higher variations should be on top. The program should not take any argument (files should be read from the data directory) and output the domain list on STDOUT. The output should have two columns, separated by tab: domain name (hmm_name) and a number indicating variation, like this

```
Domain  Variation
BRAC1   150
Chromo  100
...
```

Remember, your output will differ from the above shown output. The first line is the header. **Note**: You may skip writing a `run.sh` file for this problem.

Hint: The location of the variation can be extracted from the `"AA change"` column of SwissVar data. For e.g., p.His52Arg means the variation is at the location 52.

4 Problem 3

Write a program that will generate the abundance of domain in human genome. Use the 9606.tsv.gz file and count the number of times a domain present in a genome. The program should generate the output in descending order, the higher abundant domain should be on top. An example output will be like this:

```
Domain  Abundance
AAAtpase    250
SH3         100
...
```

Note: Your output will vary. The program should not take any argument. You need not write a `run.sh` script for this.

5 Problem 4

From the Swissvar file in Problem 2, we found the number of variations present in each domain. But this may be due to an artifact of domain abundance in human genome. Highly abundant domains will have higher chance of accumulating variations. We will test this hypothesis using a correlation between the abundance of domain and the accumulated variation. We calculated the abundance of domain in problem 3.

First run the scripts in the problems 2 and 3 and save their outputs in files. The output should remain in their original locations. **Caution**: The rows in the files are different. You many need to write a separate python/R script to merge the columns of the file.

Use a Rscript to read the files created in problem 3 and 4 (or, a merged file). Draw a linear regression plot between the abundance in X-axis and number of variation in Y-axis. The script should also report the correlation between these two variables.

Hint:

1. Read the tables from within your script. Read the files from their original location, like this:

   ```
   abundance <- read.table("../problem3/abundance.txt",header=T)
   ```

2. If you have used an intermediate script to merge the two files, then you should modify your RScript accordingly.

3. Don't forget to create a `run.sh` for this problem.

4. You can output the plot in a PDF file if you want. The correlation should be done using the R function `cor.test`. The test should be `two.sided` and method can be anything.