BIMM 185 Lab Report 2

Lihui Lu A99108553 4/18/2017

Exercise 1 - Navigating directories

The goal of exercise 1 is to practice how to access multiple files with similar storage structures, process data to obtain useful information for later use using both python and bash.

Exercise description:

- Copy the folder dirtree to your account (located in ../public/data/terminal/dirtree).
- Inside dirtree there are many directories, inside each directory there is another directory with the same name (e.g., 1.A.14_vs_1.A.26/ 1.A.14_vs_1.A.26/), and inside the second directory there is a file named report.tbl
- To verify this information, inside dirtree try the command: ls //*.tbl
- The results in report.tbl are sorted in descending order.

- For each directory in dirtree extract the top GSAT Z-score (row 3, column 4) from file report.tbl and generate an output file with to columns (base dir and Z-score) sorted in descending order by the Z-score:

```
2.A.123_vs_3.E.1        67
2.A.58_vs_9.A.14        19
1.A.76_vs_9.A.14        18
2.A.102_vs_2.A.112      18
....
```

This can be done using both bash and python. Using bash, we can access each report.tbl file using a for loop; get the third line using `head` and `tail`; get the fourth column using `cut`; save all the data in a separate file using redirection `>>` and finally sort using `sort`.

Bash script

```
for dir in *;do (echo -ne "$dir\t" >> test.txt;head -3 $dir/$dir/*.tbl | tail -
```

```
1 | cut -f 4 >> test.txt);done; sort -k2nr -k1 test.txt

#Explanation of the bash script:
for dir in *;do (echo -ne "$dir\t" >> test.txt;head -3 $dir/$dir/*.tbl | tail -
1 | cut -f 4 >> test.txt);done;
#For loop to save the sample name and the corresponding top GSAT Z-score in tes
t.txt, separated by tab
sort -k2nr -k1 test.txt
#sort test.txt first by the second column numerically in descending order(-k2nr
), then by the first column alphabetically(-k1)
```

Using python, we can access each file using function `os.listdir()`, read the top GSAT Z-score and save the information(sample, zscore) as a tuple into a list. Then after all the files have been read, and sort the list with the second item of the tuple in descending order.

Python script

```
import os

mylist = []
#go through all the files and directories in the current directory
for file in os.listdir("."):
    #if it is a directory, we go in and get report.tbl
    if os.path.isdir(file):
       #path to report.tbl
        report = os.path.join(file+"/"+file + "/" + "report.tbl")
        with open(report,'r') as input:
            #read the top GSAT Z-score
            score = input.readlines()[2].split('\t')[3]
            mylist.append((file,score))

#sort by score in descending order
mylist.sort(key=lambda x:x[1],reverse=True)

for line in mylist:
    print(line[0]+"\t"+line[1])
```

Biological Databases

Useful data resources in bioinformatics can be found in Biological Databases. In class, we explored NCBI(The National Center for Biotechnology Information is part of the NLM, a branch of

the NIH.) and practiced searching and downloading data(E.coli K12 MG1655 genome sequences and genome annotation) from NCBI GenBank.

Exercise 2 - Exploring a genome

The goal of exercise 2 is to practice the usage of bash commands for simple data processing.

Exercise description:

From the NCBI Genome database, download the genome annotation in tabular format for E. coli K12 MG1655. Using shell commands to process the data.

The genome annotation file header is:

Column #	Header
1	Replicon Name
2	Replicon Accession
3	Start
4	Stop
5	Strand
6	GeneID
7	Locus
8	Locus tag
9	Protein product
10	Length
11	COGS(s)
12	Protein name

Exercise problems and solutions:

- What is the name (locus) and length of the largest protein?

Get the 7th(locus) and 10th(length) column from the file, sort by the 10th column first numerically in descending order, and get the first line in the sorted file.

```
cut -f 7,10 table.txt | sort -k2nr | head -1
```

- What is the name and length of the smallest protein?

  Get the 7th(locus) and 10th(length) column from the file, sort by the 10th column first numerically in ascending order, and get the first line in the sorted file.

  ```
  cut -f 7,10 table.txt | sort -k2n | head -1
  ```

- How many proteins are in the forward strand?

  Get the 5th(strand) column, use `grep` to get all the lines contain a '+'(all the proteins in the forward strand), and use `wc` to count the number of lines.

  ```
  cut -f 5 table.txt | grep '+' | wc -l
  ```

- How many proteins are in the reverse strand?

  Get the 5th(strand) column, use `grep` to get all the lines contain a '-'(all the proteins in the reverse strand), and use `wc` to count the number of lines.

  ```
  cut -f 5 table.txt | grep '-' | wc -l
  ```

- What is the largest protein in the forward strand?

  Get the 5th(strand), 7th(locus),and 10th(length) column, use `grep` to get all the lines contain a '+'(all the proteins in the forward strand), sort by the 10th column numerically in descending order, and get the first line in the sorted file.

  ```
  cut -f 5,7,10 table.txt | grep '+' | sort -k3nr | head -1
  ```

- What is the largest protein in the reverse strand?

Get the 5th(strand), 7th(locus),and 10th(length) column, use `grep` to get all the lines contain a '-'(all the proteins in the reverse strand), sort by the 10th column numerically in descending order, and get the first line in the sorted file.

```
cut -f 5,7,10 table.txt | grep '-' | sort -k3nr | head -1
```

- What are the gene names (locus) of all ribosomal proteins in the genome?

  Get the 7th(locus) and 12th(Protein name) column, use `grep` to find all the proteins those have 'ribosomal' in their protein name

  ```
  cut -f 7,12 table.txt | grep ribosomal
  ```

Exercise 3 - Extract DNA sequences for analysis

Exercise description:

- From the NCBI Genome database, download the DNA sequence of the full E. coli K12 MG1655 genome and its annotation in tabular format. Write a script that:
  - Reads the DNA sequence of the genome.
  - For each gene in the tab-separated annotations file, extract the DNA sequence and save it in a file in FASTA format.
  - If the protein is in the reverse strand you will have to complement the DNA sequence and reverse it, so it is displayed in the 5' to 3' direction in the output file.
  - Split each gene sequence in segments of 70 nucleotides before saving it.
  - This is the format that each sequence should use:

```
>NP_414543.1|thrA|b0002
ATGCGAGTGTTGAAGTTCGGCGGTACATCAGTGGCAAATGCAGAACGTTTTCTGCGTGTTGCCGATATTC
TGGAAAGCAATGCCAGGCAGGGGCAGGTGGCCACCGTCCTCTCTGCCCCCGCCAAAATCACCAACCACCT
GGTGGCGATGATTGAAAAAACCATTAGCGGCCAGGATGCTTTACCCAATATCAGCGATGCCGAACGTATT
TTTGCCGAACTTTTGACGGGACTCGCCGCCGCCCAGCCGGGGTTCCCGCTGGCGCAATTGAAAACTTTCG
TCGATCAGGAATTTGCCCAAATAAAACATGTC…TGA
```

This exercise is accomplished using python. First, the whole genome reference sequence is read in and stored. Then the gene annotation file is read line by line, gene by gene. For each gene, its

start and end position, as well as its strand direction, are first collected. Then if the protein is on the forward strand, it's gene sequence is extracted using the start and end position. It the protein is on the reverse strand, it's gene sequence is first extracted using the start and end position, and then take the reverse complement(A-T, C-G). To create a fasta file for all the genes, the basic information of each gene, the "Protein product", "Locus" and "Locus tag" are also extracted from the annotation file. Finally, after printing out the header, the sequence is outputted in segments of 70 nucleotides per line.

Python code:

```
#Read in the whole genome reference
genome = ""
with open("GCF_000005845.2_ASM584v2_genomic.fna",'r') as file:
    file.readline()
    for line in file:
        line = line.strip()
        genome = genome + line


#Function used to produce the reverse complement of a gene when the protein is
on the reverse strand
def reverse(seq):
    newseq = ""
    for i in range(len(seq)-1,-1,-1):
        if seq[i] == "A":
            newseq = newseq+"T"
        elif seq[i] == "T":
            newseq = newseq+"A"
        elif seq[i] == "C":
            newseq = newseq+"G"
        else:
            newseq = newseq + "C"
    return newseq

#Read the gene annotation file
with open("ProteinTable167_161521.txt","r") as file:
    for line in file:
        #skip the header line
        if line[0] == "#":
            continue
        line = line.strip().split('\t')
        start = int(line[2])
        end = int(line[3])
```

```
        strand = line[4]
        print(">" + line[8]+"|"+line[6]+"|"+line[7])


        if strand == "+":
            #forward strand
            seq = genome[start-1:end]

        else:
            #forward strand
            #Take reverse compliment of the genes
            seq = reverse(genome[start-1:end])

        #Print the genome sequence in segments of 70 nucleotides
        for i in range(0,len(seq),70):
            print(seq[i:i+70])
```

Mini project - protein translatability

Project description:

In this project, we want to calculate the CUI(Codon Usage Index) for each protein in E.coli K12 MG1655 to testify our hypothesis on protein translatability that If the genome has evolved to have most of its genes translated at adequate rates when it needs them, then the genomic frequencies of codons should be an adequate reference of translatability. We first generate a matrix, counting the occurrences of 64 codons for each gene as well as for all the genes in E.coli. Then we will use the matrix to calculate the frequency of each codon and then the CUI for each gene and all the genes.

Generate fasta of all the genes in E.coli K12 MG1655

- To prepare our sequences, we'll make a slight modification to the script you develop in Exercise 3 during last class. You downloaded the DNA sequence of the full E. coli K12 MG1655 genome and its annotation in tabular format. Your script should now:
    - Read the DNA sequence of the genome.
    - For each gene in the tab-separated annotations file, extract the DNA sequence and save it in a two column format. The first column should be the locus_tag (b-number) and the second column the DNA sequence of the gene in one string:

```
b0001   ATGAAACGC…TGA
b0002   ATGCGAGTG…TGA
b0003   ATGGTTAAA…TAA
```

To achieve the modification, we modified the output format of the fasta in the python script of exercise 3. Instead of printing the genome sequences in segments, print them in the same line. Instead of listing the the "Protein product", "Locus" and "Locus tag" in the header for each gene, only uses "Locus tag".

The modified python script:

```
with open("ProteinTable167_161521.txt","r") as file:
    for line in file:
        if line[0] == "#":
            #skip annotation header
            continue
        line = line.strip().split('\t')
        start = int(line[2])
        end = int(line[3])
        strand = line[4]
        if strand == "+":
            #forward strand
            seq = genome[start-1:end]
        else:
            #reverse strand
            seq = reverse(genome[start-1:end])

        #output locus tag and genome sequence separated by tab
        print line[7]+'\t'+seq
```

Count codon occurrences and compute CUI score

To count the codon occurrences in each gene, the fasta file is read line by line. The genome sequences is separated into segments of 3 nucleotides(codons). Iterate through each codon in the gene and print the occurrences, separated by tab. Then the total occurrences of each codon are stored in a dictionary(python hashmap) and outputted at the end of the file. For each gene, the relative codon frequencies $q_{g,i} = \frac{n_{g,i}(c)}{L_{g,i}}$ are also calculated and stored in a $n \times 64$matrix(n = number of gene). Since the calculation of the CUI requires the global relative frequencies of each codon $p_g(c) = \frac{N_g(c)}{T_g}$, after all the genes have been read and the frequencies are calculated, the

CUIs for each gene are then calculated using the formula:

$$CUI_{g,i} = \sum_{c=1}^{64} q_{g,i}(c) p_g(c)$$

Finally, a histogram of the CUIs in E.coli is plotted.

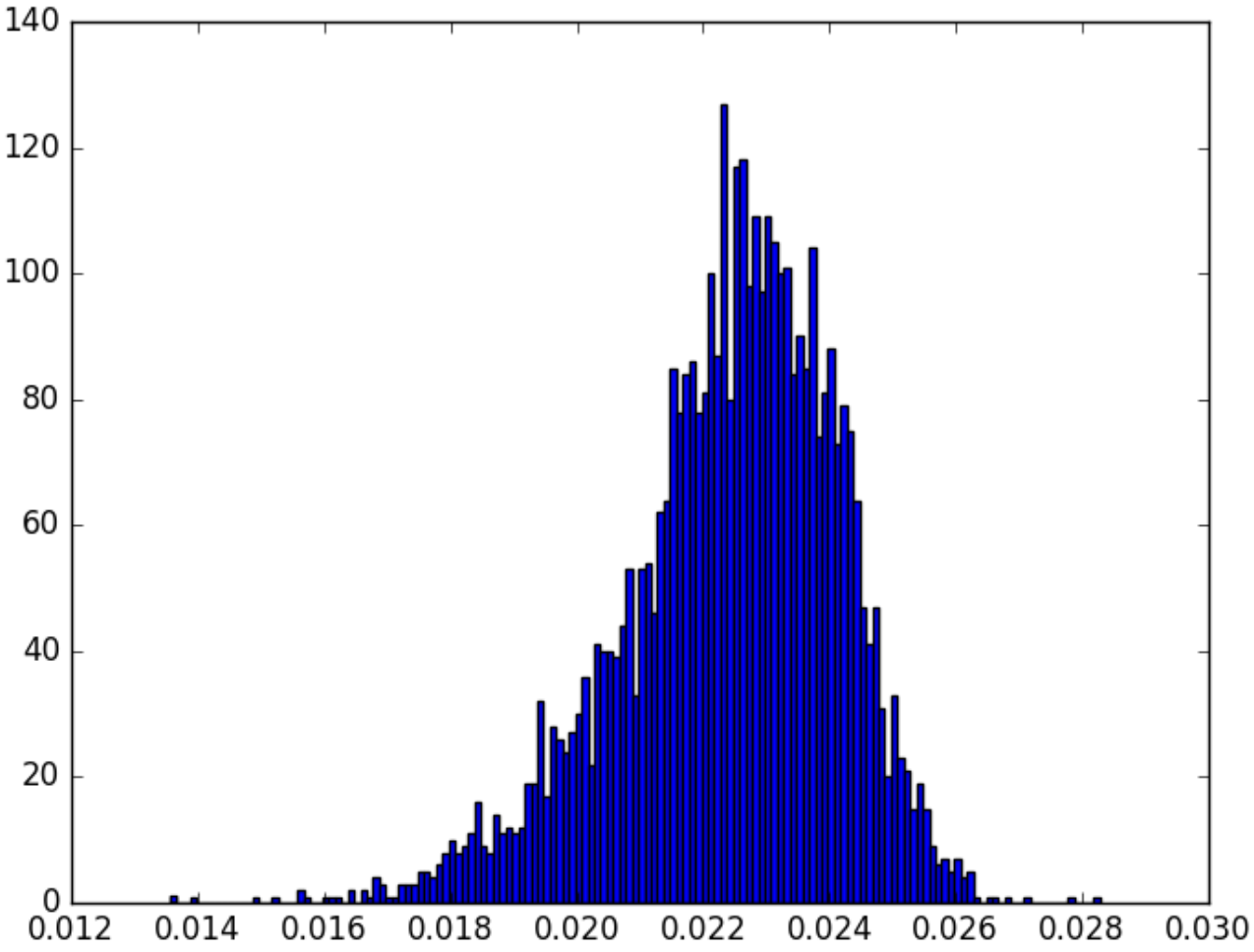
Python code

```
import textwrap
import matplotlib.pyplot as plt
import numpy as np

#Use the the listed order of codon in the output
codon_list = ["ATT", "ATC", "ATA","CTT", "CTC", "CTA", "CTG", "TTA", "TTG","GTT
", "GTC", "GTA", "GTG","TTT", "TTC","ATG","TGT", "TGC","GCT", "GCC", "GCA", "GC
G","GGT", "GGC", "GGA", "GGG","CCT", "CCC", "CCA", "CCG","ACT", "ACC", "ACA", "
ACG","TCT", "TCC", "TCA", "TCG", "AGT", "AGC","TAT", "TAC","TGG","CAA", "CAG","
AAT", "AAC","CAT", "CAC","GAA", "GAG","GAT", "GAC","AAA", "AAG","CGT", "CGC", "
CGA", "CGG", "AGA", "AGG","TAA", "TAG", "TGA"]

#Output header
print ("Gene",end='\t')
for c in codon_list:
        print (c,end='\t')
print ("Length")


counts = {}
freqs = []
genes = []
readfile = open("exercise2_modified.txt",'r')
gene_prob = open("gene_prob.txt",'w')
for line in readfile:
        freq = [0] * 64
        line = line.strip().split('\t')
        gene = line[0]
        genes.append(gene)
        #Separated genome sequence into segments of 3 nucleotides(codons)
        codons = textwrap.wrap(line[1],3)
        print (gene,end='\t')
        #Count occurrences of each codon in order
        for i in range(len(codon_list)):
                c = codon_list[i]
```

```
                #print(codons)
                print(codons.count(c),end='\t')
                #increment global codon occurrences and store in a dictionary
                if c in counts:
                        counts[c] += codons.count(c)
                else:
                        counts[c] = codons.count(c)
                #calculate relative frequencies for each codon
                freq[i] = float(codons.count(c)/(len(line[1])/3))
        #store the frequencies in the matrix
        freqs.append(freq)

        #output total codon count of each gene
        print(len(line[1])/3,end='\t')

        #if the length of the gene is not multiply of 3, output warning message
        if len(line[1]) % 3 != 0:
            print ("Seq length not multiply of 3")
        else:
                print()

#Output the global occurrences of each codon
print("Totals",end='\t')
for c in codon_list:
        print(counts[c],end="\t")
print(sum(counts.values()))

#Calculate CUIs
total_CUI = 0
CUIS = []
for i in range(len(genes)):
        #iterate through each gene
        CUI = 0
        for j in range(len(codon_list)):
                c = codon_list[j]
                CUI += freqs[i][j] * counts[c]/sum(counts.values())
        total_CUI += CUI
        CUIS.append(CUI)
        gene_prob.write(genes[i] + "\t" + str(CUI) + "\n")

#Plot the histogram
binwidth = 0.0001
plt.hist(CUIS, bins=np.arange(min(CUIS), max(CUIS) + binwidth, binwidth))
plt.savefig("count_hist.png")
```

Histogram of CUIs in E.coli K12 MG1655

It can be observed from the histogram, that the distribution of CUIs is bell-shaped, and the mean is shifted to the right compare to a symmetric bell-shaped distribution ($mean = \frac{max}{min}$). This indicates that E.coli has optimized some of its genes to have higher translatability and so that the number of genes those are semi-optimized(optimized to a level where it's translatability is enough for this gene to function) are more than the number of genes that are semi-optimized in a randomly generated gene library. Further investigations on the identities of the proteins with high CUI scores can further testify our hypothesis. More investigations can also be done using the global relative frequencies of E.coli and relative frequencies of genes from other species. It can be used to discover the translatability of foreign genes in E.coli as if a direct transfer of the genes from another species would happen.



Python source codes - GitHub

All of the python sources codes for Exercise 1, Exercise 3, as well as the mini project can be found on GitHub in the Week2 folder. The GitHub link is: https://github.com/luna5124/BIMM185