BioE 131/231
Fall 2018

Lab Assignment #7

Overview and references

You will write a small bit of Python code to generate random binary and FASTA files. Your random binary files will contain a specified percentage of zeroes and ones, and your random FASTA files will contain either DNA or protein sequences. Using this code, you will generate a set of random data which you will then compress using gzip (similar to LZW) and bzip2 (uses Burrows-Wheeler), and arithmetic coding. You will generate a table comparing the original file size, compressed file size, and run time for each algorithm on each sequence.

You are expected to keep a thorough record of everything you did in your notebook. Create a folder in your home directory for each lab, and keep all your files there. Try to create a directory hierarchy that makes sense, like the one we went over in Lab 1. Copy and paste any terminal commands you used into a Markdown section and explain what the input was, what the tool did, and what the output was. Plot any results in-line and explain them.

An iPython notebook containing your analysis is due at midnight on Wednesday of next week. You can upload a link to your GitHub repo on bCourses.

Information theory
https://en.wikipedia.org/wiki/Entropy_(information_theory)

Compression algorithms
https://en.wikipedia.org/wiki/Gzip
https://en.wikipedia.org/wiki/Bzip2
https://en.wikipedia.org/wiki/Arithmetic_coding

Numpy
https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.choice.html
https://docs.scipy.org/doc/numpy/reference/generated/numpy.packbits.html

Single-letter amino acid code
https://en.wikipedia.org/wiki/Proteinogenic_amino_acid

HIV gp120 envelope protein
https://en.wikipedia.org/wiki/Envelope_glycoprotein_GP120

Background
You're working at a biotech company that generates 1000 terabytes of data every day. In a meeting, your boss mentions that it costs the company $50 per terabyte of hard disk space, and so every 1% reduction in data that must be stored translates into a $500 savings per day. Your team will get a bonus this year equal to the amount of savings you're able to generate by compressing the company's data.

Incentivized by this bonus, your team gets to work determining which compression algorithm will generate the most savings. The algorithm you choose must either be quick enough to compress 1000 terabytes a day, or efficient enough that even if all the data isn't compressed, the savings are maximized by the data that you have time to compress.

Simulating the data
Fortunately, you know that most of the data you'll be compressing will be nucleic acid sequences. Some of it, however, will be binary files, and some will be protein sequences. Start by writing some code to simulate files containing random DNA, protein, and binary data.

Using `np.random.choice`, generate **100 megabytes** (8 bits/byte * 1024 bytes/kilobyte * 1024 kilobytes/megabyte * 100) of random data containing 100%, 90%, 80%, 70%, 60%, and 50% zeros. Be sure to call `np.packbits` on your data before writing it to a file. For example:

```
myvar = np.random.choice([0, 1], size=1024, replace=True, p=[0.5, 0.5])
myvar = np.packbits(myvar)
```

Then write this data to a file in your home directory, like this:

```
open("zeros_100p", "wb").write(zeros_100p)
```

Next, generate DNA and protein sequences **100 million** letters long and write those to your home directory. The probability of each letter should be equal. To write strings generated in Numpy to a file, you'll have to use a slightly different command, like this:

```
open("nt_seq.fa", "w").write("".join(my_nt_seq))
```

Compressing the data
You'll have to do this from the terminal on bioe131.com via SSH (PuTTY on Windows) or via iPython in your web browser. On each of the files you generated above, run gzip, bzip, pbzip2 and ArithmeticCompress as follows:

```
time gzip -k zeros_100p
time bzip2 -k zeros_100p
time pbzip2 -k zeros_100p
time ArithmeticCompress zeros_100p zeros_100p.art
```

The time command on Linux will run whatever commands follow and report how long it took. Gzip and Bzip2 will silently compress your file and produce a new file with the extension .gz or

.bz2, respectively. The –k option tells them not to delete the original file once the compression has completed. ArithmeticCompress takes the input file as the first parameter and the output file as the second parameter.

Keep track of the size of the input files, the size of the output files, and the time each command took to run in a table in your iPython notebook.

<u>**Questions**</u>
Please answer these in your iPython notebook.

Which algorithm achieves the best level of compression on each file type?
Which algorithm is the fastest?
What is the difference between bzip2 and pbzip2? Do you expect one to be faster and why?

How does the level of compression change as the percentage of zeros increases? Why does this happen?

What is the minimum number of bits required to store a single DNA base?
What is the minimum number of bits required to store an amino acid letter?
In your tests, how many bits did gzip and bzip2 actually require to store your random DNA and protein sequences?
Are gzip and bzip2 performing well on DNA and proteins?

<u>**Compressing real data**</u>
Now that you have a sense of how random data can be compressed, let's have a look at some real biological data. Using what you've learned about querying biological databases, find the nucleic acid sequences of gp120 homologs from at least 10 different HIV isolates and concatenate them together into a single multi-FASTA.

A priori, do you expect to achieve better or worse compression here than random data? Why?

Now, compress the multi-FASTA using gzip, bzip2, and arithmetic coding.

How does the compression ratio of this file compare to random data?

<u>**Estimating compression of 1000 terabytes**</u>
Let's make some assumptions about the contents of the data at your biotech company. Most of the data, say 80%, is re-sequencing of genomes and plasmids that are very similar to each other. Another 10% might be protein sequences, and the last 10% are binary microscope images which we'll assume follow the worst-case scenario of being completely random.

Given the benchmarking data you obtained in this lab, which algorithm do you propose to use for each type of data? Provide an estimate for the fraction of space you can save using your compression scheme. How much of a bonus do you anticipate receiving this year?