Data Analysis, Homework 3

Elizabeth Gould

June 6, 2025

1 Compressor Statistics

1.1 Intro

- Escape probability given by C.
- In an old version of the code, if there is an extremely rare character or context, the character will not be correctly read, after which the decoder will soon encounter an error. I have fixed this problem, but many of the statistics use the old decoder.
- The number of cycles where the last bit doesn't change at the end was not read properly. I believe this has been fixed, but it doesn't effect statistics in any case.
- There is a new wrapper code. They will all give results with two more bytes (k and N) than the statistics for which I have here. I just hadn't thought about sending these two numbers.
  - encode.run_encoder(import_file, export_file, k = None, N = 64)
    * import file and export file are file names
    * k = None will try to calculate k based on the most likely character and context, otherwise k gives the desired context length
    * N is the length of the window, and must be a whole number of bytes
  - encode.run_decoder(import_file, export_file)

1.2 Texts

1. With This Ring Season 1: Text has been cleaned. Size is 4115 kB on disk.
2. With This Ring Season 2: Text is uncleaned. Size is 5632 kB on disk.
3. With This Ring Season 2: Text still uncleaned, but one section has been removed. Size is 5632 kB on disk.
4. With This Ring Season 1 + 2: This is just text 1 and text 3 combined into one file. Size is 9746 kB on disk.

5. With This Ring Season 1: Decapitalized version of the text. It is actually not saved to disk, as it is easy to produce in Python.
6. With This Ring Season 1: Reversed version of the text. It is actually not saved to disk, as it is easy to produce in Python.
7. With This Ring Season 1: Decapitalized and reverse version of the text. It is actually not saved to disk, as it is easy to produce in Python.

1.3 Static Machines

Compressor as of right now only works on the first text. It should be easily alterable to work on the other texts, but I would need to clean the second text, and there is an issue with time requirements.

- k = 3
- window size = 40 bits
- Decapitalization time: 1.5 hours
- Decapitalization size: 24 kB
- find count time: 1.5 hours, of which 20 min actually unnecessary
- find count size: 89 kB on disk
- encryption time: 3 hours, but less for an alternative counter
- encryption size: 1012 kB
- full decryption time: 70 min almost all of which are decryption

The long time here compared to the short time for the dynamic code indicates an inefficiency in the code. I have very little desire to find it, but the only difference is how the counts are stored, so it seems a recursive storage is best. I believe I can search the code in a single pass for better efficiency for decapitalization and find counts.

The old method for storing the encoder array took 20 minutes to encode. I think searching the long arrays is inefficient. If I need to rerun this, I will try to clear the inefficiencies.

1.4 Dynamic Machines without Masking

For text 2, N = 64 too small. I found a chunk of text strangely written (ultra rare). I deleted the problematic part of the text and tried again.

For text 4, as I suspected, the decoder reaches a problem at the first non-ascii character.

- k = 3

- window size = 64 bits
- encoder time, text1: 3 min
- decoder time, text1: 4.5 min
- size text1: 1181 kB
- encoder time, text3: 5+ min
- decoder time, text3: 6-10 min
- size text3: 1575 kB

1.5 Dynamic Machines with Masking (Update Exclusions)

In this section, I added masking of characters which we already ruled out. So far, doesn't seem faster, but might save some on size. It bypasses the problem with decoding ultra-rare cases, but for text2, I encountered the end of file issue. Manually telling it to decode the last four characters returns the original text, so it is just stopping at the wrong place. Text1 has an end of file issue of 1 character. I stopped looking for this afterwords. The data is in table 1.

The decoder is still failing at the first non-ascii character for text 4. This may require the second fix which I noted in the issues section. For all except k = 3, text2 also fails. I have updated the decoder, so the statistics for text2 use a different decoder than the others, which works for text2.

k = 6 appears to be the best variant from the table, but k = 5 is not much worse and faster. This contrasts with the k = 4 estimate for the recommended length of k.

1.6 Choice of First Model

The data for the probabilities for the probability of the most likely character are given in Table 2. For every step, I found the most likely character out of the remaining characters, adding a character onto the beginning. By the end, my count is only 9. I believe that the method I use here to estimate the best k is not the correct version, it is not very useful at all, but every other variant I can think of would run into the time issues of the static machine count, which I was trying to avoid.

The current calculation takes 30 seconds on my original computer and 5s on my faster computer.

2 Wikipedia Benchmark

Timings are done on a separate computer with the updated decoder. I have an update of Table 1 given in Table 3 for comparison. All of these use the most recent version of the decoder.

The times for the 100 MB benchmark are:

Table 1: Encoding and decoding with masking time and space requirements based on text and maximum length of context. Order is – number of bits, encoding time, decoding time. Note that text 2 uses an updated decoder, so the times are not based on the same metric. Also all the timings are unstable.

	Text 1	**Text 2**	**Text 3**	**Reverse**	**Decap**	**Decap Rev**
find_k k	k=4	k=4	k=4	k=8	k=4	k=8
k=2	1476416 3 m 23.8s 5m 23.1s	1990550 4m 32.5s 10m 4.9s	1990215 4m 57.6s 9m 7.4s	1476623 2m 52.5s 5m 11.9s	1470545 3m 14 s 4m 37.9s	1470470 3m 32.9s 7m 28.3s
k=3	1192730 3 m 9.2s 4m 50.1s	1593626 4m 13.5s 9m 29.2s	1593289 5m 29.2s 7m 14.1s	1193200 3m 6.0 s 5m 3.6 s	1190498 2m 48.7s 4m 28.7s	1190568 4m 12.8s 6m 27.3s
k=4	1058909 3m 36.6s 5m 29.3s	1399465 6m 1.2s 10m 54.4s	1399128 6m 6.4s 9m 46.1s	1059650 3m 55.0s 6m 6.1s	1057195 3m 25.5s 5m 4.1s	1057632 4m 54s 7m 20.7s
k=5	1017021 4 m 22.6s 6m 2.8s	1335544 5m 46.9s 8m 56.5s	1335207 7m 32.5s 10m 49.4s	1018142 5m 1.3s 7m 20.3s	1012264 3 m 51.6s 5m 17.3s	1013437 6m 21.1s 6m 47.7s
k=6	1012453 6 m 10.6s 6m 34.5s	1325093 7m 10.2s 10m 30.9s	1324756 8m 27.2s 10m 11.2s	1013572 6m 11.2s 8m 32.9s	1004875 5 m 24.5s 6m 43.4s	1007044 5m 38.3s 7m 1.8s
k=7	1023743 6m59.5s 8m 8.7s	1338156 9m 91.s 11m 52.8s	1337819 9m 49.2s 12m 15.4s	1024888 7m 38.6s 9m 12.8s	1014294 6m 11.5s 7m 43.6s	1018111 6m 37.7s 8m 2.0s
k=8	1040281 8m 14.0s 10m 29.9s	1359464 11m 59.5s 14m 30.2s	1359126 12m 55.5s 14m 58.7s	1041179 8.5m error	1029459 7m 47.5s 8m 50.4s	1034963 7m 48.6s 8m 36.4s

Table 2: Probability of the most frequent character given the most frequent context.

k	Probability	Added Bit
0	0.172	32
1	0.166	101
2	0.327	104
3	0.750	116
4	0.997	32
5	0.153	110
6	0.474	105
7	0.944	32
8	0.191	101
9	0.233	114
10	0.333	101
11	0.566	104
12	0.529	119
13	0.667	101

- estimated k = 4
- time to estimate k = 1m 44.0s
- k = 5
- size = 24,329,385
- encode time = 34m 50.3s
- decode time = 57m 31.0s
- k = 6
- size = 23,483,486
- encode time = 41m 49.8s
- decode time = 61m 39.8s

And for the 1GB benchmark are:

- estimated k = 4
- time to estimate k = 15m 41.0s
- k = 5
- size = 218,062,631

Table 3: Encoding an decoding time based on text and maximum length of context on a faster computer for comparison with Wikipedia benchmark times. Order is – encrypted size in bytes, encoding time, decoding time. Text 4 has an estimated k of 4.

k	Text 1	Text 2	Text 3	Text 4	Rev	Decap	Dec+Rev
2	1476416 55.2s 2m 14.8s	1990550 1m 18.3s 3m 9.4s	1990215 1m 16.9s 3m 8.5s	3487140 2m 19.4s 5m 43.9s	1476623 53.5s 2m 5.9s	1470545 54.1s 2m 10.2s	1470470 55.9s 2m 10.3s
3	1192730 1m 0.9s 2m 0.2s	1593626 1m 24.4s 2m 44.1s	1593289 1m 24.6s 2m 45.7s	2789743 2m 33.3s 5m 6.9s	1193200 58s 1m 54.2s	1190498 58.6s 1m 55.2s	1190568 1m 0.2s 1m 58s
4	1058909 1m 12.5s 2m 3.6s	1399465 1m 39.3s 2m 49.1s	1399128 1m 40.5s 2m 49.2s	2438273 4m 34.9s 7m 54.8s	1059650 1m 8.8s 1m 57.4s	1057195 1m 8.5s 1m 56.8s	1057632 1m 11.2s 2m 3.1s
5	1017021 1m 28.4s 2m 15.4s	1335544 1m 59.1s 3m 3.9s	1335207 1m 59.3s 3m 2.6s	2311383 5m 28.8s 6m 11.8s	1018142 1m 22.9s 2m 9.0s	1012264 1m 22.9s 2m 7.4s	1013437 1m 25.8s 2m 13s
6	1012453 1m 49.5s 2m 32.4s	1325093 2m 27.5s 3m 25.8s	1324756 2m 24.7s 3m 25.1s	2280430 4m 22.2s 6m 11.9s	1013572 1m 42.9s 2m 25.9s	1004875 1m 42.6s 2m 26.8s	1007044 1m 47.4s 2m 32s
7	1023743 2m 13.2s 2m 55.8s	1338156 3m 0.1s 3m 55.4s	1337819 2m 59.8s 3m 57.2s	2294026 5m 20.1s 6m 57s	1024888 2m 8.6s 2m 48.3s	1014294 2m 7.9s 2m 47.4s	1018111 2m 14s 2m 54.6s
8	1040281 2m 44.4s 3m 22.7s	1359464 3m 38.2s 4m 32s	1359126 3m 37.3s 4m 30.3s	2326546 6m 24.2s 8m 2.2s	1041179 2m 38.1s 3m 14.7s	1029459 2m 36.4s 3m 13.8s	1034963 2m 45.1s 3m 21.1s

- encode time = 353m 52.7s
- decode time = 561m 52.7s