3F7 Laboratory
Guidance for Full Technical Reports

Dr Jossy Sayir
js851@cam.ac.uk

Michaelmas 2021

In the Full Technical Report (FTR), you will be guided through two extensions of the short lab:

1. compressing text files such as Hamlet while taking the context into account rather than approximating the file as a sequence of independent and identically distributed random variables;

2. compressing files adaptively or "on the fly" while building up a probabilistic source model, instead of the two-pass approach taken in the short lab where source statistics were measured offline and stored in a separate file available to the decoder.

These extensions will be investigated and documented based on theory and statistical measurements. While we encourage anyone who is a good programmer to also implement the compression algorithms resulting from your theoretical investigation, implementation of algorithms will not be formally assessed as part of the FTR. This is because we want to keep this lab accessible for students of all programming levels, and while we do believe that any IIA student taking this module should have the skill to implement the methods developed in this exercise, we also understand that this may take some students several days to complete and hence making implementation part of the score would unfairly disadvantage such students. Your aim in the FTR should be to document the information theoretic knowledge you have acquired as part of the exercise, rather than your prowess at implementing complex algorithms.

1 Extenstion to the short lab

1.1 Sources with memory

The best compression ratio you obtained for the text of Hamlet in the short lab lies somewhere just above 4 bits per source byte. This is not a very good compression ratio. Try to compress Hamlet with a commercial compression tool such as `gzip` or `bzip` and you will see that it achieves a far lower compression ratio around 2.9 bits per byte. So how do they do it? Have they found a strategy to beat the source coding theorem that states that the entropy is the lower bound for the expected codeword length for any uniquely decodable (or prefix free) code? Are their codes not uniquely decodable?

The answer is that their codes are of course uniquely decodable and they are not beating the source coding theorem, but they are considering files as a source *with* memory whereas in the short lab, we approximated text as a string of indepedent and identically distributed (i.i.d.) random variables. This meant that the single letter entropy $H(X_i) = H(X)$ for all i was the lower bound for our compression techniques, but this lower bound does not apply to compression of sources with memory. So what is the correct limit of compressibility for sources with memory?

The answer lies in a very elegant set of results that are not part of the 3F7 course contents but completely accessible to you with your current level of knowledge, so we propose to investigate these as our first extension to the short lab.

We first assume that we are dealing with *stationary* sources. If you are taking 3F3, then you will already have learned the definition of a stationary random process. If you are not taking 3F3, spend a few minutes researching this concept. You do not need to understand the concept in all its intricacies, and make sure you limit your attention to stationarity for *discrete* random processes (stationarity for continuous random processes is a far more intricate concept and is covered in 3F1, for those of you taking that module.) You should reach a level of understanding sufficient to answer the following two questions:

- for a stationary source of random variables $X_1, X_2, \ldots$, why is $H(X_1, X_2, \ldots, X_n) = H(X_{k+1}, X_{k+2}, \ldots, X_{k+n})$ for any k and n?
- for a stationary source of random variables $X_1, X_2, \ldots$, why is $H(X_n|X_1, X_2, \ldots, X_{n-1}) = H(X_k|X_{k-n+1}, \ldots, X_{k-1})$ for any k and n?

Next, assuming a stationary source of random variables $X_1, X_2, \ldots$, we define the following two sequences

$$H_n = \frac{1}{n} H(X_1 \ldots X_n) \tag{1}$$

$$H_n^\star = H(X_n|X_1 \ldots X_{n-1}) \tag{2}$$

The following theorem gives us some indications as to the usefulness and meaning of these sequences:

Theorem 1 *The following statements hold:*

1. $H_n^\star$ *is non-increasing as* n *increases (i.e.,* $H_{n+1}^\star \leq H_n^\star$ *for all* n*)*
2. H_n *is non-increasing as* n *increases*
3. $H_n^\star \leq H_n$ *for all* n
4. *The limits of* H_n *and* $H_n^\star$ *as* n *goes to infinity exist and* $\lim_{n\to\infty} H_n = \lim_{n\to\infty} H_n^\star$

You can try to prove the statements in the theorem and supply the proof as part of your FTR report if you wish. Statement 1 is very easy to prove, and you should find it fairly easy to prove statements 2 and 3 as well. For statement 4, you can prove that the limits exist by showing that H_n and $H_n^\star$ are lower-bounded by any finite number: any lower bounded non-increasing sequence must have a limit. In order to prove that the limits are equal, you

can show that $\lim_{n\to\infty} H_n \geq \lim_{n\to\infty} H_n^\star$ and $\lim_{n\to\infty} H_n \leq \lim_{n\to\infty} H_n^\star$. The first of these inequalities is very easy to show. For the second inequality, show that

$$\lim_{n\to\infty} H_n = \lim_{n\to\infty} H_{N+n} \leq H_N^\star \text{ for any } N$$

by expressing H_{N+n} and upper bounding it by an expression of H_N and $H_N^\star$, then letting n go to infinity. Now letting N go to infinity, you've proved the inequality.

The limit in step 4 of the theorem has a name:

Definition 1 *The entropy rate of a discrete stationary source of random variables $X_1, X_2, \ldots$, denoted $H_\infty(X)$, is defined as*

$$H_\infty(X) = \lim_{n\to\infty} H_n = \lim_{n\to\infty} H_n^\star.$$

After thinking of the meaning and repercussions of this theorem and definition, you should now have sufficient knowledge to prove and understand the coding theorem for discrete stationary sources:

Theorem 2 *For any discrete stationary source of random variables $X_1, X_2, \ldots$, the average codeword length for a prefix-free code encoding blocks $X_{kn+1}, \ldots, X_{(k+1)n}$ for $k = 0, 1, 2, \ldots$ of n symbols at the time into words of length L_k satisfies*

$$E[L_k]/n \geq H_\infty(X).$$

Moreover, for any $\epsilon > 0$, there exist coding techniques that will achieve

$$E[L_k]/n < H_\infty(X) + \frac{1+\epsilon}{n}$$

for n large enough.

Note that the presence of an ϵ in the upper bound is a technicality and for all essential purposes, the theorem mirrors the coding theorem you've learned in lectures for discrete memoryless sources, for which the expected codeword length achieves the entropy plus $1/n$ as a strict upper bound.

After spending some time thinking about these results (and possibly proving them), think about what this implies in terms of the coding techniques you've learned during the short lab. How would you implement Huffman and arithmetic coding for discrete stationary sources to achieve a good compression rate close to the entropy rate of a discrete stationary source? Can you think of a practical compression technique for which n can be made very large and hence have performance approaching the entropy rate, and how would this work in practice?

For measuring the entropy rate, take the text file `hamlet.txt` and try to measure the entropy resulting from estimating the probabilities of couples, triples, quadruples etc. of consecutive symbols in the text file. We've provided a Jupyter notebook to help you measure these. You will observe that while your estimates of H_1, H_2 and possibly H_3 are fairly accurate, the estimated entropies keep decreasing with apparently no positive lower bound, implying possibly that $H_\infty = 0$. Do you really believe that the entropy rate of text is zero? What do you think may be going wrong in this experiment?

1.2 Estimating probabilities

The problem with the experiment in the previous section is that, while a file of length 207,039 may have seemed very long when estimating the probabilities of single ASCII characters, once you go to n-tuples the data becomes more and more sparse and the probabilities you are estimating are no longer accurate enough for our entropy measurement. If you've given some thought to compressing discrete memoryless sources to approach the entropy rate, you'll agree that if these probabilities aren't good enough for entropy measurement, they are also not good enough to realistically achieve good compression with a practical algorithm that uses them as a basis for compressing data. Of course, in the extreme, if you have access to an offline dictionary measured from the file `hamlet.txt` for all tuples of length n for $n = 1, 2, \ldots, 207039$, then since the probability distribution for $n = 207039$ will assign a probability of 1 to the whole text of Hamlet, you'd have achieved a compression rate of 0 since the decoder doesn't need any compressed file to know that, with probability one, the source data is `hamlet.txt`. This is irrelevant and not much use for practical compression.

The problem of probability estimation is even more interesting when you consider that a good compression algorithm will in fact not need a separate stored probability model. Compression can proceed starting from a startup probability model that assigns for example a uniform distribution to all symbols, and adapt its model "on the fly" as you progress through the file. As long as you always encode a symbol using the "old" model before updating your estimated probabilities to get a "new" model, the decoder will be able to decode the source symbol and update the source model to mirror exactly the updates done by the encoder.

In IB Paper 7, you've learned about the Beta distribution (Lecture slides 3) for estimating the unknown parameter of a source of Bernoulli random variables. If p is the unknown Bernoulli parameter $p = P_X(1)$ of a binary source, by assuming that p is a random variable P uniformly distributed between zero and one ("uniform prior"), then the distribution of P conditioned on the data if you've observed n_0 zeros and n_1 ones is a Beta distribution $\text{Beta}(n_0, n_1)$. The Laplace estimator is the expected value of the Beta distribution $E[P|X_1, \ldots, X_n]$ where $X_1, \ldots, X_n$ is the data seen after $n = n_0 + n_1$ observations

$$E[P|X_1, \ldots, X_k] = \frac{n_1 + 1}{n_0 + n_1 + 2},$$

which is the same as saying that before you start observing source outputs, you assume that you've already seen one 0 and one 1. You can prove all of the above if you feel adventurous. If you assume a different prior on P you get different estimators but they always tend to be of the form

$$\hat{P} = \frac{n_1 + \delta}{n_0 + n_1 + 2\delta}$$

for some real number $\delta \geq 0$, i.e., you bias both counters with some value δ that isn't necessarily an integer.

Binary sources are of limited interest. In our short experiment we were dealing with sources that didn't have one unknown parameter but 127 unknown parameters (the probabilities of all ASCII symbols, minus one symbol that is naturally one minus the sum of all other parameters.) Extending the theory above to non-binary symbols is a very difficult task and you should by no means undertake it, but the answers for reasonable priors comes out as an estimator that again gives a bias δ to all counters and then divides each biased counter by the sum of all biased counters. In other words, if you have observed data $X_1, \ldots, X_n$ from

a discrete random process over an alphabet of size k, and n_x is the number of times you've observed symbol x, then your estimate for the probability of symbol x would be

$$\hat{p_x} = \frac{n_x + \delta}{n + k\delta}. \tag{3}$$

Without actually implementing an adaptive arithmetic code, you can get a good estimate of its performance by assuming that the length of the code sequence for a string of data $x_1, x_2, \ldots, x_n$ whose estimated probability is $\hat{p} = \prod_i \hat{p}_{x_i}$ is

$$L = -\log \hat{p} = -\log \prod_{i=1}^{n} \hat{p}_{x_i} = -\sum_{i=1}^{n} \log_2 \hat{p}_{x_i}. \tag{4}$$

Based on this estimate, you can investigate and answer the following questions by conducting a few statistical measurements on the file `hamlet.txt`:

- how does the performance of adaptive arithmetic coding approach the entropy when probabilities are adapted at each step using the estimator in (3) in function of the data length n? You could plot L/n calculating L based on (4) using the first n symbols in Hamlet, for n going from one to, say, 10,000, using, say $\delta = 1$ (as in the Laplace estimator). Does the resulting performance tend to the entropy of the distribution?

- How does the bias δ affect the performance of the compression algorithm? Try various values for δ and give an indication of how δ is best picked. If you're feeling adventurous, you could try to search for the optimal δ for a few compressed files using a Python search library.

The above seems to indicate that, while adaptive arithmetic coding works and can approach the performance of non-adaptive arithmetic coding and also approach the entropy, it is always at a disadvantage with respect to non-adaptive coding. This argument gives an unfair advantage to non-adaptive arithmetic coding by omitting to account for the storage requirement for storing the probability model offline. Of course, there may be some scenarios where storage of such a model can indeed be discounted, for example if compressing many files with a unique probability model that has been measured offline and can be viewed as "part of the software package" rather than stored specifically for each file as we have done in the short lab. For other scenarios, you could also compare the performance of the two while accounting for the fixed extra storage need for the offline model. What are your conclusions from this comparison? Is there a "sweet spot" for the file length from when it becomes more advantageuous to store the model offline than to measure it on the fly?

Finally, you should give some thought to combining what you've learned in both sections of your investigation: how does adaptive probability modelling apply to sources with memory? What are the advantages and drawbacks of considering ever longer block lengths or context lengths, if you are working with estimated probabilities rather than assuming that probabilities are just given to you and accurate as we assumed in the previous section?

2 Writing your report

Your report should cover the following parts:

- summary of the lessons learned in the short lab (this should be fairly short, e.g., one page, and stick to the essentials)
- coding for discrete stationary sources (about 3 pages)
- adaptive coding using probability models measured "on the fly" (about 3 pages)

Your task is to explain the lessons you've learned by following the guidance in the previous section, using your own words, and trying to formulate your thoughts as clearly as possible. You are welcome to repeat information in the guidance (e.g. the theorem) as long as you don't copy sentences word for word (copying equations is of course allowed.)

While you are welcome to provide proofs of all results and implementations of algorithms, the assessment will focus principally on how well you have understood and how clearly you have described the information theoretic concepts elaborated in this report. Add figures, flowcharts and graphs if you wish and ensure that they are all labeled and referred to in the text. As per the guidance issued for all IIA FTRs, your report should not exceed 10 pages, but given the nature of this work, note that there is no obligation to submit reports near the maximum number of pages. We suggest that a reasonable length for your report would be 6-7 pages. There is no need to attach a standard cover sheet to your FTRs as the 3F7 lab will be assessed using a specific assessment and feedback sheet, but please ensure that your report bears your name on the first page. You must submit your report electronically as a PDF on moodle (please do not submit as a word document! If using word, convert your document to PDF format first). You can submit extra files such as programs you've written on moodle as well. Remember that these will not give you extra points by themselves, but we may look at them if they were instrumental in reaching particularly interesting insights that are documented within your FTR report. There is no restriction on the length or number of such extra files (unlike the guidance for FTRs in general) but any appendices provided within the PDF of your report must adhere to the guidance on maximum page length. Submitting source code of programmes you've written as a text appendix is strongly discouraged. If you read any further articles such as technical papers, articles in encyclopedias or book chapters and want to refer to them in your report, please include a reference list. Every reference in a reference list should be referred to in the text of your report. There is no need to include the lab guidance as a reference in your report. If you chose to repeat a sentence word for word from a reference, it becomes a quote and must be placed in quotation marks and be immediately followed or preceded by the reference, for example, as stated in [2], "the quick brown fox jumped over the lazy dog". The submission deadline for the 3F7 FTR is the submission deadline for all FTRs advertised on the IIA guide on `teaching.eng.cam.ac.uk`

Feedback will be provided on moodle.