University of Sussex

INDIVIDUAL PROJECT

UNIVERSITY OF SUSSEX

SCHOOL OF ENGINEERING AND INFORMATICS

Compression of Natural Language

Author:
Guy Aziz
267649

Supervisor:
Prof. Ian Mackie

April 12, 2024

Contents

1. Introduction

Human language contains many redundancies and regularities. It is possible, for example, for a person to infer from an incomplete sentence a missing letter or word, or to spot an error through an inconsistency between text and context.

This is done, first, through the abstraction of an underlying, concise representation of the text, and then through a re-generation of the elaborated text from the conceptual understanding. In humans, one's ability to induce a general pattern and deduce its application to a specific case in this way is often an indication of their comprehension of a text, and of having a grasp of the underlying meaning.

Because of this, it is believed by some researchers (most notably Marcus Hutter) that the ideal lossless compression of a text would require comprehension, and that the first is therefore an AI-complete problem.

This project aims to explore the relationship between compression and comprehension.

To view the code used in this project, visit `https://github.com/Guy29/FYP`.

1.1 Motivation

Occam's razor is a general principle in science and rationality commonly attributed to William of Ockham (1290?-1349?) which states "entities should not be multiplied beyond necessity", usually interpreted to mean "the simplest theory is usually the correct one."

It is easy to take this principle for granted, as it seems to work in both science and in our daily experience. But why this should be the case, i.e. why we live in a regular enough world that simpler models of it tend to be more correct than more complex ones, is not obvious, and it has been historically pointed out by many philosophers (most famously David Hume) that knowledge gained in this way stands on shaky foundations. (Henderson, 2018)

Solomonff's theory of inductive inference provides a formalism for Occam's razor. It posits an agent making observations in a world that operates by an unknown algorithm, and based on that premise shows that the agent would do well to assume that the length of the algorithm by which its world runs in effect follows a probability distribution that assigns shorter (and therefore simpler) algorithms more probability. This probability distribution is known as the universal prior.

The argument Solomonoff uses can be understood as follows: if the agent considers all algorithms of the same complexity as equally likely, then the algorithms can be divided into subsets where each subset is functionally equivalent. Because there are more ways to implement simpler algorithms than more complex ones, they accrue more probability mass. For example, a crime investigator who knows that Alice likes apple pie and Bob doesn't may consider the following hypotheses (of equal complexity) for the disappearance of an apple pie:

1. Alice stole it, wearing a red shirt.
2. Alice stole it, wearing a blue shirt.
3. Bob stole it, after having a change of taste.

If each of these hypotheses is given the same probability initially, based on being of equal complexity, then a grouping of the first two as functionally equivalent (the colour of the shirt being irrelevant) makes it twice as likely that Alice is the culprit.

Solomonoff's theory of inductive inference uses the concept of Kolomogorov complexity, which refers to the length of the shortest program that would produce a specific output. For example, the Kolomogorov complexity of the string "1111 ... 11111" is lower than that of "wp9j8 ... fd27c", as the first is more regular.

In the same way that Occam's razor lacked formalism and proof until Solomonoff, the concept of intelligence similarly lacks formalism in modern computer science, and psychologist R. J. Sternberg remarks "there seem to be almost as many definitions of intelligence as there were experts asked to define it." (Legg, Hutter, et al., 2007)

Marcus Hutter (Hutter, 2000) proposes a formalism of an intelligent agent which he terms AIXI that combines the above ideas as well as ideas from reinforcement learning. AIXI is a theoretical agent which, in each time step,

1. Makes an action a_i.
2. Receives an observation o_i and a reward (positive or negative) r_i.
3. Generates all possible algorithms by which its world can run which would have predicted all of its observations and rewards so far, and weighs the probabilities of those algorithms inversely to their length (i.e. it applies the universal prior).
4. Uses the most likely algorithms (or models of its world) to simulate the world, and thereby decide on its following actions to maximize its reward.

It can be seen that the above description of AIXI requires an agent that can effectively create a concise, compressed model of the observations it has made of its world so far, and that having such a model is most predictive of its success. For this reason, Hutter believes that intelligence, adaptability, and data compression are tightly bound.

1.2 Professional and Ethical Considerations

To the best of my knowledge, there are no professional or ethical considerations, as given by the BCS code of conduct (https://www.bcs.org/membership/become-a-member/bcs-code-of-conduct/) that would constrain any part of this project. The core of the project is a review and discussion of existing techniques, and practical applications will likely be limited to improvements on compression algorithms. All data used in this project is in the public domain.

1.3 Theory

1.3.1 Shannon

In his seminal 1948 paper "A Mathematical Theory of Communication", Claude Shannon introduced the concept of information theoretic entropy (Shannon, 1948), defining it as the average amount of information provided by a stochastic source of data, given by the equation

$$H(X) = \mathbb{E}[-\log p(X)] = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

for some discrete random variable X. In simple terms, the entropy of a variable is the average number of yes/no questions one needs to ask to determine the value of that variable. For a fair coin, one needs to ask only one yes/no question to learn the outcome, and so the entropy of that outcome is 1 bit (sometimes called 1 shannon).

For the English language, the entropy of individual letters based on their frequency is known to be approximately 4.14 bits. This can be obtained by calculating the negated sum of the products of the frequencies of the letters with the logs of those frequencies, in line with the formula above.

In his paper on the entropy of English text, Shannon improved on this, estimating that - when the letter is considered within its context - English contains 0.6 – 1.3 bits of information per letter (Shannon, 1951). He arrives at this estimate by asking subjects to reconstruct a text by guessing its letters one by one, only being told whether their guess is correct or not, and by noting how well

they perform at this task given a varying amount of context (1 to 100 previous letters). He finds that, when the subjects are given 100 previous letters for context, they are able to guess the next letter on their first try 80% of the time.

Shannon's analysis provides a theoretical best-case limit on how well natural language can be compressed.

1.3.2 Montague grammar

In 1970, logician Richard Montague posited that there is "no important theoretical difference between natural languages and the artificial languages of logicians" and that it is "possible to comprehend the syntax and semantics of both kinds of language within a single natural and mathematically precise theory." (Montague et al., 1970)

Montague gives a grammar which can be used to parse English, and a specification for a parser based on this grammar is given by Friedman & Warren (Friedman & Warren, 1978).

1.4 Practice

1.4.1 Tools

A variety of compression algorithms are in use today, many of which have been applied to natural language.

Mahoney (2011) has compiled a document which lists many compression program benchmarks on the `enwik9` challenge, along with the algorithms and techniques used by each, which is especially relevant to this topic. I touch briefly on each of these below.

Lempel Ziv and its variations

These include the original LZ77 (Ziv & Lempel, 1977) and LZ78 (Ziv & Lempel, 1978) algorithms.

LZ77

The first (LZ77) works by replacing repeated substrings inside a larger string with a reference to their first occurrence, indicated by a distance D from the current position at which that occurrence starts and the length L of characters that should be copied from that offset.

For example, the underlined part of the string "CC**BAABABBBAB**AABABBBABBABB" could be represented by (D=9, L=10), indicating that the beginning of the string "CCBAABABBBA" is followed by a 10-character long sequence which is copied from an offset that starts 9 characters back (indicated above in bold).

LZ78

The second algorithm (LZ78) works by incrementally building up a dictionary of previously seen substrings, each composed of a pair of (substring seen even earlier in the text, plus one additional character).

For example, the string "AABABBBABAABABBBABBABB" would be divided into the sections

A|A**B**|AB**B**|**B**|AB**A**|ABA**B**|B**B**|ABB**A**|BB

each of which is a previously seen substring plus one additional character (marked in bold). If the sections are numbered 1, 2, 3, etc., the whole string can be abbreviated to

A|1**B**|2**B**|0**B**|2**A**|5**B**|4**B**|3**A**|7

In this representation, 1B simply means "the contents of section 1, plus a B".

Symbol Ranking

The SR family of algorithms each keep a ranking of potential symbols, usually as a probability density function. For example, it could estimate a 0.1 probability that a randomly chosen letter is "e" and a 0.01 probability that it is "j", and consequently assign "e" a shorter code.

This can be done in a rolling fashion where the encoder will update the ranking of the symbols based on how frequent they have been in the text in a sliding window ending at the current symbol, and similarly the decoder will update its own ranking and codebook identically as it is decoding the text. In this case, the codebook keeps changing throughout the text, but both encoder and decoder can construct identical ones based on the text seen so far.

Prediction by Partial Matching

PPM is a SR-based algorithm augmented by conditioning the probabilities of symbols on their "context", consisting for example of the last n characters read.

If the algorithm has so far read the symbols "aard", for example, it may estimate a 0.9 probability of the next symbol being "v" and a 0.01 probability of the next symbol being "x". Based on these estimates, it assigns each of the characters "v" and "x" codes of lengths that correspond to how surprising they would be and will encode the actually observed character using that code.

In decompression, an identical predictor will read the encoded text, create the same codebook, and then use the recorded symbol code to reconstruct the text. (Fenwick, 1998)

Burrows Wheeler Transform

This algorithm, due to Michael Burrows and David Wheeler (Burrows, 1994) acts as an aid to other algorithms which perform better when fed data with runs of repeated characters. The algorithm outputs a reversible permutation of the characters of a string which contains more such runs.

The forward transform works as follows: first, an EOF character is appended to the string, then all circular shifts of the string are calculated and added to an array. This array is then sorted lexicographically, and the concatenation of the last character in each entry of the array is output.

Dynamic Markov Coding

Created by Cormack and Horspool (Cormack & Horspool, 1987). As with SR and PPM, this method of compression uses a predictor which assigns probabilities to potential next tokens, as well as arithmetic coding to assign shorter codes to more likely tokens. Unlike SR and PPM, this method compresses the input one bit at a time, rather than one byte at a time.

Context Mixing

Context mixing uses multiple predictors, each using a different context (or features of the text) and each producing its own probability distribution for the next symbol in its input, and combines these probability distributions through one of many possible averaging methods (e.g. linear, logistic) into a unified distribution which is often closer to the true distribution. This results in decoders which are less "surprised" by the next received symbol and which have accordingly assigned it a shorter code. To aid accuracy, CM is implemented in an "adaptive" way, meaning that the weights assigned to each sub-model's predictions are re-evaluated based on how accurate they've been so far. (Mahoney, 2005)

Long Short-Term Memory and Transformers

LSTMs are a type of recurrent neural network which is fed its previous output as one of its inputs and which maintains an internal state indicating which of its previously seen inputs are relevant to future outputs. Using the attention mechanism, transformers can similarly select relevant parts

of an input sequence as context for the part being encoded. Because of their ability to maintain context information, LSTMs and transformers are particularly suited for sequence prediction and so form the basis of another type of predictive coding.

1.4.2 Known results

In 2006, Hutter created a prize to "encourage development of intelligent compressors/programs as a path to AGI" (Hutter, 2006). The challenge is to compress 1GB of text extracted from Wikipedia, chosen because "Wikipedia is an extensive snapshot of Human Knowledge. If you can compress the first 1GB of Wikipedia better than your predecessors, your (de)compressor likely has to be smart(er)" and explaining that "while intelligence is a slippery concept, file sizes are hard numbers."

The prize is awarded based on improvements in data compression on a specific 1 GB text file, titled `enwik9`, which is extracted from the English Wikipedia.

The approaches taken by the record holders as well as the performance of their methods gives an indication of the current practical possibilities.

2. Investigation

2.1 Applying Shannon's methods

Shannon (1951) sets forth a method of analyzing the amount of information or uncertainty (i.e. entropy) each letter in an English text carries. In this section, I employ this method of analysis.

Shannon's approach in this paper builds on the concept of entropy as explained in 1.3.1. Rather than examining the entropy of individual letters out of context, he looks at the entropy of letters when the previous N-1 letters - the previous [N-1]-gram - is known. It formalizes the intuitive concept that there is more uncertainty when trying to guess a random letter in a text than there is when trying to guess a letter knowing its context.

To do this, he defines the sequences G and F, where

$$G_n = -\sum_{b \in B_n} p(b) \log p(b)$$

$$F_n = G_n - G_{n-1} = -\sum_{b \in B_n} p(b) \log p(b) + \sum_{b \in B_{n-1}} p(b) \log p(b)$$

where B_n is the set of all possible N-grams and $p(b)$ is the probability of a randomly chosen N-gram from an English text being equal to b.

G_n is simply the entropy of N-grams. For example, $G_1 = p('e')logp('e') + p('t')logp('t') + ...$ is the entropy of 1-grams, that is, single letters.

F_n is the differential or marginal entropy at the last letter of an N-gram, that is, how much uncertainty there is about that letter when the previous $n-1$ letters are known, and correspondingly how much information it carries.

To estimate these values for various values of n, I concatenate 97 texts obtained from Project Gutenberg and use python's `Counter` tool to count the number of occurrences of n-grams for various values of n. The results are shown in tables 2.1 and 2.2.

N	N-gram entropy	Marginal entropy
1	4.674789	4.674789
2	8.193527	3.518738
3	11.077931	2.884403
4	13.430718	2.352787
5	15.434333	2.003615
6	17.216262	1.781929
7	18.825973	1.609712
8	20.264030	1.438057
9	21.517189	1.253159
10	22.574182	1.056994

Table 2.1: Entropies and marginal entropies for substrings which are n letters long

Table 2.1 shows the entropies for substrings of the text which are n letters long, along with the entropy of the last letter, conditional on knowledge of the previous $n-1$ letters, while table 2.2 shows entropies for substrings which are n words long, along with the entropy of the last word.

As can be seen in both tables, more knowledge about the context in which a letter or word appears decreases the information content and increases the predictability of that letter or word.

N	N-word entropy	Marginal entropy
1	11.477039	11.477039
2	18.702746	7.225707
3	22.187337	3.484592
4	23.213621	1.026283
5	23.434336	0.220715
6	23.479111	0.044775
7	23.491305	0.012193
8	23.496988	0.005683
9	23.500703	0.003715
10	23.503611	0.002908

Table 2.2: Entropies and marginal entropies for substrings which are n words long

2.2 Benchmarking compression algorithms

For a rough bound on what's practically possible given common tools, I start by analyzing the performance of various existing non-text-specific compression algorithms on a collection of 97 texts in the public domain obtained from Project Gutenberg. The result of this benchmarking is displayed in table 2.3.

Algorithm	Compressed size (bytes)	Ratio
LZMA	20815738	3.745084
bzip2	21055590	3.702422
gzip	28702450	2.716029
zlib	28840353	2.703042
No compression	77956688	1.000000

Table 2.3: Compression algorithm benchmarks

The best performing compression algorithm is the Lempel–Ziv–Markov chain algorithm (LZMA), with a compression ratio of 3.745. Given that the vast majority of letters in Gutenberg texts are represented in a single byte (except for unicode characters), this compression method indicates that, at most, each byte in these texts contains on average $8/3.745 = 2.14$ bits of entropy. We can expect compression methods which exploit more of the structure of natural language to push this closer to the theoretical limit.

2.3 Co-compression

As previously outlined, the theme of this project is the relationship between compression and comprehension. One point of commonality between these is the need for parsimony. This is expressed by Occam's razor, which states that "entities must not be multiplied beyond necessity" and commonly understood to mean that "the simplest explanation is usually the best one".

Compression algorithms, for example the family of Lempel-Ziv algorithms, operate by eliminating the multiplication of entities through the creation of a codebook. If the relationship between compression and comprehension holds, we should expect to be able to use an existing compression algorithm like LZMA to get a rudimentary understanding of text.

Jiang et al. (2023) showed that it's possible to use gzip for text classification, based on

> "the intuitions that
>
> 1. compressors are good at capturing regularity;
> 2. objects from the same category share more regularity than those from different categories"

To illustrate this using the same 97 Gutenberg texts mentioned earlier, I use the following scoring methods to estimate the similarity of a pair of texts:

$$add(y|x) = \frac{C(xy) - C(x)}{C(y)}$$

where $C(t)$ is the compressed length of t expressed in bytes, xy is the concatenation of the texts x and y. The function $add(y|x)$ is then a measure of the additional information given by y when x is taken as a known basis.

To give an intuition for why the measure is defined in this way, it's helpful to use an example. Suppose that x is the text of the complete works of Shakespeare and that y is the text of Romeo and Juliet. Since x contains y as a substring, we should expect that a good compression algorithm would compress the concatenation xy in little more space than is required for just x, as the entire text of y can be signified with a symbol in the codebook, compressed once, and referred to twice.

Because of this, we should expect the value of $add(y|x)$ in this case to be close to 0, corresponding to the fact that y does not really add information to x. Conversely, since the complete works of Shakespeare contain a lot of information not contained in just Romeo and Juliet, we should expect $add(x|y)$ to be close to 1, indicating that our compression algorithm's knowledge of the text of Romeo and Juliet only makes a small dent in the number of additional bits it needs to represent the works of Shakespeare.

The measure used above contrasts with Normalized Compression Distance (NCD) as used by Li et al. (2004) , in that it is directional, i.e. that $add(x|y)$ is not necessarily equal to $add(y|x)$.

We may also define a more intuitive similarity score as follows

$$similarity(y|x) = 1 - add(y|x) = \frac{C(x) + C(y) - C(xy)}{C(y)}$$

This score giving a measure of the similarity of y to x. In this case, $similarity(a|b) = 1$ would indicate that a is fully described by b (as it contains no additional information), and $similarity(a|b) = 0$ indicates that a is unlike anything seen in b. Again, note that $similarity(y|x)$ is not necessarily equal to $similarity(x|y)$.

The similarity score is plotted for pairs of texts in figures 2.1 and 2.2, each plot giving x on the horizontal axis and y on the vertical axis, the color of the cell representing $similarity(x|y)$. In figure 2.1, the texts on the vertical axis are sorted by how predictive they are on average, and the texts on the horizontal axis are sorted by how predictable they are. In figure 2.2, the same texts are instead sorted by file size on both axes.

The figures were produced using the `seaborn`, and `matplotlib` python libraries, and `pandas` was used to process the data.

The following observations can be made:

- In general, larger texts are more predictive.
- In general, smaller texts are more predictable.
- There are a few very bright dots which do not lie along the diagonal in 2.2. These appear because the dataset includes three versions of Frankenstein which have a high similarity score with each other, as well as the texts for Romeo and Juliet, and the Complete Works of Shakespeare. For this last pair, the latter strongly predicts the first but the first only weakly predicts the second.
- There is a strong horizontal dark line in both graphs, which represents a text that is unpredictable regardless of what other text it is paired with. On inspection, this turns out to be Don Quijote, which is in fact unlike the other texts because it is in Spanish, whereas most other texts are in English.
- As can be seen in 2.1, when texts are sorted most-predictive-first on the y-axis and most-predictable-first on the x-axis, texts generally lie along the diagonal. This indicates that,

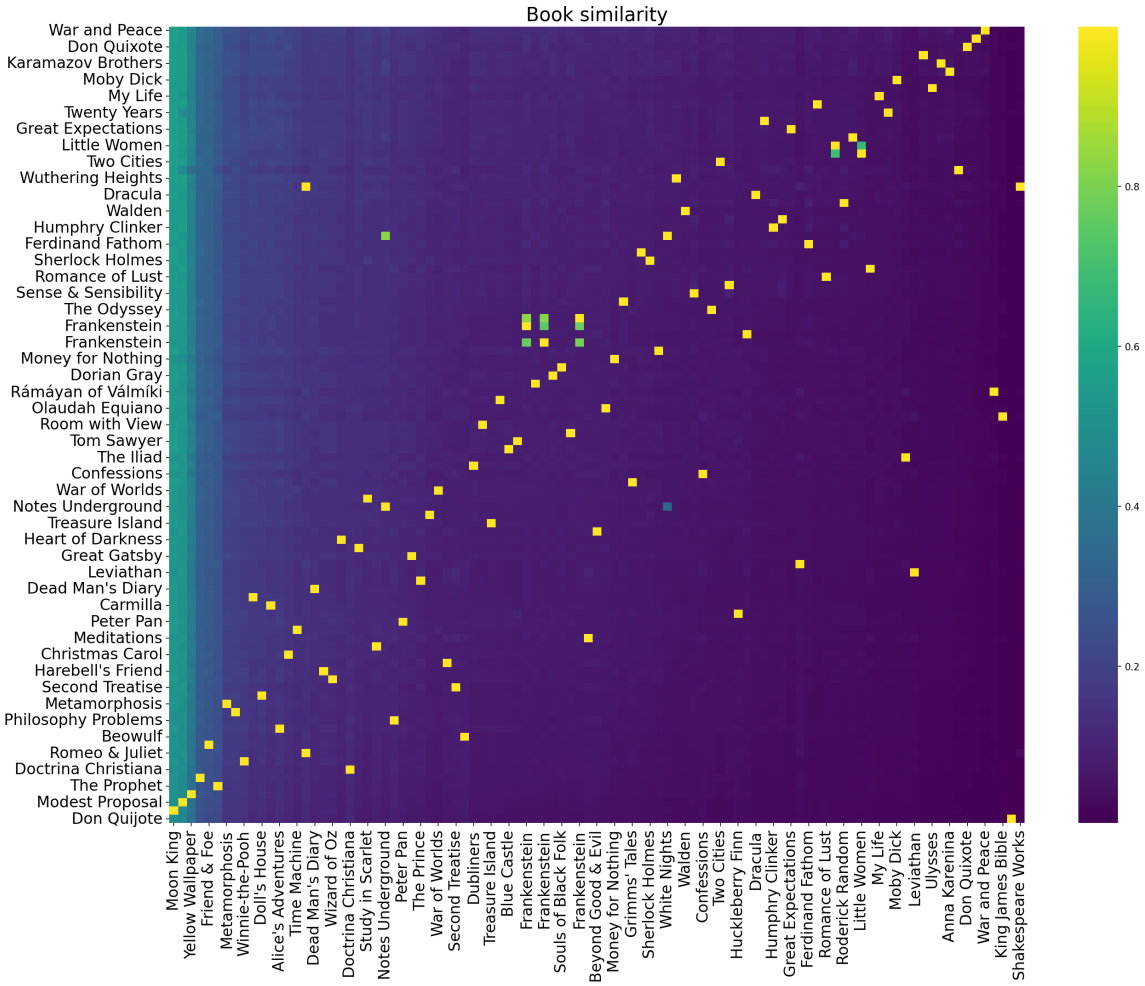

Figure 2.1: Similarity scores for pairs of texts. The texts on the vertical axis are ordered most-predictive-first, and those on the horizontal axis are ordered most-predictable-first.

within a set of texts, there is a trade-off between how predictive a text is and how predictable it is. At least part of this effect has to do with file size, as larger texts tend to contain a larger set of the possible words and expressions of a language.

- There is a visible pattern of strong vertical and horizontal lines in 2.2 (i.e. there is a low level of noise), indicating that texts tend to be "generally predictive" or "generally predictable" within a given collection.

2.3.1 Practical application

As can be seen in the previous figures, it is always the case for any two pairs of texts x and y that "the whole is less than the sum of its parts", i.e. that $C(xy) < C(x) + C(y)$. This opens up the question of whether it is possible to obtain a better compression of y by encoding only that information which it adds to x. If this is possible, we should expect that this information should be of the size $C(xy) - C(x)$ which, by the previous equation, is necessarily less than $C(y)$.

Intuitively, this means that if two parties Alice and Bob each have a copy of the works of Shakespeare, and the Alice wants to send over Romeo and Juliet, she can simply encode it by giving the range of pages on which it appears in their shared reference text.

Is this possible to implement in practice? The fact that Lempel-Ziv algorithms work by going through the data in one pass and creating a codebook as they go indicates that $D(xy)$ should contain $D(x)$ as a prefix, where $D(t)$ stands for the compression of t. This is, in fact, more or less the case with LZMA, and I was able to create a simple tool that implements this idea, accessible at

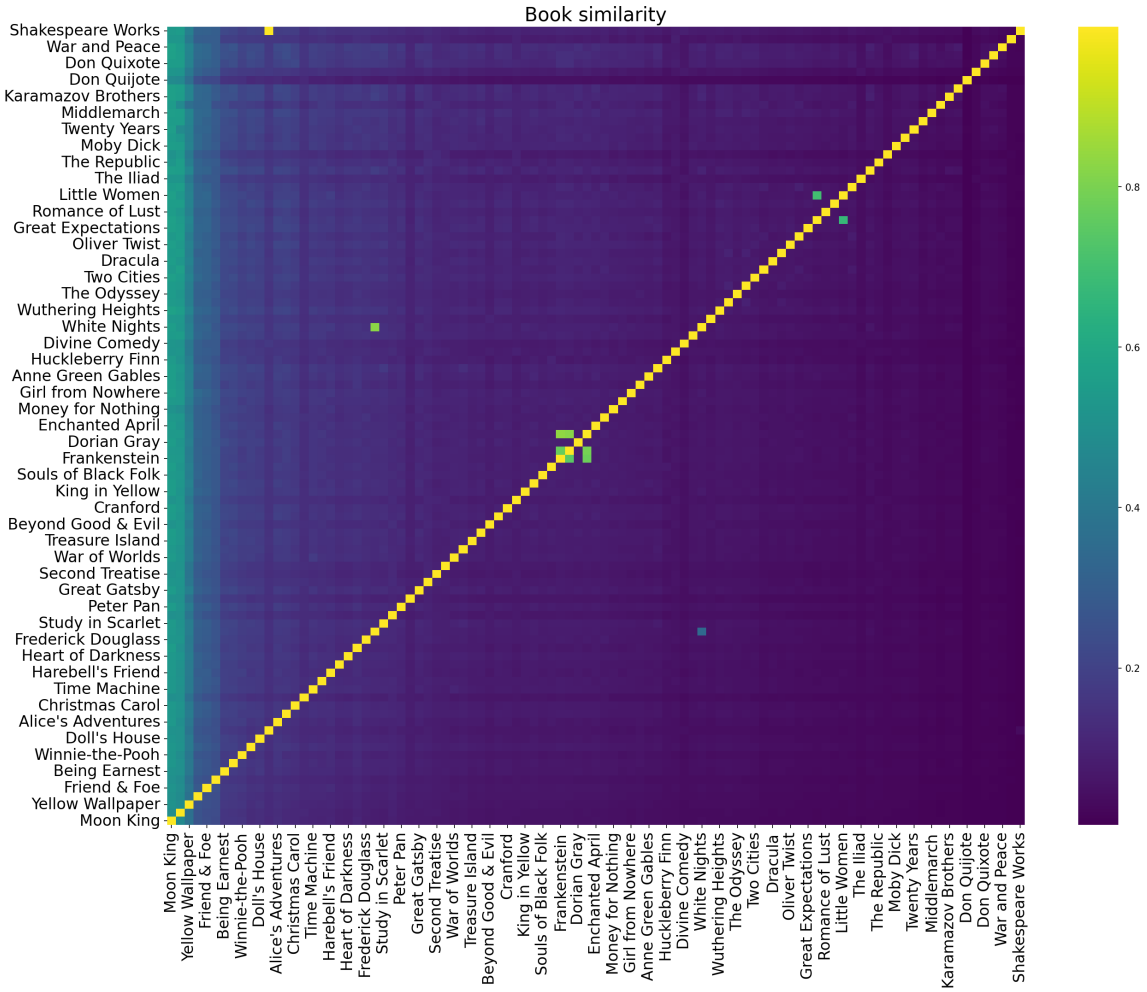

Figure 2.2: Similarity scores for pairs of texts. The texts on both axes are sorted by file size.

`https://github.com/Guy29/FYP`.

The `CoCompressor` class I implement is instantiated with two arguments: a reference text r and a compression algorithm. Its `compress` method takes a text t to compress will output $D(rt)$ with the prefix $D(r)$ removed, and its `decompress` method takes a text compressed in this way and reverses the process.

Co-compressors instantiated this way differ in their power depending on the choice of reference text. See figure 2.3 below for a comparison of the performance of co-compressors trained on different reference texts.

There are a few remarkable things about this figure,

- Each co-compressor shows an tan- or sigmoid-like curve, performing exceptionally well on texts which are very similar to it (left side of the plot) and steeply dropping down in performance for sufficiently different texts.
- There is very little criss-crossing between the lines in this plot. That is, for any pair of co-compressors, one will usually outperform the other *on every output* in the dataset. Why there should be a strict hierarchy of this sort is not clear, as one might expect that a specific co-compressor would be specialized for one sort of text which are similar to it but not for others.
- The best performing co-compressor in this dataset is the one trained on Middlemarch (performing 12% better than naive LZMA in the median case), followed by A Tale of Two Cities (7.3%) and The Romance of Lust (7.2%).

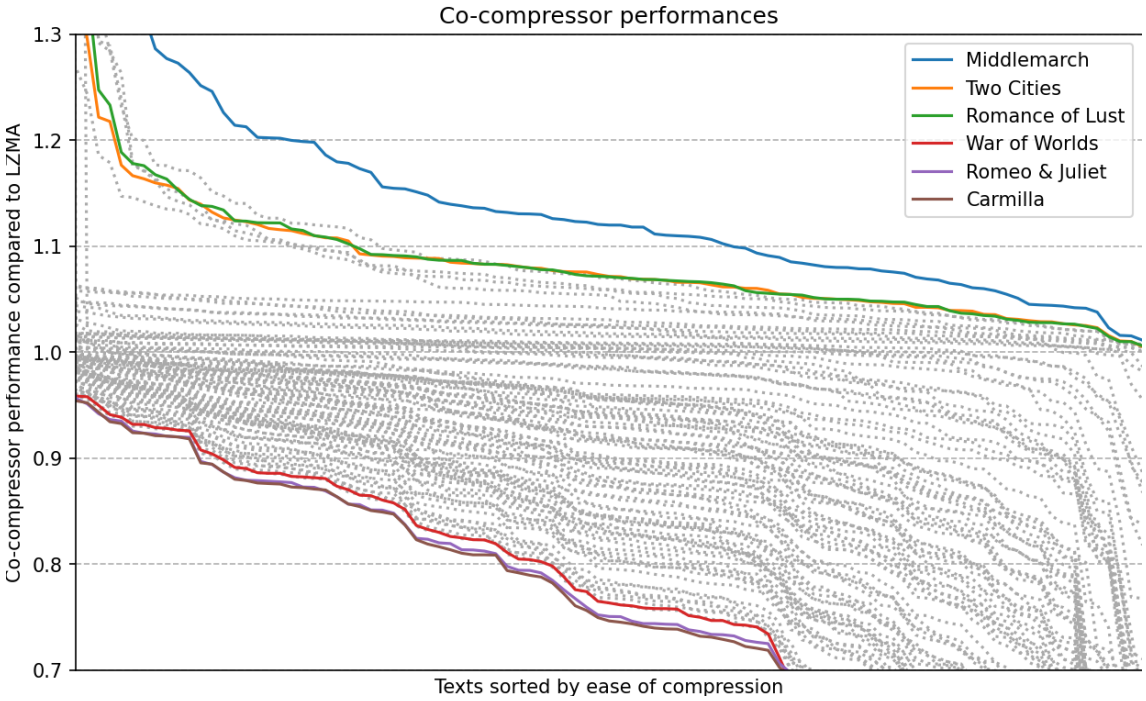

Figure 2.3: The performance of instances of the `CoCompressor` class trained on 97 different texts. Each line represents a `CoCompressor`. The value on the vertical axis is the ratio of the size of the encoding produced by naive LZMA to that produced by the `CoCompressor`, while the horizontal axis goes through possible inputs for comrpession.

It is tempting to explain away the unusual performance of the Middlemarch co-compressor (CC_{MM}) as an artifact of the chosen dataset, that it might be situated (historically or otherwise) "in the middle" of the dataset, sharing some features with both preceding and following literary works.

One strong piece of evidence against this is that, as noted above, the performance of co-compressors is strictly hierarchical, and this effect extends all the way to the left of the plot where the co-compressor is fed the text it performs best on (that is, its own text) as its input. Examining this, we notice that

$$CC_{MM}(MM) > CC_t(t) \qquad \forall t \in G$$

where $CC_t(x)$ is the performance of the co-compressor trained on text t when fed input x, and G is the dataset. That is, CC_{MM} is not only the most performant co-compressor on all the texts in the dataset, it is also the co-compressor that performs best on its own reference text.

This observation gives a simple way to find other good reference texts for co-compressors: one simply calculates $CC_t(t)$ for each candidate and finds values of t that yield the highest performance.

2.4 Natural language compilation

2.5 LLM-based compression

Bibliography

Burrows, M. (1994) A block-sorting lossless data compression algorithm. *SRS Research Report*. 124.

Cormack, G. V. & Horspool, R. N. S. (1987) Data compression using dynamic Markov modelling. *The Computer Journal*. 30 (6), 541–550.

Fenwick, P. (1998) Symbol ranking text compressors: review and implementation. *Software: Practice and Experience*. 28 (5), 547–559.

Friedman, J. & Warren, D. S. (1978) A parsing method for Montague grammars. *Linguistics and Philosophy*. 2 (3), 347–372.

Henderson, L. (2018) The problem of induction.

Hutter, M. (2006) 500'000€ Prize for Compressing Human Knowledge by Marcus Hutter. Available from: http://prize.hutter1.net/

— (2000) A theory of universal artificial intelligence based on algorithmic complexity. *arXiv preprint cs/0004001*.

Jiang, Z., Yang, M., Tsirlin, M., Tang, R., Dai, Y. & Lin, J. (2023) "Low-Resource" Text Classification: A Parameter-Free Classification Method with Compressors, 6810–6828.

Legg, S., Hutter, M., et al. (2007) A collection of definitions of intelligence. *Frontiers in Artificial Intelligence and applications*. 157, 17.

Li, M., Chen, X., Li, X., Ma, B. & Vitányi, P. M. (2004) The similarity metric. *IEEE transactions on Information Theory*. 50 (12), 3250–3264.

Mahoney, M. (2005) Adaptive weighing of context models for lossless data compression.

— (2011) Large text compression benchmark.

Montague, R. et al. (1970) Universal grammar. *1974*, 222–246.

Shannon, C. E. (1951) Prediction and entropy of printed English. *Bell system technical journal*. 30 (1), 50–64.

Shannon, C. E. (1948) A mathematical theory of communication. *The Bell system technical journal*. 27 (3), 379–423.

Ziv, J. & Lempel, A. (1977) A universal algorithm for sequential data compression. *IEEE Transactions on information theory*. 23 (3), 337–343.

— (1978) Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*. 24 (5), 530–536.