Universitetet i Bergen

INF144

Informasjonsteori

Assignment 1

Author:
Oskar Leirvåg
OLE006

March 28, 2018

Preface:

I wrote the entire program in Java 9 so standard compiler should let you build and use. Also I really like unit testing so you might need to add Junit4 and hamcrest core if you want to run those, but they are independent to the result.

The classes hold their perspective functionalities. Everything from Huffman is inside the Huffman class, everything for Markov is inside the Markov class. And so on...

Markov:

If we generate the Markov in zero-th order, we should not be able to see any words. That would just be dumb luck. And the same counts for the first order as well, and there are really not that many words containing only 2 letter. But once we generate the second and third order we begin to see really clearly some of the words from the text. Using a larger file and 10th order we could have grammatically correct sentences generated. Rather funny actually.

Now if i had actually READ the entire task before solving it, I would probably choose another approach using 4-D arrays instead of hash-mapping. Since the entropy computation does require the ω_k, we need to solve linear equations preferably using a matrix. Which I did not have, and all my attempts to solve it in Java rendered me a ton of DNE. which I don't think is correct.

What i can say about the models is that from the way we define them they HAVE to be unifilar. We build the model / source so that if we detect another instance we do not add that ass a new state but rather modify the probability / weight for that state to happen. Therefore each state will only produce unique outputs of their functions.

Compression:

Compression is probably my favourite topic of algorithms as it is so genius and so useful in simple everyday thing.

The theoretical file-size is calculated like it was UTF-8 wich stores ASCII as 8-bit and everuthing else as 16 bit. If the string was saved properly in binary the bits would just be bits so we count them only as 1bit.

LZW:

By the way LZW works it will compress patterns in a string, which is the predictability we exploited with the creation of the markov-chains. This means that a "sentence" generated by the markov-chain should be compressible to a large degree.

Compressing the short story we get:

$$S_{old} = 65568bit, \quad S_{new} = 28912bit$$

$$Ratio = 100\frac{S_{new}}{S_{old}} \approx 43\%$$

Huffman:

Now as a rule there is no point in compressing a compressed file. In most cases the resulting file can get larger. In these examples this happens because in the Huffman tree we need to place both 1 and 0, but ALSO every index. that means that either 1 or 0 will need to be atleast 2 bits, that's DOUBLE the required amount. So now lets test this.

Instead of the new file being 43% of the original size, it is now 62% which is larger, just as predicted.

More testing:

Now if we do this a hundred times and average, comparing Huffman, LZW and LZW + Huffman we can see the pattern repeating. I did not remove the parenthesis nor commas but the result should be roughly the same. But this is going to take quite the time. Producing the results:

- LZW avg: 34.791556650590884
- Huffman avg: 50.08600068181691
- LZW + Huffman avg: 52.038648660885755

Which makes sense! The LZW is much better at compressing these generated files as they have many patterns. The Huffman is ok because it compresses the UTF file-size, but the absolute worst result is the double compression. Which just ruins the LZW.