Assignment 5 Writeup

Sean Siddens

CSE13s

Spring 2021

Entropy of Hamming Codes

In information theory, the amount of entropy in a message is defined as

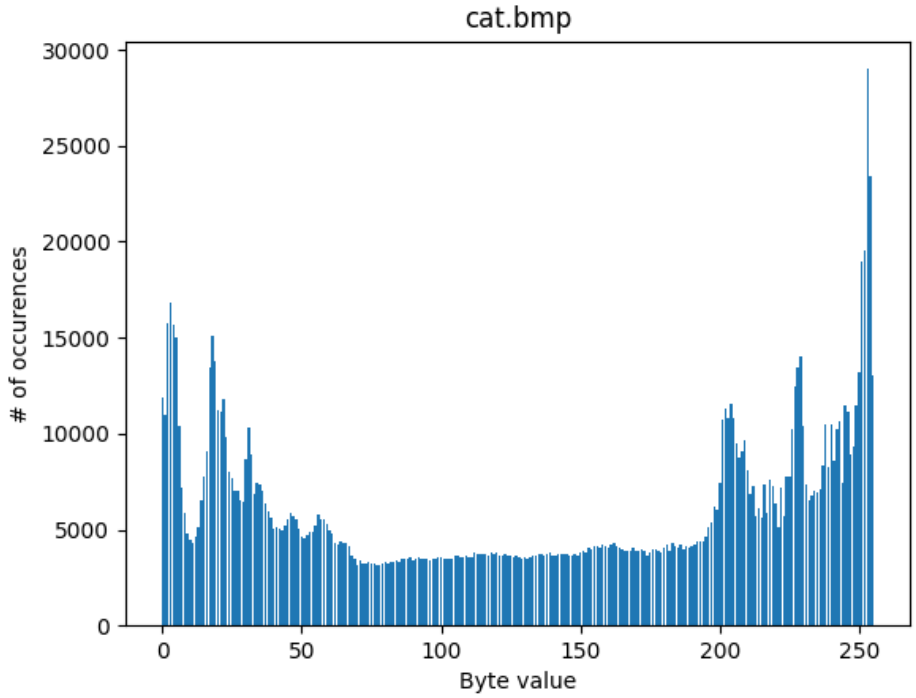
$$\mathrm{H}(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log \mathrm{P}(x_i)$$

Where $P(x_i)$ is the probability of some outcome of a given variable for occurring. In a file, the variable is a byte, which can take on 256 possible states. You can get the probability of that byte occurring in the file by counting the number of times it shows up, and dividing it by the total number of bytes in a file.

For example, the entropy of this image:

is 7.79 bits per byte. The byte values of the image can take on many different values, so the probability of any one such value occurring is quite low, increasing the total entropy of the image.

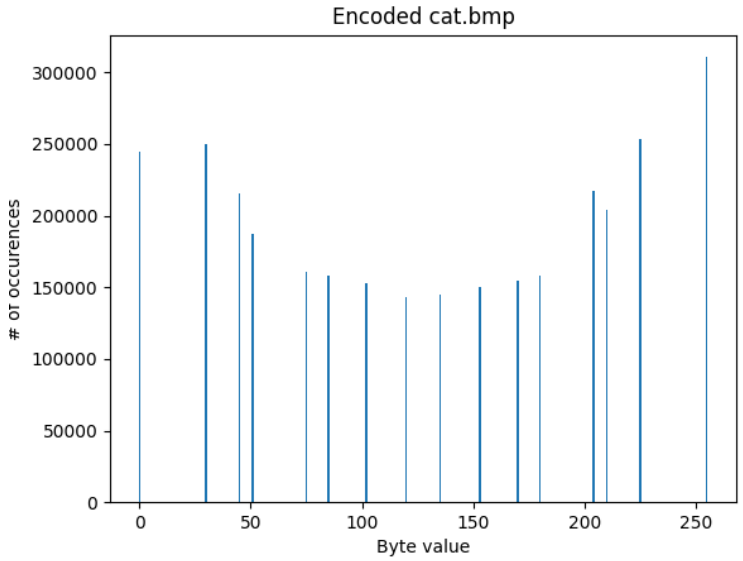
We can visualize this entropy by plotting the number of occurrences of each byte value in the image:



The wide distribution of values corresponds to a large entropy in the file.

If we encode the image data to (8, 4) systematic Hamming codes, the entropy decreases significantly to 3.956730 bits per byte.

Plotting the byte counts of the Hamming codes demonstrates this lower entropy:



Even though the encoding process doubles the total amount of data, the range of possible values that the Hamming codes can take on is much smaller. The reason this is the case is because the parity of a valid Hamming code is always even. The parity bits are always flipped in order to ensure this even parity scheme, so the vector space of our Hamming codes is cut at least in half.

Entropy of Noisy Hamming Codes

If we encode a file, such as a text file only containing letters of the alphabet, and inject noise at a specified rate into the encoded data, the entropy will change depending on the rate of the noise:

Error Rate	Entropy
0.00	3.131353
0.01	3.768161
0.05	5.293142
0.10	6.412802
0.25	7.696790
0.40	7.967438
0.50	7.99068
0.60	7.965681
0.75	7.696452
0.90	6.425497
0.95	5.301896
0.99	3.774587
1.00	3.131353

The entropy roughly follows a bell curve corresponding to the error rate. As you increase the error rate, the entropy quickly increases due to the even parity scheme of the Hamming codes being disrupted. This entropy maxes out when the error rate is 50%. However, after that point, the entropy begins to decrease as the noise increases, and in fact the entropy of the file with 100% noise is the same as the file with 0% noise.

When the error rate is 50%, one in two bits are flipped. Essentially, we are going from our Hamming code space, where the parity is always even, to a space where the parity of the byte can be anything. This is why the entropy nears 8 bits per byte, which essentially means the bytes take on completely random values across the entire byte space from 0 to 256. When the error rate is 100%, every single bit is flipped, which means that the parity is preserved, and is why the entropy stays the same.

Incorrect Decoding

With our encoding scheme, we can detect up to two errors and correct only one. Essentially, we can detect an error has occurred in a given byte if the byte is within a Hamming distance of two of the original Hamming code. Our Hamming codes are a subset of $\mathbf{F}_2^8$, which is the vector space with length 8 taking on values over the binary finite field. The Hamming distance between two vectors in this space is simply the number of differences between each component of the vectors. For example, if our Hamming code is (01111000), then the set of all vectors with a Hamming distance of one from our code is

((11111000), (00111000), (01011000), (01101000), (01110000), (01111100), (01111010), (01111001))

When we are decoding, if the byte read in is one of these vectors, we can detect where the difference occurs, and simply flip that bit in order to get back to the original Hamming code. This is why the Hamming distance for correcting an error is one. For example, if the bit vector we read in while decoding was (01011000), our decoded error syndrome would be (1101), which corresponds to the 3rd row of the transpose of the parity checker matrix, which means flipping the third bit will get us our original Hamming code: (01111000).

If the Hamming distance of the read in bit vector to our original code is two, then it will be outside of this set of vectors with a distance of one, and we will know that we cannot correct it. This is why we can detect the occurrence of two errors, but we can't fix it.

However, an issue arises when there are three errors. For example, if our original Hamming code was (01111000), but during transmission the last three bits were flipped due to noise (01111111), then the decoded error syndrome would be (0111). This syndrome corresponds to the first row of the matrix, and will therefore be recognized as a code that can be corrected. However, simply flipping the first bit won't get us back to our original code. The reason why this occurs is because the minimum distance between valid codes in our space is 3. Every single Hamming code has a "ball" surrounding it containing all of the vectors with a Hamming distance of one (a single bit flipped). When we flip three bits in our Hamming code, we move outside of

our “ball” and into a different one. In fact, any odd number of bit flips will result in this issue, where we decode and correct back to a different but valid Hamming code from the original. The set of valid codes and their corresponding balls with radius 1 are disjoint over our space, which is why an even number of errors will place us in the complement of this set, and we’ll be able to recognize an error has occurred. When an odd number of errors occurs, it places us in the ball surrounding a valid Hamming code, so not only can we not detect that an error has occurred, but we believe we successfully corrected the error (when in reality we end up with a totally different Hamming code).

If we believe that noise during transmission/storage will occur in bursts, then one way to prevent uncorrectable errors and/or incorrect decoding would be to interleave the Hamming codes, so that the errors which occur during a noise burst would distribute to many code blocks.