Applied Information Theory 650

Assignment 6

1 Intro

This assignment explores Huffman source coding, Hamming channel coding, and how they interact when they are used together. You can form teams of up to 3 people in doing this assignment, or you can do it on your own, whichever you prefer. If you do it in a team of 2 or 3, you only need to turn in 1 write-up for the whole team, but make sure you include the following items:

1. The names of each person on the team.
2. The contribution made by each person on the team.

If a person made no contribution to the team, they will not get credit for the assignment. I realize it is not possible to balance the parts exactly, but try to split up the chores as evenly as possible among team members.

2 The assignment

Select a text file containing English prose. You can use anything you want, but it should be at least 10,000 characters. You can use public domain novels from Project Gutenberg, or any other source you want. If you want to preprocess your text to remove difficult to process characters (like dashes or quotes), that is fine, but make sure you have at least 32 different characters (you can have as many different characters as you like).

1. Calculate the frequency of every character in your text, and show it in a table. Don't forget to include spaces and punctuation.
2. Build a Huffman code for the text you have chosen, based on the frequencies you calculated. Do not use code you can find on the web. This should be your own implementation. To do this, you will need to write two functions. One is HuffmanEncode(string) which takes as input a string and returns the Huffman encoding of that string as a binary string. (NOTE: you do not have to "pack" the bits into bytes. The binary string returned can use one matlab integer or floating point number for each bit, which is not efficient spacewise, but shows the number of bits you can actually compress to.) The second function is HuffmanDecode(string) which takes as input a binary string and decodes the string. This function should report an error if, at the end of decoding, only a "partial code" remains. For example, if the last portion of the string to be decoded is a "0", but there is no character whose encoding is a "0", then an error should be reported.
3. Let the original document length be computed as $N \times \log_2 M$, where M is the number of allowable characters and N is the number of characters in the document. Compress the document using the Huffman code. What compression factor did you get?
4. Build a (15,11) Hamming code as described on the Wikipedia page for Hamming codes. This uses 11 data bits and 4 parity bits per use of the code. Write a function HammingEncode(bitstring) that takes a sequence of 11 bits and returns the 15-bit codeword to be sent over the channel. You will also need a function HammingDecode(bitstring) that takes a 15 bit number (the output of the channel) and returns a guess at the 11 bit number originally sent.

5. You will use your Hamming code to communicate over a Binary Symmetric Channel. You will simulate the channel by randomly flipping each bit in each codeword with a probability $\alpha = 0.02$. Answer the following questions about your Hamming code and the BSC:

   (a) What percentage of bits do you expect to be erroneous?

   (b) What percentage of codewords do you expect to have exactly one error?

   (c) What percentage of codewords do you expect to have exactly two errors?

   (d) What percentage of codewords do you expect to have greater than two errors?

   Write a function called BSC(alpha, bitstring), which takes as input a bitstring to send over a binary symmetric channel, and simulates flipping bits at a rate α. It then returns the corrupted string.

6. In this part, you will use your Hamming code to try to correct errors sent over the BSC. First, you will send your *uncompressed* book. To do this, you need to convert every character in your book to a binary string, by replacing each character with its 8-bit ASCII code. You can look this up if you don't know how to do it. Once you have converted your book into a giant binary string, break it up into 11 bit chunks, use your HammingEncode function to turn this into a 15 bit string with error correction, and send one 15-bit chunk at a time over the noisy BSC described above. Use your HammingDecode function to try to correct errors on the other side. (If you detect more than one error, you can just make a guess at the original string, since you will not know it for sure.) Once you have your corrected string at the output, convert it back into ASCII characters, and figure out how many errors were left. Please report all of the following:

   (a) How many characters was your document?

   (b) How long was the binary version of your document in bits (should be 8 times the number of characters).

   (c) How many bitwise errors were introduced by the channel?

   (d) How many 15-bit codewords had zero errors?

   (e) How many had exactly one error?

   (f) How many had exactly two errors?

   (g) How many had more than two errors?

   (h) How many bit errors were left once you decoded the bit string using HammingDecode?

   (i) How many codeword errors were there, meaning, for each 11-bit string, how many of them were wrong after error correction?

   (j) How many actual characters did you get wrong?

7. Finally, you will try sending your Huffman compressed document over the BSC, error correcting it using Hamming decoding. Unlike the case in question 6, you may find that the decoded document is not the same length as the original document. This will make it much more challenging to find the number of errors in the decoded document. Instead of finding the number of errors, look at the decoded document and find the first place it does not agree with the original document. What happens? What is the overall quality of the transmitted document? Explain why this happens.