COMPSCI 650 Applied Information Theory March 31, 2016

Lecture 17

Instructor: Arya Mazumdar *Scribe: Names Redacted*

1 Exam Review

Everyone did well on the first problem (on Huffman coding), so we did not go over it in class.

2. For the second problem, you are given a fair 6-sided die. If the die comes up 2, 3, 4, or 5, the payoff is 1, otherwise if it comes up 1 or 6 the payoff is -1.

   - In the first part, calculate E[gain] $= \frac{4}{6} \cdot 1 + \frac{2}{6} \cdot -1 = \frac{1}{3}$.
   - In the second part, you must find a probability distribution on $\{1, 2, 3, 4, 5, 6\}$ that maximizes entropy while maintaining a payoff of at least T. Since the payoff for 2, 3, 4, and 5 is the same, and the payoff for 1 and 6 is the same, observe that we can maximize entropy by setting the probability for values with the same payoff to be equal, since the uniform distribution maximizes entropy. Then we have $p(2) = p(3) = p(4) = p(5) = p$, and compute $p(1) = p(6) = \frac{1-4p}{2}$.

     Then E[gain] $= 2 \cdot \frac{1-4p}{2} \cdot (-1) + 4p \cdot 1 = 8p - 1$, so let $T = 8p - 1$, and finally calculate $p = \frac{T+1}{8}$, then substitute to obtain the final probability distribution.

3. In the third problem we are given a family of distributions $f(x; \theta) = \dfrac{1}{\sqrt{2\pi\theta}} \cdot e^{\frac{-x^2}{2\theta}}$.

   - In the first part, compute the score function:

   $$V(x) = \frac{\partial}{\partial \theta} \ln f(x; \theta) = \frac{x^2 - \theta}{2\theta^2}.$$

   - In the second part, compute the Fisher Information:

   $$J(\theta) = \mathrm{E}[V(x)^2] = \int_{-\infty}^{\infty} f(x; \theta) V(x)^2 \, dx = \int_{-\infty}^{\infty} (x^4 - 2\theta x^2 + \theta^2) f(x; \theta) \, dx = \frac{3\theta^2 - 2\theta^2 + \theta^2}{4\theta^4} = \frac{1}{2\theta^2}.$$

   - In the third part, the Cramér-Rao bound tells us MSE $\geq \frac{1}{J(\theta)} = 2\theta^2$.
   - For the fourth part we are asked to give an example of an unbiased estimator. There are multiple things that would work, but a good example is to take the square of the average difference between x_i and all other samples, then average that over all x_i to obtain

   $$\frac{1}{n} \sum_{i=1}^{n} (x_i - \frac{1}{n} \sum_{j=1}^{n} x_j)^2.$$

     Anything similar to this got full credit, but notice that the above estimator actually is not unbiased; to make it unbiased we must replace the outermost $\frac{1}{n}$ by $\frac{1}{n-1}$.

4. In the fourth problem, we are trying to distinguish between two very biased coins, one of which has $p_1(H) = \epsilon$, the other with $p_2(H) = 1 - \epsilon$. Though not specifically asked for in the question, an estimator for this after m flips would be to compare the observed number of heads to $m\epsilon$ and $m(1-\epsilon)$, and predict whichever is closer. Also, to compute P_e exactly we know $P_e = e^{-mD(1/2||\epsilon)}$.

- In the first part we are asked for a lower bound on P_e using Le Cam's identity and Pinsker's inequality. Combining the two, we have

$$\begin{aligned} P_e &= \frac{1}{2}(1 - ||P_1^{(m)} - P_2^{(m)}||_{TV}) \\ &\geq \frac{1}{2}(1 - \sqrt{\frac{(\ln 2) \cdot D(P_1^{(m)}||P_2^{(m)})}{2}}) \\ &= \frac{1}{2}(1 - \sqrt{\frac{\ln 2}{2} m D(\epsilon||1-\epsilon)}), \end{aligned}$$

  where the first equality is Le Cam's identity and the following line uses Pinsker's inequality. We have also

$$D(\epsilon||1-\epsilon) = \epsilon \log \frac{\epsilon}{1-\epsilon} + (1-\epsilon) \log \frac{1-\epsilon}{\epsilon} = (1-2\epsilon) \log \frac{1-\epsilon}{\epsilon},$$

  so combining this with the above we get

$$P_e \geq \frac{1}{2}(1 - \sqrt{m \frac{\ln 2}{2} (1-2\epsilon) \log \frac{1-\epsilon}{\epsilon}}.$$

- We want $\frac{1}{100} \geq P_e$, so using the bound from the first part, we have

$$\begin{aligned} \frac{1}{50} &\geq 1 - \sqrt{m \frac{\ln 2}{2} (1-2\epsilon) \log \frac{1-\epsilon}{\epsilon}}, && \text{so} \\ \sqrt{m \frac{1}{2} (1-2\epsilon) \ln \frac{1-\epsilon}{\epsilon}} &\leq \frac{49}{50}, && \text{so} \\ m &\geq \frac{2(\frac{49}{50})^2}{(1-2\epsilon) \ln \frac{1-\epsilon}{\epsilon}}. \end{aligned}$$

2 Coding Theory Review

The general setting for which we are considering error correcting codes is that we have a channel through which we are transmitting data, which sometimes corrupts or erases the data. We are interested only in memoryless channels, where the probability of a flip or erasure is independent for each transmission, and have so far restricted ourselves to binary channels, where only single bits are transmitted. In the Binary Symmetric Channel (BSC) each bit is flipped with probability p, and in the Binary Erasure Channel (BEC) each bit is erased (replaced by a new symbol "?") with probability p.

The goal of error-correcting codes is to encode some extra data along with the data we actually care about, so that if some amount of the data is corrupted by the channel, the correct message can still be decoded. Depending on how we encode our data, this decoding process can be very expensive.

In general, a code is just a set of vectors, here called codewords, over $\{0,1\}^n$. We say such a code is an (n, M) code, where M is the number of vectors in the code. We are especially interested in a special type of code, linear codes, because they are much easier to work with (especially for decoding) than general codes. A linear code is a set of vectors over $\{0,1\}^n$ where $M = 2^k$ for some $k \leq n$, and these codewords form a vector space under the operations of addition and multiplication mod 2 (binary XOR and AND, respectively). We say such a code is a linear $[n, k, d]$ code, where d is the minimum Hamming distance between any two distinct codewords. As we saw last class, when the vectors are binary the minimum

distance between any two distinct codewords is equal to the minimum weight (number of nonzero entries) of any nonzero codeword.

Because the codewords form a vector space, linear codes have several very nice properties. We can represent the entire code by a generator matrix G, of dimension $k \times n$. Then we can encode any message (recall messages are binary vectors of length k) by simply multiplying the message by G, yielding a codeword of length n. Any k linearly independent codewords form a generating matrix for the code, so this matrix is not unique in general.

We can also represent the code by its parity check matrix H, of dimension $n \times n - k$. Each row of the parity check matrix corresponds to a linear equation (mod 2) among the entries of a codeword which must be equal to 0. Thus a length n vector is a codeword if and only if its product with the parity check matrix is the zero vector. As we will see in the next section, the parity check matrix can be useful for decoding certain linear codes.

3 Hamming Code

One example of a linear code is the Hamming Code. The below example is of a [7,4,3] Hamming Code, which encodes 4-bit messages into 7-bit codewords. This code has the parity-check matrix

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

3.1 Encoding

Each codeword must satisfy the following system of equations (mod 2):

$$x_4 + x_5 + x_6 + x_7 = 0 \quad (1)$$
$$x_2 + x_3 + x_6 + x_7 = 0 \quad (2)$$
$$x_1 + x_3 + x_5 + x_7 = 0 \quad (3)$$

Are the following potential codewords actually codewords?

- 0010110 - Yes.
- 1011010 - Yes.
- 1001100 - Yes. (Note that this codeword is the sum mod 2 of the previous two codewords)

To build a generator matrix from a parity check matrix, one must find four linearly independent code words. The following is one (of several possible) generator matrices for the Hamming code:

$$G = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

Note here that each of the rows of the generator matrix G is a codeword, and the minimum weight of any codeword is 3. That means that we can correct 1 error in this code, because any vector that has distance 1 from a codeword will be closer to that codeword than any other codeword.

3.2 Decoding

When decoding a message, one approach is to compare the received transmission with each codeword, and decode to the codeword with the minimal Hamming distance. This approach is slow (2^n possible codewords, where n is the bit-length of our original message).

There is a more efficient approach, however. If we multiply the parity-check matrix H by our received message, we can identify the bit that has been flipped. Consider the following example: we transmit the codeword 1001100, but the channel distorts the message and the decoder receives the codeword 1000100. Let's multiply H (from above) by the message that we received:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

If we convert the result from binary, we get $100 = 4$. This tells us that the 4th bit in the transmitted message has been flipped, which is consistent with what we saw above.

What if the 6th bit had been flipped in the above example, e.g. the message we received was 1001110? Using the same calculation as above,

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}.$$

Converting this result shows that $110 = 6$, which corresponds to the bit that had been flipped in the transmission. So, why does this work as a method to identify the flipped bit?

Let's consider the composition of the message we received. The received message, y, consists of the encoding of our original message x (7-bit vector) and one flipped bit e, where e is also a 7-bit vector, but with only one 1 in the position of the flipped bit. So we have that $y = x + e \pmod 2$. We also know that $H(x+e) = H(x) + H(e)$ by the linearity of the parity check matrix. By definition of H, $H(x) = 0$, so we have that $H(x+e) = H(e)$. Looking back at H, we can see that the columns of H are ordered by binary representation, so the result of $H(e)$ is the location of the flipped bit.

More generally, if H has m rows, then it can have 2^{m-1} unique columns, and can correct 1 error. So we can define an $[n, k, d]$ Hamming code where $n = 2^m - 1$ is the codeword length, $k = 2^m - 1 - m$ is the message length, and $d = 3$ is the minimum weight of the codewords.

4 Random Linear Codes Achieve Capacity

In order to correct more than one error, the combination of any two columns in H must be unique. Then $H(e)$ will correspond to a unique combination of columns indicating the bits that are in error.

One question to consider is whether the best possible codes are linear. It turns out that if we randomly generate a linear code, we can achieve capacity (for Binary Erasure Channels and Binary Symmetric Channels). Below we give a very informal argument why this should work.

For this method, we randomly generate each bit of the parity-check matrix H according to a Bernoulli distribution. Then, use Gaussian elimination to build the generator matrix G from H. If we assume that the probability of a bit being flipped is p, then for an n-bit codeword there will be approximately np bits in error. When we multiply to compute $H(e)$, the resulting vector (called a *syndrome*) is a linear combination of the columns of H. There are approximately $\binom{n}{pn}$ possible syndrome vectors. The probability that any bit of a syndrome vector takes a particular value is

$$\Pr(\text{row} \times e = 0) = \Pr(\text{row} \times e = 1) = \frac{1}{2},$$

due to the fact that each row of H is generated randomly. So, the probability that two syndrome vectors are equal is

$$\Pr(\text{2 syndromes are equal}) = (\frac{1}{2})^{n-k}.$$

Considering all possible syndromes, $\binom{n}{pn}\ (\frac{1}{2})^{n-k}$ must be less than 1 in order for there to be some parity check matrix H such that we can avoid collision. Setting $k = nR$ we have:

$$\begin{aligned}
\binom{n}{pn} &= 2^{nh(p)} \\
2^{nh(p)} \times (\frac{1}{2})^{n-nR} &< 1 \\
2^{nh(p)-(n-nR)} &< 1 \\
2^{nh(p)-n(1-R)} &< 1 \\
nh(p) - n(1-R) &< 0 \\
h(p) - (1-R) &< 0 \\
R &< 1 - h(p) \\
R &= 1 - h(p) - \epsilon
\end{aligned}$$

for some small ϵ, which is arbitrarily close to capacity.

This argument can be made more precise by using something like Lovasz Local Lemma.