Homework 3

Due: Friday, December 9, 2016

1. **Implementing Binary Arithmetic Coding**

   Note: This problem is a programming problem. We have provided starter code in MATLAB and Python (available on the Homework page of the course website), but you are free to use another language if you prefer. In this case, you will also need to write an equivalent of the starter code and include clear instructions on how to run your code.

   Recall that arithmetic coding is a simple, computationally efficient method of encoding very long sequences of symbols drawn from an alphabet according to a known prior distribution. In this problem, you will implement functions to perform binary arithmetic encoding and decoding according to a pre-specified source distribution.

   Here are the specifications of the implementation:

   - Our alphabet has 5 symbols: $\mathcal{X} = \{1, 2, 3, 4, 5\}$.
   - The encoding alphabet is binary $\{0, 1\}$.
   - For simplicity, we will use the symbol "1" as a termination marker, explicitly indicating the end of the sequence. The encoder should stop encoding after it encodes "1", and the decoder should stop decoding after it decodes "1".
   - You should implement two functions:
     (a) `arithmeticEncode(p, S)` takes in a prior distribution p on $\mathcal{X}$ and a sequence S symbols in $\mathcal{X}$, and returns the arithmetic encoding $C(S)$ of S according to the prior distribution p. [1]
     (b) `arithmeticDecode(p, C)` takes in p and the output $C(S)$ of `arithmeticEncode(p, S)` and returns the original sequence S.

   In your homework submission, you should include the output of the starter code when run with your implementation of the above functions. You should additionally email the TA your code (3 files if you use MATLAB, or 1 file if you use Python).

   Hint: Recall that in arithmetic coding, an entire sequence is represented as an interval in $[0, 1]$. *While this works nicely in theory, in reality, a finite-precision computer quickly runs out of representable numbers in* $[0, 1]$. *One solution is to eagerly encode input symbols as soon as you can do so unambiguously, while rescaling the intervals so that they are never too small.*

[1] In MATLAB, S and $C(S)$ are numerical vectors, and p is a numerical vector of length 5. In Python, S and $C(S)$ are lists of integers, and p is a list of floats with length 5. $C(S)$ should only contains 0's and 1's.

2. **Shannon Lower bound on $R(D)$**

   In this problem, we will prove a fairly general lower bound, known as the "Shannon lower bound", that is often helpful for computing the rate-distortion function.

   Consider a discrete source X taking values in $\mathcal{X}$ and a distortion measure $d : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ that is symmetric in the sense that each column of the distortion matrix $[d(x, \widehat{x})]_{x,\widehat{x}\in\mathcal{X}}$ is a permutation of the sequence $\{d_x\}_{x\in\mathcal{X}}$ (e.g., this holds if $d(x, \widehat{x}) = 1_{\{x\neq\widehat{x}\}}$ is 0/1 loss). (This symmetry assumption can be weakened substantially, but we'll retain it here for simplicity.)

   (a) Define $\phi : [0, \infty) \to [0, \infty)$ by

   $$\phi(D) = \max_{p\in\mathcal{P}:\mathbb{E}_{X\sim p}[d_X]\leq D} H(p),$$

   where $\mathcal{P}$ is the family of distributions over $\mathcal{X}$. Show that ϕ is a concave function.

   (b) Show that, if $\mathbb{E}\left[d(X, \widehat{X})\right] \leq D$, then $I(X; \widehat{X}) \geq H(X) - \phi(D)$. Conclude that

   $$R(D) \geq H(X) - \phi(D). \tag{1}$$

   (c) Show that, if X has a uniform distribution, then (1) is tight, i.e., $R(D) = H(X) - \phi(D)$.

   (d) Suppose X is distributed uniformly on $\mathcal{X} = \{1, \ldots, 2k\}$, and we use the distortion measure

   $$d(x, \widehat{x}) = 1_{\{x\not\equiv\widehat{x} \mod 2\}} = \begin{cases} 0 & \text{if } x - \widehat{x} \text{ is even} \\ 1 & \text{if } x - \widehat{x} \text{ is odd} \end{cases}.$$

   Derive a simple closed form for the rate distortion function $R(D)$.

3. **Lower bound on group testing**

   Group testing is the analogue of compressed sensing over the binary field. In group testing, the goal is to figure out which of the k locations in an n-dimensional *binary* (0/1) vector b are non-zero. One can query some subset of the dimensions, i.e. a query vector ϕ is binary with 1s in the dimensions you want to query. The outcome of the query is binary and equals $\vee_i \phi_i b_i$ (is 1 if and only if at least one of the queried dimensions is 1) in the noiseless case and flipped independently with probability $q \leq 0.5$ in the noisy case. With m such queries, the noiseless outcome is $\Phi(b) = \Phi b$ where Φ is an $m \times n$ matrix containing the m query vectors and corresponding outcome in the noisy case.

   Using Fano's inequality, show that,

   (a) in the noiseless case, any group testing algorithm requires at least $(1 - \epsilon)k \log(n/k) - 1$ queries to have probability of error $\leq \epsilon$.

   (b) in the case with noise probability $q \in [0, 1/2)$, any group testing algorithm requires at least $\frac{(1-\epsilon)k\log(n/k)-1}{1-H(q)}$ queries to have probability of error $\leq \epsilon$.