Class 3 Exercises

CS250/EE387, Winter 2025

1. In the videos/notes, we saw that a random linear code of rate

$$R \geq 1 - \frac{\log_q(\mathrm{Vol}_q(d-1,n)) - 1}{n} \approx 1 - H_q(d/n) - o(1) \tag{1}$$

has distance at least d with high probability.

   (a) Would this style of argument have worked if we had started with a completely random code of about that rate?

   That is, let $C \subseteq \mathbb{F}_q^n$ be defined by including each element of $\mathbb{F}_q^n$ in C independently with probability q^{Rn}/q^n. Is it true that with high probability, a completely random code of rate (1) has distance at least d?

   If yes, work it out; if no, what goes wrong?

   (b) **(Bonus, come back to this if you have time later)**

   - If the argument would have worked, can you do better with a completely random code than a random linear code? Prove it.
   - If the argument wouldn't have worked, is it even true that a completely random code approaches/exceeds the GV bound?
     - If so, prove it.
     - If not, prove it. Is there some distribution close to that of a completely random code that would work instead?

Solution

(a) The argument does not immediately work. The issue is with the union bound: instead of union bounding over q^k codewords that might have low weight, instead we are union bounding over $\binom{q^k}{2} \approx q^{2k}$ *pairs* of codewords that might be close together. We get that

$$\Pr[\exists x \neq x' \in \mathbb{F}_q^k : \Delta(Enc(x), Enc(x')) < d] \leq q^{2k} \frac{\mathrm{Vol}_q(d-1,n)}{q^n},$$

and then we need

$$k \approx \frac{n - \log_q(\mathrm{Vol}_q(d-1,n))}{2},$$

which is off from what we want by a factor of two.

(b) It turns out that in fact, with high probability, a completely random code *won't* have distance approaching the GV bound. However, there will only be a few pairs of points that are too close together. To see this, notice that the above computation also gives the expected number

of $x \neq x'$ so that $\Delta(Enc(x), Enc(x')) < d$, and it is

$$\mathbb{E}[\sum_{x \neq x'} \mathbf{1}[\mathrm{wt}(Enc(x) - Enc(x')) < d] \approx q^{2k+nH_q(d/n)-n} = q^{n(2R+H_q(d/n)-1)}.$$

If $R \approx 1 - H_q(d/n) - \varepsilon$ to meet the GV bound, then this is about $q^{n(1-H_q(d/n)-2\varepsilon)} = q^{k-\varepsilon n}$, which is positive when $\varepsilon \to 0$. So in fact we *expect* there to be many pairs of codewords that are close together. (And it's not too hard to see that this holds with high probability).
However, the point is that there are not too many of these codewords, compared to the size of the whole code. So you can modify the completely random code by throwing out just a few points. In more detail, let C' be the same as C, except that we greedily remove from C any codeword which is closer than d to some other codeword. By the computation above and Markov's inequality, with probability at least $1/100$, the number of codewords that we remove is at most $100 \cdot q^{k-\varepsilon n}$. Thus, the number of codewords that remain is

$$|C'| \geq |C| - q^{k-\varepsilon n} = q^k - 100q^{k-\varepsilon n} = q^k(1 - 100q^{-\varepsilon n}) \geq \frac{q^k}{2} \geq q^{k-1}$$

when n is sufficiently large. But then the rate of $|C'|$ is at least $\frac{k-1}{n}$, which goes to $k/n = R$ as $n, k \to \infty$. So this gives us a family of codes with distance at least d and rate R as in (1), so it matches the GV bound. (This technique of throwing out a few bad codewords is called *expurgating*.)

2. Let $q \geq 3$ and fix some paramater $\alpha \in (1/q, 1 - 1/q)$. Suppose we draw an element $x \in \{1, 2, \ldots, q\}^n$, independently at random. Give an expression for the (approximate) probability that x has at least αn "3"'s in it. Your answer should be simple, and it should have a q-ary entropy term in it.

Solution

The probability that x has at most αn 3's in it is the same as the probability that a random vector in $\{0, 1, \ldots, q-1\}^n$ has at least αn 0's, since everything is symmetric. This is the same as the probability that a random vector has at most $(1-\alpha)n$ nonzero elements, aka, that a random vector lives in the Hamming ball of radius $1 - \alpha$. The probability of this is

$$\frac{\mathrm{Vol}_q((1-\alpha)n, n)}{q^n} \approx \frac{q^{nH_q(1-\alpha)}}{q^n} = q^{-n(1-H_q(1-\alpha))}.$$

3. Your friend is dubious about the statement, from the videos/lecture notes, that decoding a random binary linear code from up to half the distance is thought to be hard. They think that there is a polynomial time algorithm. Their reasoning is as follows.

   - Suppose that G is the generator matrix for a code C with distance d. Let $t < \lfloor \frac{d-1}{2} \rfloor$ be the number of errors that might occur.
   - The goal is, given a noisy codeword $y = Gx + e$ for $\mathrm{wt}(e) \leq t$, to find the x.
   - Since $t < \lfloor \frac{d-1}{2} \rfloor$, there is a unique such x, and we have $e = Gx - y$. In particular, x is the solution to the optimization problem
   $$x = \mathrm{argmin}_{x'} \mathrm{wt}(Gx' - y).$$
   - Since we are working over $\mathbb{F}_2$, for any vector v we have $\mathrm{wt}(v) = \|v\|_2^2$, where $\|v\|_2 = \sqrt{\sum_i v_i^2}$ is the ℓ_2 norm. Thus, x is the solution to
   $$x = \mathrm{argmin}_{x'} \|Gx' - y\|_2^2.$$

- But this is just linear regression! Use your favorite efficient technique from linear algebra to solve it. (For example, we could compute the pseudoinverse $G^\dagger = (G^T G)^{-1} G^T$ and compute $G^\dagger y$).

Unfortunately, your friend has missed something. What's wrong with the above approach?

Solution

The problem is that we are working over a finite field, not over $\mathbb{R}$. Unfortunately, linear regression doesn't work over finite fields! Intuitively, this is because orthogonality doesn't work like we expect. Over $\mathbb{R}$, the solution to the problem $\text{argmin}_x \|Gx - y\|_2$ is asking us to take the orthogonal projection of y onto the column span of G, which we can do efficiently. But "orthogonal projection" doesn't make sense over finite fields. Concretely, one thing that goes wrong with your friend's pseudo-inverse suggestion is that $G^T G$ may not be invertible, even though G has linearly independent columns. For example, if C is self-dual (so $C \subseteq C^\perp$, which can happen – consider $C = \text{span}\{(1, 1, 0, 0, \ldots, 0)\}$ over $\mathbb{F}_2$), $G^T G$ is actually zero!

The following problem is quite long, but most of it is exposition—we will walk through parts (a), (b) and (c) together as a class.

4. In this exercise we'll walk through an attack on the McEliece cryptosystem called "Stern's attack." It's not a devastating attack—by making the numbers big enough you can still protect against it—but it does give a non-trivial way for Eve to figure out what Bob's message is. (Note: If you don't care about crypto, this is still an interesting algorithm for decoding an arbitrary linear code!)

   (a) Recall that the problem Eve wants to solve to break the McEliece cryptosystem is to decode a binary linear code. Let $C \subseteq \mathbb{F}_2^n$ be the binary linear code that Eve has to decode in the McEliece cryptosystem. (So, in the language of the vidoes/notes, a generator matrix for C had a special form, $P \cdot G_0 \cdot S$). Say that C has dimension k, length n, and distance $d \geq 2t+1$. Let G be the generator matrix for C. (Note: in the lecture notes, G was $\hat{G}$...we're losing the hat since the original G won't be relevant for this question.) Eve's job is to find a vector x, given $y = Gx + e$, where $\mathrm{wt}(e) = t$.

   Consider the code C', one dimension larger than C, given by $C' = C + \{0, y\}$. (That is, $C' = C \cup \{c + y : c \in C\}$ — convince yourself that this is indeed a linear code if it's not immediately clear).

   Show that, if Eve can find a weight-t vector in C', then she can find Bob's message x.

   Solution

   Let $y = Gx + e$ as above. Then $e \in C'$ and has weight t. If Eve can find e then she wins, so we just need to show that there is no other vector of weight t in C'. Every vector in C' either looks like c for some $c \in C$, or like $c + y$ for some $c \in C$. Any nonzero $c \in C$ has weight at least $2t+1$ by the distance of C. On the other hand, any vector of the form $c + y$ that is *not* equal to e is of the form $Gx' + y = Gx' + Gx + e = G(x + x') + e$, for some $x' \neq x$. Since $x' \neq x$, the weight of $G(x + x')$ is at least $2t + 1$, again by the distance of C. Thus, by the triangle inequality, the weight of $G(x + x') + e$ is at least $(2t + 1) - t = t + 1$. Therefore, e is the unique element of C' of weight t, so if Eve finds any weight-t vector in C' then she wins.

   (b) In light of the previous part, we will focus on the problem of finding a low-weight vector in a linear code C'. (This will actually work for *any* linear code.). In part (b) of this problem, there is no question, we're just going to present Stern's algorithm for finding a codeword of C' of weight t.

   Before we get into it, here's a quick overview:

   A. We are going to construct a randomized parity-check matrix H for C'.
   B. We will enumerate over some guesses for (part of) the support of a weight-t codeword c.
   C. We will check to see if we can fill out the rest of the support.
   D. It turns out that A-C will succeed with some small, but not-too-small, probability. We'll repeat A-C a bunch of times until we win.

   Okay, now we'll go through steps A,B,C,D in more detail.

   A. **Construct a randomized parity-check matrix.** We'll also set up some notation. Fix parameters p and ℓ to be determined later. (Think of $p \ll t/2$, and think of $\ell > p$ as being pretty small as well). We are given as input a generator matrix of the code C'; use linear algebra to compute a parity-check matrix.

      i. There are several parity-check matrices of C'. We will choose a random parity-check matrix $H \in \mathbb{F}_2^{n-k}$ as follows. Choose a random set $W \subseteq \{1, \ldots, n\}$ of size $n - k$ and choose H — by doing row operations on the parity-check matrix you already have — so that the $(n-k) \times (n-k)$ given by the columns indexed by W form the identity matrix.

(Note: The astute reader will realize that not every set W will allow this! That is, if the columns indexed by W are linearly dependent you will not be able to diagonalize them to get I. Just ignore this[1]...)

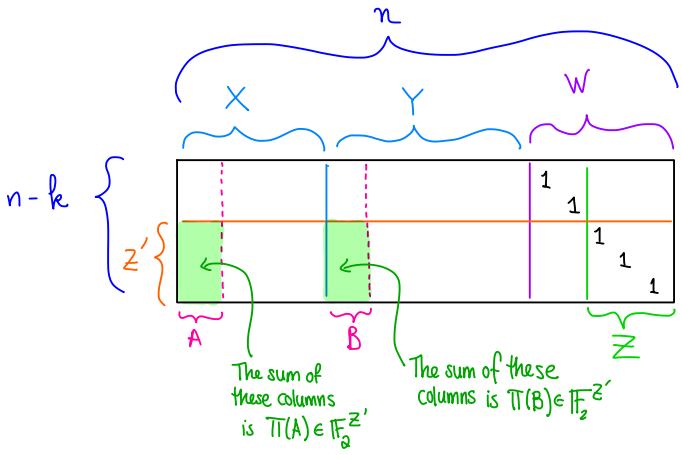
ii. Choose a random subset $Z \subset W$ of size ℓ. Let $Z' \subset \{1, \dots, n-k\}$ be the set of columns that correspond to Z according to the entries of H. That is, for each $z \in Z$, the z'th column of H is equal to $e_{z'}$ for some $z' \in \{1, \dots, n-k\}$. Let Z' be the set of all such z'.

iii. Consider the k elements of $\{1, \dots, n\} \setminus W$. Partition them randomly into two parts, X and Y. (That is, each of the k elements joins X with probability $1/2$ or joins Y with probability $1/2$, independently).
Notation: Let h_i denote the i'th column of H. Given a set $A \subset X$, define $\pi(A) \in \mathbb{F}_2^{Z'}$ by

$$\pi(A) := \left(\sum_{a \in A} h_a \right) \Big|_{Z'}.$$

That is, we look at all the columns indexed by A and add them together, then restrict to the rows in Z'. For $B \subset Y$, we define $\pi(B)$ similarly.

Altogether, the picture looks something like this, except the sets X, Y, Z, W are random and so probably not contiguous.



B. **"Guess" some potential supports.** For each set $A \subseteq X$ of size p, compute $\pi(A)$. For each set $B \subseteq Y$ of size p, compute $\pi(B)$. If you find A, B with $\pi(A) = \pi(B)$, make a note of it.
Aside: Later we will want to know how long it takes to do this. We probably won't spend time on this in class, but if you are curious, here is a sketch of one way to do this and how long it takes. The total time it takes is about:

- $O\left(p\ell\binom{|X|}{p}\right) + O\left(p\ell\binom{|Y|}{p}\right) \approx O\left(p\ell\binom{k/2}{p}\right)$ to enumerate over all A and compute $\pi(A)$, and then (in a separate loop) do the same thing for all of the B's.
- The number of vectors in $\mathbb{F}_2^{Z'}$ is 2^ℓ. So we can keep a hash table with 2^ℓ keys to find collisions. As a back-of-the-envelope calculation, the number of collisions that we expect (using the fact that everything in sight is random, so we hope that $\pi(A)$ and $\pi(B)$ are

[1] Stern's original algorithm says you should resample W until you can make the identity in those columns, and notes that this doesn't seem to affect the distribution of W very much in practice.

each approximately uniformly random in $\mathbb{F}_2^{Z'}$) is approximately:

$$\begin{aligned}\mathbb{E}[\text{number of collisions}] &= \sum_B \left(\sum_A \mathbb{P}[\pi(A) = \pi(B)] \right) \\ &= \mathbb{E}\binom{|Y|}{p} \cdot \binom{|X|}{p} \cdot \frac{1}{2^\ell} \\ &\approx \binom{k/2}{p}^2 \cdot 2^{-\ell}.\end{aligned}$$

(The above is not strictly legit — e.g., $|X|$ and $|Y|$ are correlated so I shouldn't just apply $\mathbb{E}$ to each of them independently — but it's close enough).
Thus, the amount of time it takes to iterate over all collisions and check the weight of $H(\mathbf{1}_A + \mathbf{1}_B)$ is about $O((n-k) \cdot 2p)$ per collision, or about

$$O\left((n-k)p \cdot \binom{k/2}{p}^2 \cdot 2^{-\ell} \right)$$

total.

C. **For each potential support, try to fill in the rest.** For each collision — that is, for each pair A, B so that $\pi(A) = \pi(B)$ — check to see if $\sum_{a \in A} h_a + \sum_{b \in B} h_b$ has weight exactly $t - 2p$. If it does, we **claim** that you can find a vector c of weight exactly t so that $Hc = 0$. Return this vector c. (And if none of these collisions result in returning something, return `fail`.)

D. **Repeat until you win.** Repeat steps A through C with independent randomness until you return something other than `fail`.

(Again, there is no question in part (b), just make sure you understand the algorithm).

(c) Justify the **claim** above: *If* $\pi(A) = \pi(B)$ *and if* $\sum_{a \in A} h_a + \sum_{b \in B} h_b$ *has weight exactly* $t - 2p$, *then there is a vector* c *so that* $\mathrm{wt}(c) = t$ *and* $Hc = 0$. Observe that such a vector c is indeed what wanted to return.

Solution

Consider the vector $v = \mathbf{1}_A + \mathbf{1}_B$. Since $\pi(A) = \pi(B)$, the product Hv vanishes on Z'. Since it has weight exactly $t - 2p$, that means that there are $t - 2p$ elements in $[n-k] \setminus Z'$ that are nonzero. These correspond to exactly $t - 2p$ elements of $W \setminus Z$, according to the identity-matrix-part of H. Call those $t - 2p$ elements $D \subset W \setminus Z$. Then let $c = v + \mathbf{1}_D$. Now by construction the weight of c is exactly t, and $Hc = 0$, since we chose D precisely to cancel the nonzero entries in Hv.

(d) Explain why the algorithm will succeed (with a given choice of Z, X, Y) if there is a codeword $c \in C'$ of weight t so that:

I. $c|_X$ and $c|_Y$ both have weight exactly p.

II. $c|_Z$ has weight zero.

Solution

If I. holds, then there is a choice of $A \subseteq X$, $B \subseteq Y$ each of size p so that A is the support of $c|_X$ and B is the support of $c|_Y$. Write $c = \mathbf{1}_A + \mathbf{1}_B + \mathbf{1}_D$, where $D \subseteq W$ and $|D| = t - 2p$. If additionally II. holds, then $D \cap Z = \emptyset$, which means that $(H\mathbf{1}_D)|_{Z'} = 0$. Since $c \in C'$, we have $Hc = 0$, and in particular $(Hc)|_{Z'} = 0$. Then,

$$0 = (Hc)|_{Z'} = (H\mathbf{1}_A)|_{Z'} + (H\mathbf{1}_B)|_{Z'} + (H\mathbf{1}_D)|_{Z'} = \pi(A) + \pi(B) + 0.$$

Thus, $\pi(A) = \pi(B)$. Therefore, we will identify A, B as a collision. Finally,

$$\mathrm{wt}(H(\mathbf{1}_A + \mathbf{1}_B)) = \mathrm{wt}(H\mathbf{1}_D) = |D| = t - 2p,$$

again using the fact that $Hc = 0$ and so $H(\mathbf{1}_A + \mathbf{1}_B) = H\mathbf{1}_D$. Therefore we will find and return c.

(e) The expected running time of the algorithm is thus:

$$O(\text{time for A-C}) \cdot \frac{1}{\Pr[\text{I. and II. occur}]}.$$

This might seem pretty big. After all, in steps B and C we are iterating over all possible A's and B's and collisions. Moreover, the probability that this works seems pretty small, so we are probably repeating the whole thing a lot. However, it turns out that this can result in a non-trivial speed-up over the naive algorithm. To see this, let's fix:

$$n = 300, k = 150, t = 20, p = 3, \ell = 12.$$

i. What order of magnitude is the running time of the naive algorithm to find a weight-t vector c? (The naive algorithm is "iterate over all $c \in C$ and see if it has weight t"). In particular, this running time is on the order of $2^{\text{something}}$. What is that something, for the choice of parameters above?

ii. What is the order of magnitude for the running time of Stern's attack? Just try to come up with a back-of-the-envelope running time, focusing on the value of "something" in $2^{\text{something}}$. **We will walk you through some key components below; you just have to put them together in the right way.** (You may want to use your phone as a calculator or something).

- Finding all colliding pairs (A, B) and checking the weight of $\sum_{a \in A} h_a + \sum_{b \in B} h_b$:
  - Iterating over A and computing $\pi(A)$ (and then doing the same for the B's) takes time on the order of:
  $$p\ell\binom{k/2}{p} = 3 \cdot 12 \cdot \binom{75}{3} = 2,430,900.$$
  - Iterating over all colliding pairs and checking the weight of the resulting vector takes time on the order of:
  $$(n-k)p\binom{k/2}{p}^2 \cdot 2^{-\ell} = 150 \cdot 3 \cdot \binom{75}{3}^2 \cdot 2^{-12} \approx 500,935,432.$$
  For the purposes of this back-of-the-envelope calculation, let's call this whole thing **about 500,000,000** operations.
- The probability, when choosing a random subset W of size $k = 150$ out of $n = 300$ things, that the $t = 20$ ones in our desired codeword c end up with exactly 14 ones in W and exactly 6 ones outside of W is:
$$\frac{\binom{20}{6} \cdot \binom{300-20}{150-6}}{\binom{300}{150}} \approx 0.03414.$$
- The probability, when choosing the partition X, Y, that the six ones not in W get split with 3 in X and 3 in Y is:
$$\binom{6}{3}/2^6 = 0.3125.$$

- The probability, when choosing a random $Z \subseteq W$ of size $\ell = 12$, that none of the $t = 14$ ones in $c|_W$ end up in Z is:

$$\frac{\binom{150-14}{12}}{\binom{150}{12}} \approx 0.3.$$

Solution

For (i), the naive algorithm is to iterate over all of the codewords and check the weight. This takes time 2^{150}, which is not feasible. (Note, one could also iterate over all $\binom{300}{20} \approx 2^{102}$ elements of weight t and check if they are in the code. This is a bit faster but still not feasible.

For (ii), conveniently we have worked out all of the high-order terms for the running time and the probability of failure.

Since 2 million is way less than 500 million, the "iterate over all A's and then over all B's" step is dwarfed by the "iterate over all collisions" step, so let's say that the running time for steps A-C is about 500 million operations. The number of times we need to repeat this is about

$$\frac{1}{0.03414 \times 0.3125 \times 0.3} \approx 312.$$

So the total running time is about $300 \times (500 \times 10^6)$ which is about 15×10^{10}. In order to compare this to what we had before (which was in base 2), we write

$$15 \times 10^{10} \approx 2^{37}.$$

So this is still a really big number, but it's a lot less than 2^{150}, and a running time on the order of 2^{37} would not be considered secure against a powerful adversary — modern supercomputers can perform over $10^{17} \approx 2^{56}$ floating point operations per second. There are smarter ways to implement the basic idea of Stern's approach that can bring the running time down even more.