[SQUEAKING] [RUSTLING] [CLICKING]

ANKUR MOITRA: Let's get started. So we're in the home stretch. We've got one more of these topics in coding theory. Peter will be covering a related topic next week, but that'll be a bit of a different flavor. So today, we're going to cover one of the other great results in coding theory, which has to do with error correction. So far, we've been exclusively talking about compression, how you can take a message and reduce the number of bits you need to transmit in order to reliably be able to recover what was the original message.

Now, today, we're going to talk about a very realistic scenario, which is when the way that you're transmitting introduces errors. So there's a very simple model, which is a good way to get started, which is called the binary symmetric channel, or BSC for short. The way that this channel works is that every time you input a bit, which is either 0 or 1, then it has some probability of flipping the outcome, the output of the bit.

So this BSC channel is going to be governed by some parameter p. And it's going to have the property that, with p probability, it ends up flipping the input and output. And with the remaining probability, 1 minus p, it just sends through whatever bit you sent along the channel. The trouble is that you don't know when and where these errors have occurred.

So the question is, you can think about p just for concreteness as being something like 1/3. If I'm trying to send a message along the channel to you, and I've already done my compression to get it down to the shortest possible string of 0's and 1's, now, when I send it along to you, and each bit gets flipped independently with probability of 1/3, what hope do you have of recovering the original message? And what are schemes you can design that combat the fact that you're anticipating there'll be errors?

So just to be clear, we're talking about each bit is flipped independently with this probability p. And now we can ask all of the natural questions about communication and the presence of errors. So the main thing we care about, which will be what we'll address today, is it still possible to transmit information reliably?

So let's think about what this question means with some simple ideas for how to combat the errors. So one simple example you can come up with for how to transmit along a noisy channel is to use what are called repetition codes. So here, the idea is that I take my original message. Let's say it's a string of 0's and 1's.

And what I'm going to do for what I'll transmit along the channel is I'm just going to keep repeating each input bit a certain number of times so that, even if some of them get corrupted, maybe it's OK. So I'll take my first bit, which is a 1, and I'm going to repeat it T times. And I'll do the same thing for my next bit. That's a 0. I'm going to repeat it T times. And I'll do the same thing for all of the bits in my original message.

So now you can see that, already, the way that I'm combating the fact that the channel is introducing errors is that I'm increasing the length of my transmission. I'm padding it with extra information in such a way that, hopefully, I can detect and correct whatever errors the channel has introduced. So the name of the game for today will be, what are the best possible schemes for how to pad the extra-- with extra information, in such a way that I can reliably figure out what my original message was?

So just to think about this, so let's say that the original message had length k. Then what I can do is I claim a good choice for how many times to repeat each message. There's maybe something like 100 log k. So my original message was length k. And then each bit, I repeat 100 log k times. So intuitively, if I took this transmission and I sent it through the channel, some of these 1's get flipped to 0's independently. Some of these 0's get flipped to 1's independently.

How would you try and decode what you receive to figure out what the original message is? Any ideas? And keep in mind, just to be concrete, you can think about the failure of the flip probability as being something like 1/3. So maybe a bit is more likely to come through in the clear without being flipped than it is of being flipped. So if I sent you this long, padded transmission, how do you think I'd figure out what the first symbol is based on what I receive? Any ideas? Yeah?

AUDIENCE: The majority of the block.

ANKUR MOITRA: Perfect, the majority vote because, intuitively, if I'm repeating this T times, then I would expect that the number of bit flips that I have is around T over 3. And so that means the majority of them should actually correspond to what my original bit was. In fact, what I care about is not just what the expected number of bit flips are, but I care about getting bounds on how likely it is that my decoding rule would fail.

So what kind of tool that we've talked about earlier in class might be relevant here? So I started off with bit flip probability, 1/3. And then I sent this bit through the channel with a repetition code. Now, my chance of getting the wrong bit for my majority vote decoding is the chance that the expected number of flips is 1/3.

But really, the true number of flips is much larger because it's larger than 1/2. That's the only way I would get the wrong decoding for this bit is if I ended up with half errors instead of 1/3. So what kind of tool that we've covered earlier in class might be useful to get a bound on the failure probability, the chance that my decoder fails here? Yeah?

AUDIENCE: Markov.

ANKUR MOITRA: Markov would do it. So I could certainly use Markov. That's an example of a tail bound. Now, one problem, though, is that Markov uses very little information about the random variables. So if I did this, what I care about is, what's the probability that I fail for this bit? What's the probability that I fail for this bit and so on?

And it turns out that if you used Markov, if you wanted to argue that, with good chance, I fail on none of these bits and I use the union bound, I would end up having to pad the length of the message by a factor of k instead of log k. So are there any ideas for how I could hope to get sharper bounds on how much I need to increase the padding by? Yeah.

AUDIENCE: Chernoff.

ANKUR MOITRA: Chernoff. That's right. So the key point is that, if I have this repetition code and I do it 100 log k times, I can use the Chernoff bound. So you remember, the way to think about the Chernoff bound is that the error exponent goes down exponentially in the length and the number of independent samples you're taking. So the probability that I have too many flips, that I was expecting 1/3 flips and I get 1/2, is something that goes down exponentially with log k.

So if I choose this constant in front of it, the point is that my failure probability is something like 1 over k squared. And then I can say that I don't fail on this bit, I don't fail on this bit, I don't fail on this bit, I don't fail on any of the bits. So that's exactly right. If we use the Chernoff bound, we could then take the majority vote in each one of these blocks, and we could show that the majority vote succeeds with probability, let's say, 1 minus little O of 1. So maybe I fail with some probability 1 over k, something like that.

So we're not going to fully proof these things right here because we're going to design a much, much better coding scheme. So repetition codes, the first time you see this problem of communicating on a noisy channel is the first scheme you come up with. But the big surprise is that you can do way better than just repeating your message.

So what I'm going to do right now is I'm going to tell you about, what are the desiderata? So what are the things we want out of our code? And then we'll talk about Shannon's noisy coding theorem, and we'll prove it. So I claim this code is pretty good, but not really the kind of thing that we're aiming for. So let's make precise some of the things that we talked about intuitively.

So we can talk about the rate of a code. The rate just describes, however long my transmission is, how long is that compared to how much information I'm actually transmitting? So in particular, what it is, is the length of the message divided by the length of the transmission. So in particular, what do we get for repetition codes? How good are they, according to this desiderata of having a good rate?

Well, we can check that the rate that we get is 1 over 100 log k because my original length of my message was k. My new length of my message is k times T. And I set T to be 100 log k. So I get this 1 over T type of rate. So the trouble with this is that this rate, if you look at it, as the length of my message k goes to infinity, my rate is going to 0.

So that's the question. That's the main question, really, is that necessary? So when we send longer and longer transmissions, does it mean that I'm communicating asymptotically very little negligible information when I'm sending things to you? Or is it possible to achieve a constant rate that's independent of the length of the message?

So this came as a pretty big surprise. And this is our main guiding question for today. Let's ask, can we make the probability of getting an error? And what I mean by that is the probability that we don't successfully decode the message we started from. I want the probability of my error going to 0. But I want to keep the rate constant.

And here, what I mean by going to 0 is we're taking the limit as the length of my input message, k, as going to infinity. So is it possible to do much better than these repetition codes? We got error, probability of error, was going to 0. That's fine. But we were transmitting very little information at the end of the day.

So, in fact, I need to make some other notions precise too because we have to talk about, what is our coding scheme? See, for the repetition codes, I did this all informally. But really, there are two pieces to the puzzle. One is the encoding, which is how you go from the message to what you actually transmit along the channel. It's the way that you're encoding the information. Repetition is a very simple way to do that, just by repeating myself.

And the other piece of the puzzle is the decoding, which takes whatever is the output of the channel, where some of the bits have been flipped. And we try to go back to the message we started from. So in our case, the encoding was repetition and the decoding was majority vote. But in general, we could think about very different kinds of schemes. So we'll say that an Rn, n encoding is a function, and which we'll write as Enc, that goes from 0, 1 to the Rn to 0, 1 to the n.

So R here represents the rate. And it's something that, for us, will be between 0 and 1. And you can see that I'm going from a shorter message. This is my k right here is the Rn. This is the input that I want to send to another party. And then I decide how I'm going to transmit it along the channel using a longer message. So it goes from Rn to n. So that's the first part, is that this is our encoding. And you can see that R right here is literally the rate of my code. So that's a parameter I'll care about.

And moreover, an Rn, n decoding is also a function. All it does is decode, goes from a received message, Rn, back to-- oh, sorry, n, back to what we think the person was transmitting to begin with. Now, the key point is how these two functions encode and decode, work in the presence of errors and bit flips, like when we send it along the binary symmetric channel.

So in particular, we'll be interested in the following quantity. This is a bit of a mouthful, so let me write it down, and then we'll parse it together. So if you give me two parameters, R, your target rate, and n, which is how long you want the transmission to be, well, I'm going to look at the min over all Rn, n encoding and decoding functions of the max over messages m, which are n, 0, 1 to the Rn of the probability that m tilde is not equal to m, where we think about it according to the following picture.

So we start off with our message. That gets turned into some transmission, which is just the encoding of my message given my encoding function. We send this through the binary symmetric channel. And we receive C tilde. We pass C tilde into the decoding function to get back what we think was the original message.

And the question is, how often do we have the property that m equals m tilde? So this is a lot of notation, but it's really very natural. So inside right here, I just care about the probability that I successfully decode the message you were trying to send to me, despite the fact that the binary symmetric channel has introduced errors.

Now, I want this probability, the probability that I fail to get the correct message, to be small, irrespective of what message you're sending me. So I look at how bad the failure probability can be in decoding and the worst case over all of my input messages. And then this latter thing is I want to create an encoding and decoding scheme that is good for all possible messages.

So I don't want to choose an arbitrary encoding, decoding function, like these repetition codes. But I want to figure out, what is actually the best encoding-decoding pair in terms of, you fixed what the rate is. How small can I get this failure probability to be? If I fix the rate to be a constant, like 1/2 or 1/3, can I get the failure probability for all messages to be going to 0 as n goes to infinity?

So this is a bit of a mouthful. Let me give you a moment to parse this thing, but are there any questions about the terminology or what I'm asking for here? Give me a thumbs up if this makes sense. Awesome. All right, so let me tell you Shannon's second main theorem, Shannon's noisy coding theorem, also proven around the same time.

And it asserts two things, which are upper and lower bounds that meet. So what I claim is that if the rate that you're trying to hit is larger than 1 minus entropy of p, then what I claim is that lambda star, Rn, the best you can do for encoding and decoding functions, that error rate goes to 1 as n goes to infinity.

That's the first part for the theorem. Let me state the other part. We'll talk about what this tells us and how to connect it back to our intuition. So on the other side, if R is less than 1 minus entropy of p, then what I claim is that lambda star, R, n, is going to go to 0 as n goes to infinity. So let's parse these things together because, to really unwrap what this theorem is stating, we have to go back to this min-max formulation.

So what this is saying is think about p as being a third in our binary symmetric channel. H is the binary entropy function. We talked about what that expression is, minus p log p plus 1 minus p log 1 minus p. So you can just plug-in for p is 1/3 and compute numerically what entropy of p is. It's some constant.

And now the point is that if you're trying to transmit at a denser rate than 1 minus entropy of p, you're going to fail. You're going to fail why? Because no matter what encoding-decoding pair you choose, there will be messages where the probability that you successfully decode the message is actually going to 0.

The probability that m tilde is not equal to m is going to 1. So when you're trying to transmit at too high a rate, you never decode the correct message. And on the other side, if the rate at which you're trying to transmit is less than the same quantity, 1 minus entropy of p, then what this means is that there is a good encoding-decoding pair.

We'll have to talk about what this pair is and how does it work, that the error probability goes to 0 as n goes to infinity. So this is a really striking theorem because, usually, what you-- what the terminology is that you call 1 minus entropy of p, the capacity, it's just some kind of rate that the channel admits for how densely you can communicate with someone else.

In fact, what's amazing is that these types of theorems are true not just for the binary symmetric channel, but if you invent your favorite stochastic channel, something which maybe erases symbols or deletes them, or has some hybrid model between these things, they always have a well-defined capacity, that if you're above the capacity, your error rate is going to 1. If you're below the capacity, you can get the error rate going to 0.

But the interesting quirk is that, in a lot of cases, the channel you write down, even though we know there is some line in the sand that you cannot cross, we don't actually know numerically what it is because it turns into an extremely hard math problem, to give an explicit characterization the way that we have one here.

So this is a lot to digest. Let's try and get some intuition for this. So I told you H of p is the binary entropy. So recall that the entropy of p is just minus p log base 2 of p minus 1 minus p log base 2 of 1 minus p. So in particular, what this function looks like is, if you choose p is 1/2, what's going to happen is that your entropy of p is 1.

So what you can show is that this binary entropy function, it's this nice concave function. And its maximum value corresponds to p is 1/2, at which point, it equals the value 1. So let's figure out what Shannon is saying here and see if it makes sense. So I have my binary symmetric channel. And if p is 1, then I can compute the channel capacity. That's 1 minus 1. So that's 0.

So Shannon is telling me that you can't communicate at any positive rate when your flip probability for your binary symmetric channel is 1/2. Does that make sense? Let's think about that one. So why is that reasonable, that when p is 1/2, that I can't actually transmit information to another party? Yeah.

AUDIENCE: There's no reasonable way for you to tell every bit that comes through [INAUDIBLE].

ANKUR MOITRA: That's right. So if you just look back to my definition of the binary symmetric channel, well, 1 minus p is 1/2, p is 1/2. So what that means is you sit there and you receive my first symbol, which is a 0 or 1. It has the same probability of being 0 and 1, irrespective of whether you are sending a 0 or 1 to begin with.

So literally, my channel just doesn't look at the message, and it just spits out random bits. Of course, there's no way to communicate. When p is 1/3, Shannon's theorem is telling us that we can do way better than the repetition code because this will be a constant because we'll be somewhere here. And whatever that value is numerically is the fixed line in the sand, we can communicate at that rate and know better.

And we don't have something like the repetition code, where the rate is going to 0. It's actually a universal constant, even though the error probability is going to 0. Let me test your intuition again. See, what's interesting is that this entropy function, H of p, is symmetric around p equals 1/2 because, when I substitute p with 1 minus p, it doesn't change the expression.

So the same thing is true for 1 minus entropy of p. So what Shannon's theorem is telling me is that, when my bit flip probability is 2/3, I can also communicate at some positive rate. Why is that intuitive? If I use something like the repetition code majority vote doesn't work anymore, but something else does. What would work if my bit flip probability were two thirds? Yeah?

AUDIENCE: [INAUDIBLE] minority?

ANKUR MOITRA: Minority. Exactly. All I do is I'm thinking that there'll be 2/3 flips here, and so I take the opposite of the majority vote. So this theorem is pretty deep. It takes a little bit to wrap your head around it. But so far, it makes sense. p is 1/2 makes sense. The fact that it's symmetric and p-- interchanging p and 1 minus p makes sense. And we're going to prove this theorem today.

And let me give you the proof right here. I'll put proof in quotes. So, really, the question is like, repetition codes didn't work for my encoding function. What should I use for my encoding function? So what you do is you let encoding be a random function. And that's it. That's the proof.

The entire rest of today's lecture will be removing the quotes around this proof because, in order to actually analyze what happens when we choose encoding to be a random function, well, we're going to have to get into a lot of tools from the probabilistic method. So this is our strategy. Let me introduce a bit more about what the strategy is.

So what we're going to do is we're going to prove this second statement first, which is, of course, the positive statement that we really can communicate at some positive rate. So let's prove 2 first. And let me be more precise about what I mean by encode. So I'm going to choose M, capital M, equals 2 times 2 to the Rn strings from my domain, 0, 1 to the n, uniformly at random.

So if you think back to what's going on, what is an encoding function? Well it's mapping each of the possible messages-- and there are 2 to the Rn of these-- to strings, and 0, 1 to the n. And what I'm doing here, it sort of looks like I'm looking-- I'm choosing the images of what those messages would be. There's 2 to the Rn messages that I could want to encode, and I'm just going to encode them with a uniformly random string from 0, 1 to the n.

For each one of the messages, I just choose the bits in its output completely at random. I have this extra factor of 2 for reasons that will become clear much later because this scheme by itself won't quite work. We'll have to remove some of these strings to really get by, and we'll talk about that later. But that's at least the intuition. And here I'm going to define the codebook, script C. That's just all of these different strings that I've chosen, C1 up to Cm, all right?

So now, let me tell you one more key definition. And then we can get to at least the statement of some of the key lemmas. So the last key definition I need is the notion of a ring of width gamma around some point C. So C here is a vector in 0, 1 to the n. And I'm going to let this ring be the set of all C tildes, such that the Hamming distance-- I'll tell you what I mean by that in a second, in case you haven't seen this term. The Hamming distance between C and C tilde minus np is, at most, gamma n.

All right, so let me be precise, so the Hamming distance is just the number of coordinates, i, where Ci is not equal to C tilde i. So my ring is going to be centered at some vector C. And C right here, remember, is a vector in 0, 1 to the n. And my ring is going to be a subset of 0, 1 to the n. It's the set of all C tildes that are the right distance from C.

In particular, I look at how many coordinates between C and C tilde are different. np is maybe the expected number of coordinates where they would be different because I'm sending C along the binary symmetric channel, and it flips each bit independently with probability p. So in expectation, if C tilde is the output of my channel, I would expect the Hamming distance to be n times p.

And here I just added a bit of slack. You should think of gamma as being something really, really tiny that's going to 0, and it just adds a little bit of slack because my ring now just looks like not the set of things at some fixed distance, but within some tolerance of that distance. So the way to think about it as a ring, just pictorially, is that I have my vector C here.

And then I'm looking at some disk around it of radius pn in terms of how many coordinates are difference between C and C tilde. And then I flatten this ring a little bit by some width, gamma. And this shaded region in between them is exactly what the set ring gamma around C is. So it has a center and it has a thickness. So any questions about that? This is just the definition.

All right, and so now let me tell you the key lemma, which is going to tell us-- it'll be the key for proving the second part of Shannon's theorem. And it'll even tell us what the decoding function should be. We'll have to work up towards proving this lemma, but I can already state the lemma right now. So here's the key lemma, the probability that if we take C, and we send it through our binary symmetric channel, and we get out C tilde, we want it to satisfy two conditions.

We want it to satisfy the condition 1, that C tilde belongs to the ring around Ci of some width, gamma. And we want that C tilde does not belong to any other ring of width gamma around any other Cj for all j not equal to i. Well, I claim that this is at least 1 minus epsilon. And this will be true for appropriate choice of gamma and as n goes to infinity.

So let's parse this thing. This still requires a lot of digestion to understand what's going on. So I'm punting on the question, what is gamma? We'll deal with that later. And all of these things I care about is really asymptotically as a function of n, all right? And, ultimately, I'm going to want this epsilon, actually, to go to 0 too. So you should think of epsilon as something that's going to 0 as n goes to infinity.

So I haven't told you what gamma is. We'll deal with that later. But each of these C's are-- actually, sorry, I should say Ci here. Each of these Ci's are the messages from my codebook, script C. Remember, there's capital M of them. And what I'm saying is, when I choose my encoding scheme that way, where each of these Ci's are a random length n string, what I'm saying is, when I pass that message through the binary symmetric channel, and introduce flips, and get C tilde, well, with high probability, Ci belong-- C tilde belongs to Ci's ring.

And he belongs to no other ring. So if this event really did happen with all but vanishing probability, I would be in great shape because it would give me my decoding algorithm. So once I've chosen this encoding function, where each of the messages, the 2 to the Rn of them, goes to a random image, one of these Ci's, then all I have to do is I pass my Ci through the channel, I receive C tilde, and my decoder is just going to check one-by-one each of the messages in my codebook, which ring C tilde belongs to.

And the point is that the only ring it will belong to is the actual message I started from if both of these conditions hold. So that's the main point, is that, if this event happens, I'm golden. So does everyone see why decoding would succeed if this happened? Does this make sense? Yeah? I saw less vigorous head-nodding. Yes? No? All right, so we're going to have to prove this lemma. That's one of the key things we're going to prove today.

A bunch of these things, I'm trying to do the calculations in a way that makes them as unpainful as possible, but there are limits. So the first thing I want to do is I want to tell you a back of the envelope calculation because all these things, like this binary entropy, H of p, is going to have to show up. And somehow it shows up from the combinatorics when we look at the size of these rings. So let me tell you a back of the envelope calculation, which will be very useful for us.

And you can make all of this precise, but let's do this a little bit more heuristically. So we know from Stirling's approximation that n factorial behaves like n over e to the n times square root of 2 pi n. Now, I'll tell you right now, what I want to do is I want to figure out what the size of the rings are. So I want to use Stirling's approximation to figure out some of expression for a binomial coefficient. And that's where the entropy is going to pop out.

Now, I can tell you right now that, for Stirling's approximation, when we apply it to the thing I'm about to do, this term doesn't matter. So if you want, you can just ignore it. I'll still keep track of it. But let's progress. So let's move on. So what I really care about is understanding the behavior of n choose pn.

This makes sense because I start off with a message which is, say, all 0's. Let's say that's the transmission that I send along the channel. And what I could care about is, how many outputs are reasonably likely? Certainly, if my trend-- my flip probability on the channel is 1/3, then I should expect that my output started off with no 1's. And after I send it through the channel, maybe I end up with n over three 1's. That's just p times n. And what I care about is how many outputs I'm reasonably likely to get from my channel.

So I care about, what are the asymptotics of this quantity? And I can use the fact that I can express the binomial as ratios of factorials. And I'll get out-- when I plug-in Stirling's approximation, I'll get this mess of an expression that I can simplify. So I get n over e to the n. I get np over e to the np. I get n times 1 minus p over e to the n times 1 minus p. And then I get some other stuff that I already told you isn't going to matter, but let me keep track of it anyways.

And this is not mysterious because this binomial is just n factorial over pn factorial times 1 minus pn factorial. And these are the leading order terms because these are much, much smaller. Now, what I can do is I can cancel things out because this n to the n cancels with the n to the np times n to the n of 1 minus 1p-- 1 minus p. This e to the n cancels with the e's down here.

The only thing that'll be left over is I'll get 1 over p to the np times 1 minus p to the 1 minus pn. And then I can cancel a bunch of these expressions if I want to. This is square root 2 pi np times 1 minus p. So just some algebra. Nothing fancy so far. And now I'm in good shape because I can take this expression. And I can take the log and divide by n.

So what this tells me is that 1 over n times log of n choose pn, well, this is exactly where the binary entropy pops out. So I get minus p log p minus 1 minus p log 1 minus p. And then all the other terms are much smaller order because they behave like log n over n. They're tiny compared to it.

So really, what's going on is all of the terms that don't cancel when I take their log and divide by n, that's exactly where the binary entropy pops out. And that's why entropy shows up in Shannon's theorem. So this is really just saying that the limit as n goes to infinity of 1 over n times log of this binomial coefficient, n choose pn, behaves like the binary entropy of p.

So this is a key algebraic fact that we're going to appeal to in our proof of correctness. And our proof that lemma 1 holds is just this fact. We're going to see a bunch of binomials in there, and I'm just going to think about them in terms of the binary entropy instead. So now let's go back to this lemma 1. And let me tell you something basic facts that are going to be key ingredients on the proof of lemma 1.

This part is really just calculational to get to the proof of lemma 1. But let's state one of our key facts. Fact one, well, I claim that ring gamma of C, the size of it, the number of strings that are contained in this set, is bounded by 2 to the entropy of p plus or minus gamma times n. So all this is, is, really, it's related to this fact.

Think about the case where gamma is 0 for our definition of the ring. All that's happening is the number of things that are in our ring is n choose pn. And so our fact tells us that, in that case, when gamma equals 0, it behaves like 2 to the entropy of p times n. That's literally what this expression means when I multiply through by n and take everything in the exponent of 2.

And as I take this thickness, gamma, to be something, but, ultimately, I'm going to care about this thickness going to 0 as n goes to infinity. It's only going to be some fudge factor in the exponent. So you can write down this fact more precisely. It's a bit tedious because it involves a lot of algebraic expressions, but this is really how the size of the ring behaves up to some fudge factor that depends on the width of the ring. It's 2 to the entropy of p times n. That's our first fact.

And so now what I claim is that we can get into these questions like, how should I choose gamma? What I claim is that there is a gamma, such that with probability at least 1 minus epsilon over 2, p minus gamma n is a lower bound for the Hamming distance between C tilde and C. And p plus gamma n is an upper bound.

So we can forget about how we're connecting this to coding theory right now. This is a purely probabilistic question. I take any message, C of length n, and I send it along the channel. And then I observe C tilde. And remember that C tilde just comes from flipping each bit independently with probability p. So I can care about how many flips were introduced.

This is really just the standard coin-flipping question is I want bounds on the probability that the number of times I flipped the bit is larger than its expectation or is smaller than its expectation. And if my gamma is some appropriately-chosen slack, then I can guarantee that you're not outside of these bounds with all but some tiny failure probability.

So all this is this follows because we've proven things very much like this. It follows from Chebyshev or the weak law of large numbers because, as n goes to infinity, you expect the empirical number of bit flips you get to be very, very close to the expected number, p times n. So with these two facts, we're now in good shape for trying tc prove lemma 1. So let's put these things together and prove lemma 1.

So how does the proof of lemma 1 go, our key lemma in Shannon's theorem? So first of all, from fact 2, we have the probability that C tilde-- how did I write it there? C tilde belongs to the ring for some appropriately-chosen gamma around Ci is at least 1 minus 1 over 2. So my failure probability is at most epsilon over 2.

And now what we can do is let's bound the other probability. So let's bound the failure probability of the other event, namely that C tilde belongs to some other guy's ring. And so what we can do is we can fix j not equal to i, so one of the other code words in our codebook. And what we can do is, in that case, the probability that C tilde belongs to the ring of some width, gamma, around Cj, well, I claim that this is the same thing.

This is one of the key tricks in here, is the same thing as the probability that Cj belongs to the ring of C tilde. So this looks like a pretty obvious statement because this notion of ring is just symmetric. It doesn't-- whether C tilde belongs to Ci's ring, that's the same thing as vice versa, if Ci belongs to C tilde's ring.

But what's going on here is we have to go back to how exactly we chose the codebook to see why this is important, is that a priori, this might seem like a fairly complicated event to analyze because I have some message that's received by my decoder. That's C tilde. What's the chance he belongs to some other ring around Cj? The key point is that when I express the probability this way, remember, I chose each of those code words randomly. So what I could do is I could run the whole experiment in my head, but be lazy about when I choose Cj.

So you tell me you're sending Ci along the channel. Fine. You tell me you're sending that particular message, I fix what Ci is, we send Ci through the channel, and we receive C tilde. But only now after I receive C tilde do I pick what Cj is. And the chance that Cj, if it's a uniformly random string, lands in this particular ring, I can bound that based on the size of the ring.

So now I can appeal to fact 1. So we can upper bound this probability by this 2 to the entropy of p plus or minus the slack gamma, n, over 2 to the n, just because that's the size of the ring C tilde, is exactly this quantity from fact 1. And we're choosing Cj uniformly at random from all of the strings of length n that are binary. So we get exactly this expression.

And so now we're in good shape because we can appeal to a union bound because we don't care about this just for one particular j. We care about it for all j not equal to i. So now, by a union bound, the probability that C tilde belongs to any ring around a Cj, where j is not equal to i, the probability that's true for any j is, at most, the size of our codebook times this probability, 2 to the entropy of p plus or minus gamma n over 2 to the n. So this is just a simple union bound.

And now what we can do, if you ignore this gamma, what's going on is I have my codebook, remember, looks like 2 to the 1 minus entropy of-- it looks like this codebook right here, is 2 times 2 to the Rn. And we're assuming that R is strictly less than 1 minus entropy of p. What's going on right here is I have 2 to the minus 1 minus entropy of p.

So the point is that, once my rate is strictly smaller than what's happening in the exponent over here, then I'm in good shape because this probability goes to 0 as n goes to infinity. So that's the proof of our key lemma. It's really just the counting argument at the end of the day, is that my codebook isn't so large, like I union bound over all of the things in my codebook.

I look at the size of each of these rings compared to the size of the underlying domain. And as long as the size of my codebook times the size of the ring is strictly smaller than my domain, even asymptotically, then I'm in good shape. So this proves these two key facts. We know that the probability you don't belong to the ring you start with is very small, is at most epsilon over 2 by the weak law of large numbers.

And we know by a union bound and a counting argument that the probability that you belong to any other ring is very small. And the key step that we used was really this transformation right here, that the probability that C tilde belongs to another ring because the Cj's are chosen randomly, that's the same as the probability that my newly sampled Cj belongs to the ring around C tilde.

So are there any questions about this? This is one of the key proofs for today, is the proof of this key lemma. Any questions? Give me a thumbs up? Good? All right? So now let's use this lemma. And there'll be another trick. It's a subtle trick, but it's a cool trick. So I need a little bit more terminology. And then we're going to be able to prove Shannon's theorem.

Because one of the things to think about is that, so far, our attempt at proving Shannon's theorem, we're choosing a random codebook. And that codebook then defines what the encoding and the decoding functions are. Our encoding sends each message to one of those randomly chosen codewords in our codebook. Our decoder just looks at which message's ring you belong to.

But at the end of the day, to prove Shannon's theorem, I don't want to argue that a random code is usually good. I want to show that there's a code that's always good. So that subtlety is really where the probabilistic method is going to show up, is because we have to argue that there actually is one fixed choice of this random codebook that makes everything small.

And this is a little bit subtle to really prove this statement, but I need some more intuition. I need some more notation. So I'm going to let Ei of my codebook, script C, be the event that everything goes well, that I send in my ith code word from my codebook through the binary symmetric channel. I get out some received message, C tilde, i. And I have exactly the properties that are promised to me in this key lemma, lemma 1.

I want that my received thing belongs to the ring around Ci, and it belongs to no other ring. And this is true for all j not equal to i. So notice that this event now is a property of the actual codebook too. It's a property of what errors the channel introduces and a property of the choice of the codebook.

And what we're going to do is we're finally going to let lambda i of C denote 1 minus the probability that this event happens. So it's really the probability that there's a decoding failure. But notice that the decoding failure then depends on what the original-- that probability depends on the choice of the codebook. And it depends on which message we're sending, all right?

And so what I claim, we just proved through lemma 1, is the following statement. We proved that the average over all of the possible codebooks, because we're choosing these codebooks uniformly at random, the average over all of the messages that we want to send of having a decoding failure is small.

So what does this thing say? Let's parse this in English. When I choose a random codebook-- this is the average over a random codebook. When I choose a random message that I want to encode, the probability that I have a decoding error for that codebook and that message on the binary symmetric channel is at most epsilon.

This is not the same thing as saying that there exists a codebook that's always good for every message. And that's exactly what we have to try and twist the statement into. So we proved through lemma 1 was this. And we really need something else that gets rid of this average over codewords. And so what I claim now is, by the probabilistic method, we claim that there exists a codebook with the property that the average over all the messages of the decoding failure probability is at most epsilon.

So the probabilistic method is one of these things where it looks very natural, sometimes, when you state it. But other times, when it's happening with other layers of abstraction, it's a bit harder to wrap your head around. So we use the probabilistic method to show things like, if we choose that there's a graph that has no large clique or no large independent set-- we'll talk about that, I guess, later.

And here, we're really using the probabilistic method the same way that I know that this expectation of this quantity inside is small over a typical codebook. So there must exist one codebook where it's small as well. So does everyone see how I got from here to here using the probabilistic method? Is that all right? Yeah?

So now we're in good shape, and I can tell you how to finish the rest of the argument. Let's see how much space I need. Not too much. So now we're starting to make progress because, instead of saying over a random codebook, over a random message, our decoding failure is small, we know that we have a codebook where the average over the messages, the decoding error rate is small.

But we really care about, back to our original statement of what we wanted to prove, was we want that there's a codebook so that every message can be decoded well, not a random message. So we have one remaining expectation that we want to prune, which is this average over M, all right? And now we're going to do it in a kind of cool way, which is I claim by Markov's bound.

We know that at least half of the i's satisfy the bound that lambda i of C is at most 2 epsilon. So this is our old friend, the Markov bound, because we're looking at random variables that are 0, 1 valued, like did you have a decoding failure? And if the average over these messages, 1 through M, that we want to send has the decoding failure probability bounded by epsilon on average, we know that for at least 1/2 of the realizations, the probability that they're larger than 2 epsilon, the decoding failure probability, is small.

So this is the usual thing, that if this weren't true, then the expectation would be higher because these are non-negative random variables. And so now what we can do is we can let C prime, our eventual codebook, be the codebook restricted to just these messages, just these codewords, the ones that satisfy this condition right here, that their lambda i for that particular choice of the codebook is at most 2 epsilon.

So what does this mean, just putting this back into English? It means that this new codebook, C prime, is itself an Rn, n encoding. That's exactly why I started off with my original codebook was on capital M things, where M was equal to 2 times 2 to the Rn, was exactly this extra factor of 2 that I was going to have to give up when I went from C to C prime.

So whatever's left is still an Rn, n encoding because there are 2 to the RNA things in it. And we know, just by unwrapping the definition of what lambda iC is, what this means is the probability that C tilde i is not equal to--sorry, that the probability that M tilde is not equal to M is at most 2 epsilon because, literally, the way we define this was how often we expect for the given code word to have a decoding failure probability on that particular message?

So the only code words that survived were exactly 1's, who, when I fix the codebook and I fix the message, have a very small failure probability of getting the wrong decoding. So this proves the most interesting part of Shannon's theorem, that we can get positive rate and error probability going to 0. So we can drastically improve upon the repetition code.

If you actually work out the details, what ends up happening is that it's not just that the probability of failure goes to 0. The probability of failure actually goes to 0 exponentially fast, which is really cool. So it's way better than the repetition code on a few different fronts. If I wanted the failure probability to be exponentially small for my repetition code, I would have had to increase it way beyond this log k blow up.

So a bunch of people at the time really thought that the repetition code was optimal. So this came as quite a surprise that this was not true. I will briefly give you the intuition behind the first part of Shannon's theorem, and then we'll call it a day. So remember that the first part of Shannon's theorem was that you can't communicate at a rate larger than the capacity, 1 minus entropy of p. If you do, the probability of getting the correct message is going to go to 0.

And so let me give you, very briefly, the intuition. So let's suppose that we had a coding scheme, where the rate was larger than the capacity, 1 minus entropy of p. And let's suppose, for the sake of contradiction, that it really wasn't such a bad coding scheme. Maybe the probability that it successfully decoded the message is at least some epsilon, like 0.01, let's say.

Well, what we're going to argue is that we'll prove a contradiction, that the rate really cannot be this large. So the main point is that an epsilon over 2 fraction of ring gamma Ci must be decoded to Ci. So here the point is, originally, our decoder was just based on the ring.

But now we have no idea what the decoder is. But the same way, whatever our decoder is, we know that when we pass through this transmission along the channel, we're going to end up in this ring because that was the first part of lemma 1, was just that we land in the ring with all but very small failure probability.

And once we're in this ring, because that's always where we land, the fact that we're successfully decoding the original message is just saying that, among these outputs that we often reach, well, many of them must be decoded to the correct message. That's just the definition of-- that's just the translation of what this statement means.

But now the point is that, but the C tildes that are decoded to different Ci's must be disjoint because your decoding function has to map you to a unique original message. So if I look at, over all these different messages, which things are decoded to it is each of these sets are disjoint.

And so what does this mean? Well, if we add up just the total volume of these sets, we know that they're at least an epsilon over 2 fraction of the ring. We know that there's one of these rings for each of the codebook. We know that the ring has size 2 to the entropy of p times n. But this cannot be larger than 2 to the n because that's the number of possible C tildes there are, period.

And so when we rearrange this bound, we'll get exactly the thing we wanted, that the size of my codebook is at most something like 2 to the 1 minus entropy of p times n. So this is just meant as a sketch because, really, to do this precisely, I would have to prove a lot of the helper lemmas here and give similar kinds of bounds. But the intuition is much simpler. And we're only going to ask you about, really, the second part for Shannon's theorem. So are there any questions about this? Yeah?

AUDIENCE: Sorry, can you explain the [? x1 ?] over 2 part?

ANKUR MOITRA: Yeah, that's right. Sure. So what is the statement saying right here, is that at least an epsilon fraction-- so if I look at-- when I start off with Ci, and I look at the output after the binary symmetric channel, then at least an epsilon fraction of those outputs must be successfully decoded to the correct original message.

Now, almost all of the outputs live in the ring. So if we just throw out all the things that aren't in the ring, we're not really changing what the output distribution looks like from our BSC. And that means that we still must have-- at least a large fraction of this ring must also be successfully decoded. To fully prove this would be a lot more work and would be a bit tedious, but that's the intuition. Any other questions? Great questions, though.

So let me end with a little bit of a historical perspective. So this Shannon theorem is quite amazing. It's actually not such a perfectly ideal scheme. So we have the same kinds of issues that we had in Shannon's noiseless coding theorem. So we proved Shannon's noiseless coding theorem about how well we can compress a first-order source.

But then the thing I complained about was the fact that, really, if we were to execute the scheme, I would have to write down the full set of encodings, and that would be gigantic. I'd have to anticipate everything you could ever want to compress and write down this codebook of what to compress it to.

Same problem happens here, that choosing this codebook, script C, randomly is very convenient for the proof because it makes it easier to reason via the probabilistic method. It makes it easier to reason about failure probabilities of events because it boils down to some combinatorics of the sizes of rings. But the trouble is that this would be a horrible coding scheme if I really had to write down exponentially many things and define my codebook.

So what Peter is going to be talking about on Tuesday is a very powerful paradigm, which is called a linear code. So instead of explicitly writing down all of the codes and codewords in my codebook, I can instead implicitly describe them as matrix vector products. All of my messages are vectors, and my codebook describes a matrix. And when I want to encode some message as what it maps to is the code word, I just take the matrix times the vector.

So this now becomes a much more concise way to define the codebook. So he'll tell you about linear codes. And then the other thing, which that next lecture will touch on, is that, right now, the crucial assumption that we used in the proof of the theorems today was the fact that the binary symmetric channel introduced flips randomly.

So bit-by-bit, it had a third probability of introducing a flip. Now, it turns out there are other models for codes, where you could instead think about. worst case. What if sitting on your channel is an adversary who gets to corrupt 1/3 of your bits, but they get to choose when and how to corrupt those bits? That's called adversarial coding, and we can get similar semantics there, where we want an encoding and decoding function that now works even if the channel is not stochastic, but is bounded in terms of its budget of corruptions.

So for those types of problems in general, we don't understand such tight trade-offs as we do for Shannon. But he'll tell you about linear codes and he'll tell you about adversarial decoding. And then after Peter's done with that, I'll tell you about some other topics that's connected to, namely, cryptography and secret sharing. And then the semester will be over. So just to give you a little bit of a preview of what we're going to cover in the remaining lectures. But let's stop here, and I'll take any questions you guys have offline. Thanks.