Pseudorandomness Contest

1. Pseudorandomness contest, Round 1
2. Pseudorandomness contest, Round 2
3. Pseudorandomness contest: prizes, results, and analysis

Pseudorandomness contest, Round 1

This is a linkpost for
https://ericneyman.wordpress.com/2020/12/13/pseudorandomness-contest-round-1/

I've decided to run a pseudorandomness contest — a reverse Turing test of sorts, if you will. Winners will get to send some of my money to a charity of their choice! Here's how it's going to work:

Round 1: Pseudorandom number generation

Should you choose to take part, you will have **10 minutes** to come up with a sequence of **150 bits** (i.e. zeros and ones). The deadline for doing so is **Saturday, December 19th 11:59 ET**. You will use the form linked at the bottom of this post for submission. (But read the rest of this post first.)

You may not use a computer or any other resources. For example, you may not use a watch to try to generate random numbers (though you can use one to keep track of the time remaining). No using books, or calculators, or deriving random bits from cracks in the ceiling, etc. The only exception is: you may use Notepad (or something similar) to write down your sequence of bits as you come up with them (but not for scratchwork — everything besides writing down your bits must be done in your head). I recommend this website so you can keep track of how many bits you have. (You may not use a counter that tells you how many of each bit you have.) You may use anything you currently have stored inside your head. You may not memorize anything in advance of participating, though you are allowed to think in advance about general strategy (without writing anything down). *[Edit 1: Please do not look up strategies; any thinking you do in advance should be your own.]* ***[Edit 2: Think about the rules this way -- ideally you'd be doing this in a uniformly blank room with a computer where the only things you can do are type 0, type 1, backspace, and go to some point in the string you have already typed, as well as see how much time you have left and how many bits you have already typed.]***

Round 2: Distinguishing real and fake randomness

If there are n entries in Round 1, I will use a computer to come up with n sequences of 150 random bits. I will post all 2n sequences in a random order in a Google sheet. You will have as much time as you want (subject to a deadline — probably a week or so after I post the Google sheet) to try to figure out which sequences were generated by a human and which by a computer. Then, for every sequence you will submit a probability that that sequence was made by a computer (i.e. "truly" random).

You may use any resources you want to complete this task, e.g. write a computer program or do research on the Internet. The only restriction is: you may not use a program written by someone else for this (or a similar) purpose. You also may not collaborate with other contest participants. (These details are subject to change, but this will be the basic format.)

Grading

Participants in Round 2 will be graded using Brier's quadratic scoring rule, which means that they will be incentivized to report their true probabilities. Participants in Round 1 will be graded based on the average probability-of-being-generated-by-a-

computer assigned to their string by Round 2 participants. The higher this number, the better a participant did.

Prizes

Prizes will be of the form “I will donate $X to a charitable organization of your choice”, subject to my approval (e.g. I won’t donate to the Trump Foundation). At least $75 will be donated to charity through this contest, and at least $150 (possibly more) if both rounds end up with 10 or more participants. (This means that by participating, you’re causing more money to go to charity in expectation!) I haven’t decided exactly how that will be distributed, but my inclination is to give more prize money to Round 2 winners, since that round is more time-consuming.

More details

- People may participate in both rounds. In fact, participants in Round 1 will have a small advantage in Round 2, since they will be able to say 0% for the string they submitted.
    - To make the contest incentive-compatible, in Round 2 you will be able to indicate which string from Round 1 is yours. The probability you submit for that string (i.e. probably 0%) will not count toward your Round 1 grade (so your Round 2 submission doesn’t penalize your Round 1 submission).
    - Probabilities in Round 2 will be normalized in some way to add to n, so that there isn’t an incentive to say 0 to everything if you’re really keen on winning Round 1.
- In Round 1, the rules allow you to e.g. use the digits of Pi to come up with random numbers. You’re allowed to do this, but I advise you against it because Round 2 participants will be on the lookout!

How to participate

Use this form! (I ask for your email address so you can get a receipt with your submission, so I can let you know when Round 2 is available, and so I can contact you if you win a prize. Email addresses will be kept private.)

Questions?

Comment below my blog post!

Pseudorandomness contest, Round 2

This is a linkpost for https://ericneyman.wordpress.com/2020/12/17/pseudorandomness-contest-round-2/

[Note: if you're reading this on 12/20 I recommend checking again tomorrow in case I need to make any clarifications to the rules.]

Last week, I asked you all to take 10 minutes to write down and submit a 150-bit string, with the goal of making your string "seem random" without the aid of any sources of randomness (except your brain and anything you had memorized). I received 62 submissions; thank you to everyone who participated!

Now it's time for Round 2, which I think of as the "real" part of the contest. In addition to the 62 strings you all submitted, I created 62 "truly" random strings with my computer, and I have put the 124 strings in a random order. Your goal is to figure out which of the strings are truly random and which ones were submitted by a Round 1 participant. Note that **you can participate in Round 2 even if you did not participate in Round 1**. Just as for Round 1, there will be prizes for doing well! (See below for details.)

Click here to see the binary strings, but you should read the rules before starting!

The rules: For each string, you will be asked to submit your guess about the **probability** that the string is truly random. For example, you might say "70%" for a string you think is probably random and "5%" for a string you're pretty sure isn't random.

Unlike in Round 1, this time you are free to use almost any resources you want to complete this task. You can write a computer program. You may do Google for advice about how to tell apart truly random and fake-random strings. You may look up numerical constants people might have used to generate their random strings. There are two things you are not allowed to do:

1. You may not **interact with a person**, e.g. ask questions on Stack Exchange or talk to a friend, as part of completing this task, unless you are on the same team (see the team policy below).
2. You may not **use code or pseudocode that was written by someone else, if that code is meant to be used for distinguishing random and pseudorandom strings**. That is, you're allowed to read articles describing how to tell strings apart, but may not use code that someone else has written, or something that is written in a format that may as well be code. (You *may* use code that someone else wrote for some other purpose, e.g. most libraries.) I'll trust you in terms of where to draw the line between "algorithm description" and "pseudocode", but the line I have in mind is something like: if it's mostly text then it's fine to look at and if it looks basically like code then it's not.

Team policy: Teams of up to **three people** are allowed. However, if multiple members of your team participated in Round 1, as part of your submission you must indicate which Round 1 strings were submitted by your team members, and the weights of those strings will be reduced in Round 2 scoring (so that the total weight of your team's strings is 1).

[Rules change 12/20: Previously the rules said that I would normalize your probabilities to add to 62 for incentives reasons. It was pointed out to me that there's little or no incentives issue, so I no longer plan to do this.]

The deadline: Sunday, December 27th at 11:59 pm ET. However, I reserve the right to extend the deadline by one week. Specifically, I am hoping that the following things will happen by December 27th:

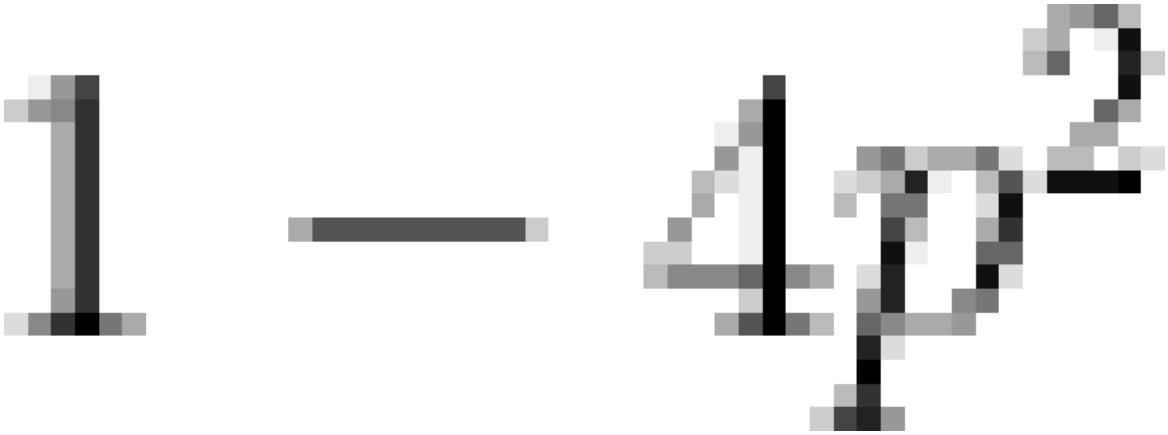
- There will be at least 10 submissions.
- There will be at least 3 submissions with a score of at least 15 (this is my cutoff for considering someone to have done a good job).

In the (I think unlikely) even that one of these doesn't happen, I will probably extend the deadline.

Click here to submit your probabilities! (But I encourage you to read the details below.)

Round 2 scoring: Let's say you submit a probability p for a string. If the string is truly random, your score will be

and if the string was submitted by someone in Round 1 then your score will be



Basically, this means that your score will be 0 no matter what if you say 50% for a string. The highest score you can get is 1 (if you say 100% and it's truly random, or if you say 0% and it's not truly random), and the lowest score you can get is -3 (if say 100% and it's not truly random or the reverse). Your total score will be the sum of your scores for all the strings (weighted as per the team policy above). Note that if you're going for maximizing your expected score, this scoring system incentivizes you to be honest.

Round 1 scoring: If you participated in Round 1, your score for your Round 1 string will be an average of all probabilities assigned to your string by all Round 2 participants (excluding you), weighted by their Round 2 score. (Entries with negative scores won't count for Round 1 scoring.) So basically, you'll get a good score in Round 1 if you manage to trick Round 2 participants into thinking that your string is truly random, with higher weights assigned to Round 2 participants who did well. (The weighting feature of the scoring is new compared to what I wrote in the Round 1 post.)

Note to people who participate in both rounds: You have a very slight advantage in Round 2 if you know your Round 1 string because you can get a free point by saying 0 for your string. You should have received an email (sent to the email you entered when you submitted your Round 1 entry) with your submission, so you should be able to figure out your string. If you didn't receive the email, you can email me (see here for my email) and I'll try to figure out your string.

Prizes: If you do well, you will be able to decide what charity I send some amount of money to (subject to my approval). Round 1 amounts will total to at least $50 (probably exactly $50). Round 2 amounts will total to at least $50, and at least $100 (quite possibly more) if the "10 submissions, 3 good submissions" criterion described above is met. I haven't decided on the particulars of how I will award these prizes, but I hope you trust me to be impartial in such determinations.

Once more, **here** is the link to the binary strings, and **here** is the link to submit your Round 2 entry. Good luck and have fun!

(Questions? Comment below, or here if you want me to notice right away!)

Pseudorandomness contest: prizes, results, and analysis

This is a linkpost for https://ericneyman.wordpress.com/2021/01/15/pseudorandomness-contest-results-and-analysis/

(Previously in this series: Round 1 , Round 2)

In December I ran a pseudorandomness contest. Here's how it worked:

- In Round 1, participants were invited to submit 150-bit strings of their own devising. They had 10 minutes to write down their string while using nothing but their own minds. I received 62 submissions.
- I then used a computer to generate 62 random 150-bit strings, and put all 124 strings in a random order. In Round 2, participants had to figure out which strings were human-generated (I'm going to call these strings **fake** from now on) and which were "truly" random (I'm going to call these **real**). In particular, I asked for *probabilities* that each string was real, so participants could express their confidence rather than guessing "real" or "fake" for each string. I received 27 submissions for Round 2.

This post is long because there are lots of fascinating things to talk about. So, feel free to skip around to whichever sections you find most interesting; I've done my best to give descriptive labels. But first:

Prizes

Round 1

Thank you to the 62 of you who submitted strings in Round 1! Your strings were scored by the *average probability of being real* assigned by Round 2 participants, weighted by their Round 2 score. (Entries with negative Round 2 scores received no weight). The top three scores in Round 1 were:

1. **Jenny Kaufmann**, with a score of **69.4%**. That is, even though Jenny's string was fake, Round 2 participants on average gave her string a 69.4% chance of being real. For winning Round 1, Jenny was given the opportunity to allocate **$50** to charity, which she chose to give to the **GiveWell Maximum Impact Fund**.
2. **Reed Jacobs**, with a score of **68.8%**. Reed allocated **$25** to **Canada/USA Mathcamp**.
3. **Eric Fletcher**, with a score of **68.6%**. Eric allocated **$25** to the **Poor People's Campaign**.

Congratulations to Jenny, Reed, and Eric!

Round 2

A big thanks to the 27 of you (well, 28 — 26 plus a team of two) who submitted Round 2 entries. I estimate that the average participant put in a few hours of work, and that some put in more than 10. Entries were graded using a quadratic scoring rule (see here for details).

When describing Round 2, I did a back-of-the-envelope estimate that a score of 15 on this round would be good. I was really impressed by the top two scores:

1. **Scy Yoon** and **William Ehlhardt**, who were the only team, received a score of **28.5**, honestly higher than I thought possible. They allocated **$150** to the **GiveWell Maximum Impact Fund**.
2. **Ben Edelman** received a score of **25.8**. He allocated **$75** to the **Humane League**.

Three other participants received a score of over 15:

1. **simon** received a score of **21.0**. He allocated **$25** to the **Machine Intelligence Research Institute**.
2. **Adam Hesterberg** received a score of **19.5**. He allocated **$25** to the **Sierra Club Beyond Coal campaign**.
3. **Viktor Bowallius** received a score of **17.3**. He allocated **$25** to the **EA Long Term Future Fund**.

Congratulations to Scy, William, Ben, simon, Adam, and Viktor!

All right, let's take a look at what people did and how well it worked!

Round 1 analysis

Summary statistics

Recall that the score of a Round 1 entry is a weighted average of the probabilities assigned by Round 2 participants to the entry being real (i.e. truly random). The average score was **39.4%** (this is well below 50%, as expected). The median score was **45.7%**. Here's the full distribution:

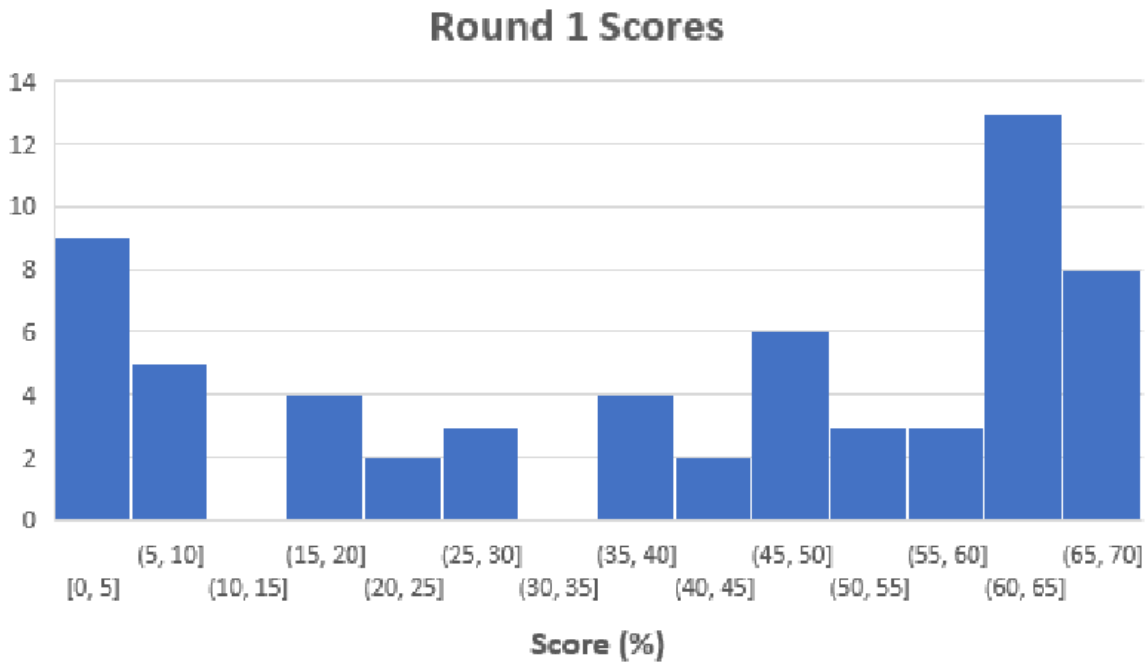


Figure 1: Histogram of Round 1 scores

Interesting: the distribution is bimodal! Some people basically succeeded at fooling Round 2 participants, and most of the rest came up with strings that were pretty detectable as fakes.

Methods

I asked participants to describe the method they used to generate their string. Of the 58 participants who told me what they did with enough clarity that I could categorize their strategy:

- 14 participants used a **memory-based** strategy. That is, they used a poem or text they had memorized, or digits of numerical constants, or their friends' birthdays, to generate their random numbers. The average score for strings in this category was **47.3%** (above average). Jenny, our Round 1 winner, used this strategy: she recited the poem "Renascence" and took the parity of the number of letters in the last word in each line.
- 19 participants used their **brain** as a built-in source of randomness. This meant things like just using their intuition to come up with 0s and 1s, or coming up with random words and taking the parity of each word's length. The average score for these strings was **41.7%** (about average).
- 14 participants used their **motor functions** as a source of randomness. This included things like "mashing the keyboard" as their free will dictated. The average score for these strings was **16.8%** (way below average).
- 11 participants used some sort of **mix** of these strategies. For instance, maybe they generated some bits from a poem and some bits using their intuition. The average score for these strings was **62.0%** (substantially above average).

Although I didn't participate, the strategy I thought to use was a mix of memory-based and brain-based. Specifically, I would have taken the digits of Pi as one source of randomness, used my brain to come up with random-ish bits, and then taken the bitwise XOR. (I'm not totally sure I would have been able to come up with 150 digits in 10 minutes with that method, though.)

Common pitfalls

By far the most common mistake made in Round 1 was not having sufficiently long (or sufficiently many) **runs**, i.e. consecutive zeros or consecutive ones. There is only a 0.5% chance that a random 150-bit string will have no run of length 5 (i.e. 5 zeros or 5 ones in a row); in comparison, 10 of the 62 strings submitted in Round 1 had no length-5 run. Many strings with insufficient runs were generated by either intuition or motor functions, though a few were not. An instructive example:

> *I took the opening lyrics from Hamilton, which I have in my head, and followed them letter by letter. Each letter later in the alphabet than the previous letter got a 0, while each letter earlier in the alphabet than the previous letter got a 1.*

This generated the following string (Round 2 ID #80):

> *001010101001010011100110110100110110100111010100110001101110101100010001001101001111011001010101011001010101100010101101101010110110101100110101011101*

The fact that this string has no runs of length more than 4 makes a lot of sense! After all, suppose you got four 0's in a row. That means you had a sequence of five letters, each later than the previous. The last of those letters is likely pretty late in the alphabet, which means that there's a much greater than 50% chance that the next letter will come earlier in the alphabet (so you'll write down 1).

The moral here is that there are many ways to mess up besides having a faulty intuitive grasp on randomness: you could also take a pretty good random number generator, but mess up when deciding how to convert it to zeros and ones. One other such pitfall was having a conversion method that resulted in an unbalanced string. One example:

> *I divided the alphabet into two groups of letters, then recited a poem I know by heart picking the bits based on which group each word's first letter was in.*

I'm not sure how this contestant divided up the alphabet, but their string ended up with only 57 ones. The probability of a random string being that skewed (in either direction) is just 0.3%.

Some contestants who didn't get caught for insufficient runs or imbalanced strings got caught with more sophisticated tests (see the discussion of Round 2 below).

Interesting sources of randomness

Here are some Round 1 techniques I found interesting (whether or not they did well):

> *I tilted my keyboard so zero was "right and down" and 1 was "up and left" and tried to play sections of DDR songs from memory. I let "jumps" across 1 and 0 hit at the same time and left the order to randomness.*
>
> ***Jon***

> *I moved around from state to state on a mental map of the United States. Move from state 1 to state 2. Convert the first letter of each state to a number 1-26. Write a 1 if state 2's number is in the thirteen above state 1's, wrapping around from 26 to 1, write a 0 otherwise. Repeat 150 times. My map contained mistakes, and I took multiple different routes around it.*
>
> ***Multicore***

> *I listed the birthdays of people by how well I know that person, and then took the month (if between 1 and 8) and the day (if between 1 and 16).*
>
> ***Anonymous***

A nearly perfect source of randomness, if you know enough birthdays! (This person didn't; they said that they then used song lyrics.)

> *I used the letters of a [poem] that we had to memorize in elementary school. I went through all the letters in my head. For every two consecutive letters x,y such that x precedes n in the alphabet and y does not, I wrote down 0. If y precedes n in the alphabet and x does not, I wrote down 1. Otherwise, I skipped the letters.*
>
> ***Anonymous***

This seems to me like a great way to extract uniformly random bits from a text (though I'm not sure why they didn't just put down 0 or 1 based on which of x and y came first).

Round 2 analysis

In Round 2, I gave participants the 62 strings generated by humans in part 1 ("fake" strings) alongside 62 strings random generated by a computer ("real" string), all in a random order,

and asked them to assign each string a probability of being real. You can see all 124 strings here.

There's a lot to say about Round 2. Let's dive in!

Fake strings were systematically noticed

Figure 1 showed the distribution of Round 1 scores: the average probability assigned by Round 2 participants to each fake string being real (weighted by Round 2 score). What if we similarly assigned scores to *real* strings?

Figure 2: histogram of probability assigned by a consensus of Round 2 participants to strin being real. Real and fake strings correspond to the orange and blue bars, respectively.

As you can see, real strings got systematically higher scores than fake strings, by a lot. Only 5 of 62 real strings were judged to be more likely to be fake than real, compared to a majority of fake strings. That's pretty cool (even if expected).[1]

What, more concretely, helped people tell real and fake strings apart? Contestants used many methods, but the vast majority of the mileage came from looking at runs (consecutive 0s or 1s): as I mentioned, many fake strings failed to have sufficiently long runs (though in some cases people overcompensated and made their runs too long).

But "looking at runs" alone isn't enough to get a good score. Let's say you find a string that has one run of length 5 and no runs of length 6. That's mildly suspicious, but how do you turn that into a *probability*? Converting a qualitative amount of "suspicion" into a probability was perhaps the most important part of the contest to get right. To understand why that was so important, let's take a look at the scoring system used for Round 2.

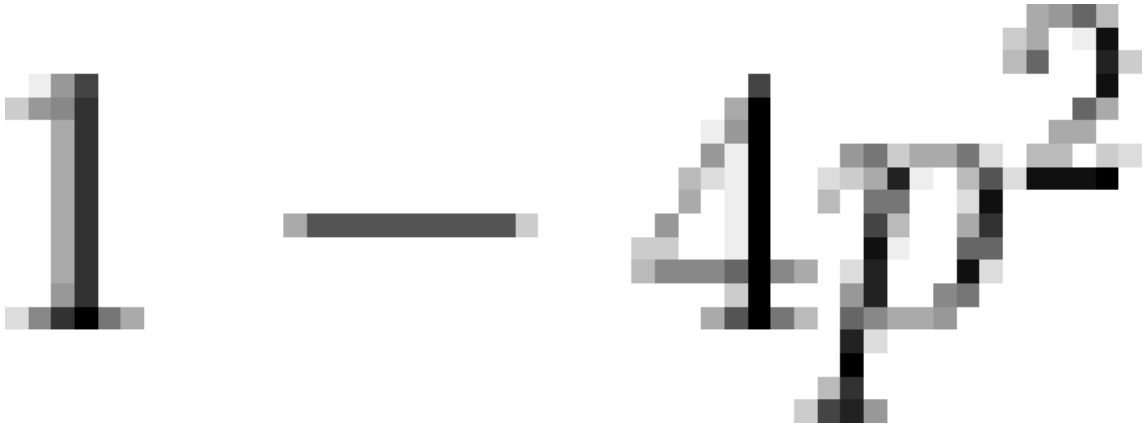
Scoring system

Slightly paraphrasing/rearranging my Round 2 post:

> *Let's say you submit a probability p for a string. If the string is **real**, your score will be*

$$1 - 4(1 - p)^2$$

> *If it is **fake**, your score will be*



*So you get a score of 1 for a string if you're completely right and -3 if you're completely wrong. Your total score will be the sum of your scores for all the strings. If you're going for maximizing your expected score, this scoring system incentivizes you to be honest . Note that **your score will be 0 if you say 50% for every string**.*

(For those of you familiar with scoring rules, this just means I was using Brier's quadratic scoring rule, scaling it so that a totally uninformative forecast would result in 0 points.)

I also didn't catch that string #106 (which was fake) had a 9 in it; I ended up asking everyone to put down 0% for string #106. This means that everyone had the opportunity to say 0% for #106 and 50% for everything else, for a score of 1.

The other benchmark I set was 15, which was my estimate for what a "good" score would constitute. As I mentioned in the prizes section, five of the 27 Round 2 contestants cleared this bar.

Before going on to the next section, you might want to make a guess about the distribution of scores!

Summary statistics

Here's the distribution of Round 2 scores.

Figure 3: Histogram of Round 2 scores. Red = below 0, green = above 15.

Scores in the red part were **negative**: participants with these scores would have been better off saying 50% for everything. Scores in the green were the ones that met my "good score" benchmark.

The average score was -5.5. The median was -1.4. **A majority of scores (14 of 27) were negative**. This means that most participants would have gotten a better score if they had said 50% for every string!

The reason for this is **overconfidence** on the part of Round 2 participants. Proper scoring rules punish overconfidence pretty harshly. For example, if a contestant managed to classify 70% of strings correctly (better than almost anyone actually did) but was 90% confident of their classification for each string, their score would have been 0.[2] Let's take a closer look.

Basically everyone was overconfident

One nice thing about proper scoring rules is that *you can use people's answers to infer what score they expected to get*. For example, let's say someone submitted 70% for one of the strings. That means they think there's a 70% chance of the string being real, in which case they'll get a score of

$$1 - 4(0.3)^2 = 0.64$$

and a 30% chance of it being fake, in which case they'll get a score of

$$1 - 4(0.7)^2 = -0.96$$

That means their *expected score* for that string is

$$70\% \cdot 0.64 + 30\% \cdot -0.96 = 0.16$$

You can calculate an expected score for every string and add those up to find the total score that the participant expected.

If the previous paragraph didn't make sense, here's a simplification: you can tell what score someone expected to get based on how confident their answers were (how close to 0% or 100%). The more confident someone's answers, the higher the score they expected to get.

Of course, someone's *true* score may differ a lot from their expected score if they aren't calibrated. If someone is overconfident, their actual score will be below the score they expected to get. If they're underconfident, their actual score will be above their expected score. So we can figure out whether someone was over- or underconfident by comparing what score they expected to get to their actual score. Here's a plot of that.

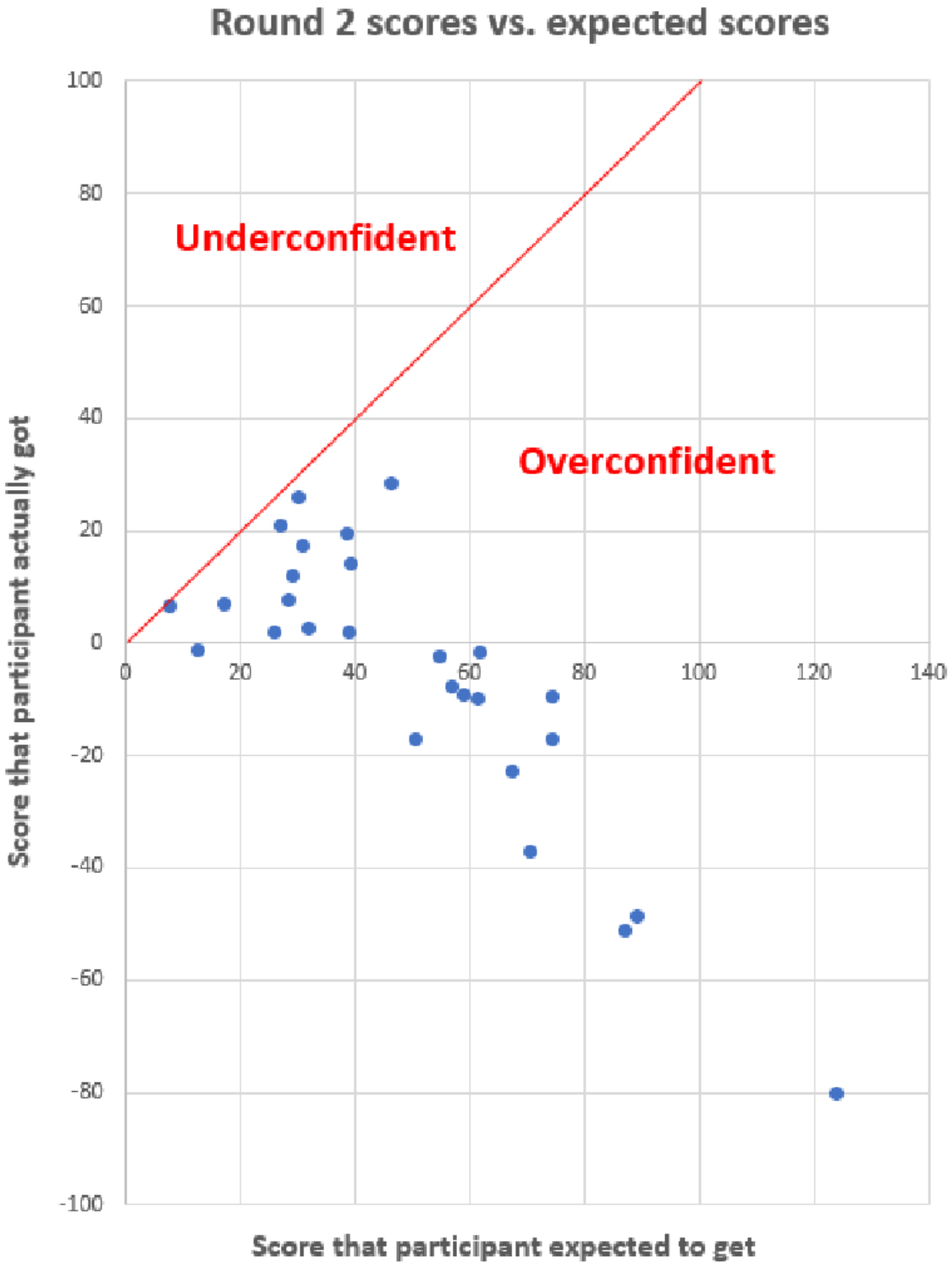


Figure 4: overconfidence in Round 2. Distance from the red line roughly corresponds to amount of overconfidence.

Every single point was below the red line, meaning that every single participant was overconfident (though three were close enough to the line that I'd say they were basically calibrated).

In fact, a nearly perfect predictor of whether someone would get a positive score or a negative score was whether they *thought* they were going to get a score above 50 or below 50. **Everyone who thought they were going to get a score above 50 got a negative score. With one exception, everyone who thought they were going to get a score below 50 got a positive score.** This is pretty remarkable!

One concrete way to measure overconfidence is "how much squeezing toward 50% would the participant have benefitted from". For example, squeezing predictions halfway to 50% would mean that a 20% prediction would be treated as 35% and a 90% prediction as 70%. We can ask: **What percent squeezing would optimize a participant's score?**[3]

For a perfectly calibrated forecaster, the optimal amount of squeezing would be 0%. But everyone was overconfident, so everyone would have benefited from some amount of squeezing. How much squeezing?

Figure 5: histogram of optimal amounts of squeezing probabilities submitted in Round 2 toward 50%

The range of optimal squeezing amounts went from 7% (achieved by Ben, who got 2nd place) to 82%, with an average of 47%. So on average, it would have been best for participants to squeeze their probabilities about halfway toward 50% before submitting. (For comparison, the winning team would have benefitted from a 19% squeeze.)

The next chart shows participant scores as a function of how overconfident they were.

Figure 6: Round 2 score versus amount of overconfidence (measured by optimal amount of squeezing toward 50%)

This plot shows that being calibrated was crucial to getting a good score. But calibration wasn't everything: Ben submitted the most calibrated entry (the red point) but got second place, while the winning submission (green), submitted by Scy and William, was only the fourth most calibrated.

So what did the winning team do so well?

Before we answer this, it's instructive to consider the second most calibrated entry, which is the yellow point in the previous chart. This entry got 10th place out of 27 — better than average, but not by much.

What happened? Out of 124 probabilities, this contestant submitted 50% 114 times, 10% 6 times, and 0% 4 times. (They went 4/4 on the 0% entries and 5/6 on the 10% ones, though the one miss was bad luck — I'll touch on that later.) This contestant did fine, but didn't do *great* because of poor **classification**, i.e. an inability to systematically sniff out fake strings.

One rudimentary measure of classification is what fraction of strings were "called" correctly. That is, one could look at how well contestants would have done if I had simply awarded each contestant a point for every real string to which they assigned a probability greater than 50% and every fake string to which they assigned a probability less than 50% (and half a point if they said 50%). Here's a plot of contestant scores vs. the fraction of strings they called correctly.

Figure 7: Round 2 score (y-axis) versus percentage of strings classified correctly (x-axis)

By this metric, Scy and William were the best at classification by a substantial margin. The 2nd and 3rd best entries by classification got 3rd and 4th place, respectively, in Round 2, behind Scy/William and Ben.

The plot below is meant to illustrate in more detail how exactly Scy and William beat out Ben despite being less calibrated.

Figure 8: probability assigned to each bit string being real by Ben (most calibrated; 2nd pla on the x-axis, versus by Scy and William (1st place; 4th most calibrated) on the y-axis. Blue points are real strings; orange points are fake strings.

Here, every point corresponds to one of the 124 strings. Fake strings are orange; real ones are blue. The x-value is the probability Ben (the most calibrated, 2nd place contestant) assigned to the string being real; the y-value is the probability assigned by Scy and William (1st place, 4th most calibrated).

Here’s the same chart again, this time with emphasis added to two parts of the chart.

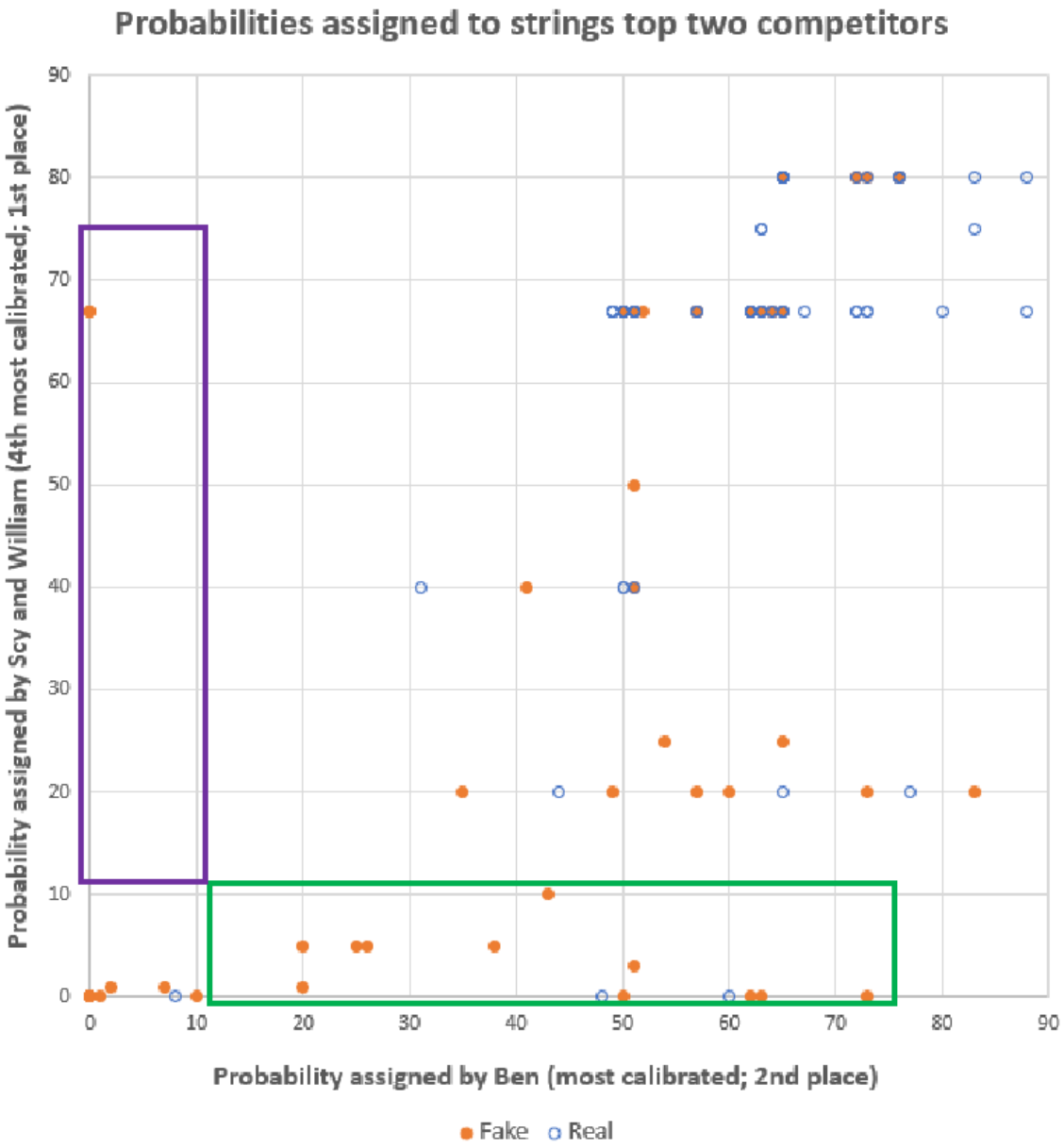


Figure 9: same as Figure 8, but with rectangles indicating strings Scy and William were confident were fake but Ben was not (green) and strings Ben was confident was fake but Scy and William were not (purple)

The green rectangle contains all the strings that Scy and William said were very likely fake but Ben did not. There are 13 such strings, 11 of which were in fact fake.

The purple rectangle is all the strings that Ben said was very likely fake but Scy and William did not. There are 3 such strings (you only see one dot because they're on top of each other). One of those was Ben's string. (The other two had long substrings of the digits of Pi mod 2, which Ben looked for but Scy and William didn't.)

So basically, even though Scy and William were less calibrated than Ben (three strings they assigned probability 0 to were real), they discerned fake strings better. And ultimately this made them win, earning 28.5 points to Ben's 25.8.

This is in part why I used the quadratic rule instead of the log rule for scoring: the log rule punishes miscalibration at the extremes really harshly, and I didn't want someone who was slightly less calibrated but much more discerning to lose. Had I used the log rule (and rounded 0% probabilities of real strings to 1%), Ben would have narrowly won.

Calibration vs. classification

Did more calibrated contestants also classify more strings correctly?

Figure 10: optimal amount of squeezing toward 50% on the x-axis (note that the axis is reversed so that "more calibrated" is toward the right) versus percentage of strings classified correctly on the y-axis

The answer is yes, somewhat. The r^2 of the relationship here is 23%, meaning that contestants' calibration explained 23% of the variance in in how well they "called" strings. This makes sense: the more effort you put (and the more experience you have with this, etc.), the better you'll do on both fronts.

(By the way, it's pretty nice how the top three finishers formed the calibration-classification Pareto frontier!)

If you're interested in reading more about the calibration-classification dichotomy, I also have a more detailed post about it.

Okay but what strategy did the winners use?

I'll quote Scy and William's description of what they did verbatim (modulo small stylistic edits), alongside clarifying comments.

> *We assigned 0 or close to for ~20 numbers that looked "obviously fake", then added another ~10 to the 0 list with:*
>
> *– Binomial CDF (too big a skew got a low number)* (I think this means if the string had too many 0s or 1s -Eric)
>
> *– Plotting the mean and standard deviation of contiguous bits with a bit string* (i.e. the lengths of runs -Eric) *and superimposing it on a plot where 360 pseudorandom bitstrings of my own devising* (i.e. computer-generated random stirngs -Eric) *were plotted, and eliminating outliers*
>
> *– Count the distribution of length-2 bitstrings within each string and figure out the difference between the most and least common (should be fairly low for random strings – the difference topped out at ~30 for most runs for ~124 randomly generated strings, so we gave a low probability to anything > 30)*
>
> *– gzip'ed them and flagged one that was unusually compressible*
>
> *For the remaining, we couldn't figure out a satisfactory way to formalize the probability that they were fake. We ended up assigning 0.67 (62 / 62 + un-eliminated human generated strings) to them by default, and using numbers-out-of-ass to dock or bump entries visually, based on how far they were from the center of mass of our pseudorandom bitstrings plotted on the mean + standard deviation 2D distribution.*

Scy and William kindly shared with me their plot of the mean and standard deviation of run lengths:

> *Figure 11: the average length of runs of the same digit consecutively on the x-axis, and the standard deviation of these values on the y-axis. Red points are the 124 given strings (with ID numbers written). Blue points are 360 computer-generated random strings. See here for a larger version.*

(Click here for a larger version of the plot.) In this plot, red points are the 124 strings given to participants (along with their ID numbers), while blue points are computer-generated random strings. The x-axis is the average length of a run; the y-axis is the standard deviation of run lengths.

The idea is pretty simple: blue (truly random) points formed a pretty narrow cloud on this plot. A red point that was far away from this cloud was therefore likely to be fake.

Many people looked at average run lengths, but not many looked at the *standard deviation*. This helped them catch a few fake strings that the consensus failed to catch, including strings 27 and 124 (whose standard deviation was too small), and 81 and 105 (too large). I haven't done a deep dive, but it looks to me like this strategy helped set Scy and William's entry apart.

What other strategies did well?

I'll again quote the other top contenders' strategies verbatim. First, Ben (2nd place, most calibrated):

> *On a hunch, I looked at the number of times a 1 followed a 1 or a 0 followed a 0 in each string ("runs"). This empirical distribution had **way** higher variance and somewhat lower mean than would happen by chance. So I used Bayes' rule to update my prior probabilities, with likelihoods coming from a smoothed version of the observed runs distribution (a sketchy technique, but perhaps not too sketchy). This was my main source of information.*
>
> *Separately, I searched for various substrings within the strings: long stretches of 0s (and 1s), parities of digits of pi and e forwards and backwards, alternating 1s and 0s, alternating pairs of 0s and 1s, and a few others. A few strings had unrealistically long substrings of these forms, so I gave them 0 probability of being random.*
>
> *I then applied normalization to ensure the sum of predictions was 124.*
>
> *I also found that the variance of the empirical distribution of Hamming weights was quite high (but not nearly as high as the runs distribution variance).* ("Hamming weight" means "number of 1s" -Eric) *This seemed to be explained mostly by the tails, so I applied some semi-ad-hoc scaling to my predictions for strings with particularly high or low Hamming weight, and applied normalization again.*

These methods are very solid, but nothing here is particularly *special* in terms of doing things no one else did. My best guess, then, is that Ben had great execution. That is, his Bayes updates (which many people tried to do with varying degrees of success) were roughly of the correct size. This theory is supported by the fact that Ben was well-calibrated.

There is something I found really interesting about Ben's submission, which is that his probabilities went as high as 88%. Ben's was an outlier among the best entries in this sense: most of the entries that did best had a maximum probability in the 60-80 percent range. This confused me: could you really be 88% sure that a string was random, when you know you're assigning something more like 60-65% to a *typical* random string? Can a string look *really random* (rather than merely random)? This sort of seems like an oxymoron: random strings don't really stand out in any way sort of by definition.

In response, Ben told me:

> *It's because those strings had run counts that were uncommon among (a smoothed histogram of) the given strings but not too uncommon among the random strings.*

But the frequency of particular run counts should be at most twice as rare in the 124 Round 2 strings and in a sample of random strings, since half of Round 2 strings are random. So is there overfitting going on? Ben's submission was well-calibrated, so perhaps not. I remain confused about this.

Next we have simon (3rd place):

> *I used LibreOffice calc – I checked for extreme values of: (1) total number of 1's (2) average value of first XOR derivative (don't know if there's a usual name for it)* (I think "XOR derivative" refers to the 149-bit substring where the k-th bit indicates whether the k-th bit and the (k +1)-th bit of the original string were the same or different. So this is measuring the number of runs. -Eric) *(3) average value of second XOR derivative and (4) average XOR-correlation between bits and the previous 4 bits* (not sure what this means -Eric)*. In each case, I checked to see if the tails were fatter than expected for random chance and penalized strings in the tails to the extent that was the case (and only the amount of tail that was in fact fatter than expected). Then I multiplied the resulting probabilities obtained for each statistic together and rescaled for the final answer... Only a minority of the strings were sufficient outliers in any test to receive a penalty, which is why most strings have the same 64 percent (rounded from 63.6 percent) estimate.*

I'm curious how much, if any, of simon's success came from (3) and (4). I'm not sure what (4) refers to, but (3) was not commonly done.

EDIT: See here for simon's explanation of the XOR-correlation, as well as an answer to my curiosity. The answer is: they both helped a decent amount!

Next we have Adam (4th place).

> *I checked each of the following: length of runs of consecutive 1s, same for 0s, squared distance of the distribution of substrings of length 6 from the uniform distribution of substrings of length 6, same for 5, same for 1 (didn't bother with other numbers), maximum length of a substring shared with another answer, and maximum-length shared substring with any mapping of digits 0-9 to 0-1 applied to digits of pi. I then multiplied a running odds ratio by the odds that an answer is that extreme relative to the expected most extreme value for that statistic among 124 random strings.*

I'm curious about the particular choices of 5 and 6. Would Adam have done better by incorporating evidence from every length?

Finally, Viktor (5th place):

> *I made a simple program that counted the sequences of consecutive numbers. For example, the sequence 0010111001100 Would generate:*
>
> *Zeros: 7. Sequences with 3 consecutive numbers: 1. Sequences with 2 consecutive numbers: 4. Sequences with 1 consecutive number: 2.*
>
> *For example if an entry had more than 35 sequences with just 1 consecutive number (meaning alone), it was very likely humanly generated. If the number of zeros was under 60 or over 90, that was also an indication the sequence was fake.*
>
> *Then I used my gut feeling to give odds to "odd seeming" entries that might have had abnormally many sequences with 5 consecutive numbers or something. If something was just a little odd, I would just give it slightly lower probability of being truly random. I also made a blind guess that entries with 8 or 9 as its longest sequence was slightly more likely to be truly randomly generated. I still don't know if that assumption was correct.*

> *I entered all my guesses into an Excel spreadsheet, and there I saw my average guess was like 35% chance of being truly random, so I added a 1 too all numbers, and multiplied with something like 1.4, so the average would be close to 50%.*
>
> *I think I might have made a mistake by multiplying by a constant, because if I had two numbers, let's say 40 and 50, if I multiply by 1.4, the difference between them increases 16%* (not sure where the 16% is coming from -Eric)*, which was probably a bigger difference than what I originally might have believed.*

It's pretty interesting that this did well, given the reliance on gut feeling and the crude 1.4x scaling at the end, but it seems to have worked out! I wonder if a good gut feeling outperforms all but the best automated methods because automated methods are pretty easy to mess up.

Other interesting strategies

Most people did some variation of "I looked at runs". Relatively few people did anything that surprised me. My favorite of these non-orthodox methods:

> *Another thing that I tried was to plot the strings with "turtle graphics", so you can visualize each string "fingerprint". For this to be done, at first I turned the turtle 90º degrees left if the digit is a 0 and to the right if it is an 1 and advanced one step per digit. Strings with a lot of 0/1 groups tend to curl and pack, while strings without those groups tend to go away. (Strings 5 and 66 are nice to be see because they are clearly different from each other.)*
>
> ***Jorge Tornero***

Cool! I bet this visualization makes real and fake strings look pretty different.

Another interesting method:

> *I had some linalg code lying around, so I just formed small matrices over* F_2 (the field of 2 elements -Eric) *out of the sequences, and calculated their ranks to tease out linear-ish dependencies.*
>
> ***Fishlips***

Interesting, I wonder how much you can get out of just looking at linear dependencies like this.

Finally:

> *I calculated the relative frequency of various substrings of equal length — e.g. "0101" vs "1111" — up to a length of 10, and compared the squared error of the competition strings to that of a population of 1k random strings.*
>
> ***Measure***

Interesting: Measure looked at relative frequencies of *all strings* of a given length, rather than only looking at runs. I wonder how much that helped.

Final comments

Did participants who did well in Round 2 do well in Round 1?

Interestingly, the was no correlation between Round 1 and Round 2 scores (though as a group, people who *participated* in Round 2 did quite well in Round 1).

Figure 12: Scores in Rounds 1 and 2 of the 23 contestants who participated in both rounds

A bit of bad luck for Round 2 participants

Figure 2 showed that two of the random strings looked pretty fake. These were #121 and #122.

#121:10111111101010110011000101011111111111001111111111111000001000110100110110001011100101011111011101111011000000000110110100101011010101111111111010101

#122:10101100111000111101001011011101101100011110100110011111111111011100011111101101110001110111111111111011100101001111101100011000111011101001110000110

What made them look fake is that they had, respectively, 56 and 53 zeros — much less than half of 150 (3.1 and 3.6 standard deviations too few zeros, respectively). The probability of having as unbalanced a string as #121 in a sample of 62 random strings is 12% (not too unusual). The probability of having as unbalanced a string as #122 is just 2%.

Given that balance between zeros and ones is one of the most basic things to look at, Round 2 participants got *unlucky* in that they were fed a string that looked much faker than the fakest real string should look — in fact, arguably two such strings.

Contestants *justifiably* gave probabilities near 0 for both strings. This cost participants a few points. This also means that the previously discussed overconfidence is *slightly* an artifact of this bad luck. If not for this, three participants (the top three by calibration) would have been essentially correctly calibrated; everyone else would have still been overconfident, though.

Raw data

- The 124 strings (62 fake and 62 real) are available here. The answer key is on the second sheet.
- Here is a (mostly anonymized) spreadsheet containing all Round 2 entries, as well as some summary statistics for each participant. I think the data is fascinating (as you can probably tell from the length of this post), and I'd be thrilled if people played around with the data and found other interesting things!

1. This is admittedly a little sketchy, since I'm determining the weights based on ability to discern fake strings. However, I think there are enough total strings that this causes only a small amount of bias.

2. If I had chosen the log scoring rule instead, I would have been punishing overconfidence even more harshly!

3. Interestingly, there is a nice pictorial representation of the optimal amount of squeezing: in Figure 4 (repeated below), it is the distance from the point to the red line, divided by the point's x-value, and then divided by the square root of 2. Alternatively (using the red point below as an example) it is the ratio of the lengths of the green segments.

Figure A: the optimal amount of squeezing for the submission represented by the red point is equal to the ratio of the lengths of the top and bottom green segments.