Tia Jin

CS109

Extra Credit Assignment

During my commutes to class this summer, I often played Spotify on shuffle and more than often I felt as though the music was not truly random and seemed to be repeatedly playing songs from the same artists. Perhaps general users of Spotify have experienced this at times as well. After we learned the approaches of repeated simulation and observing outcomes, I decided to create two algorithms to test the difference between a truly random shuffle and a "human-friendly" shuffle.

Setup: To start, I created a balanced synthetic playlist of 50 songs (ex. ["Artist A", "Artist B"]). For the random shuffle, I used the `random.shuffle()` function in Python to scramble the playlist and count the number of same-artist adjacent pairs. An adjacent pair is counted when two consecutive items in the playlist are equal. I then calculated the average number of adjacent pairs when performing this across 10,000 trials. I ran this 10K simulation many times and some example values returned were: 9.0239, 9.0018, 8.9556. These empirical values are very similar to our theoretical expected value for number of consecutive artist pairs.

Let's define an indicator variable.

$$\text{I\_\{i\}} = \begin{cases} 1 \; if \; \text{song } i+1 \; has \; the \; same \; artist \\ 0 \; \text{otherwise} \end{cases}$$

$$X = I_1 + I_2 + \cdots + I_{49}$$

The probability that the next song has the same artist as the current song, given our setup of 50 songs where each artist has 10 songs. If the current song is by artist A, there are 49 songs left total.

$$E[I_i] = P(I_i = 1) = \frac{9}{49}$$

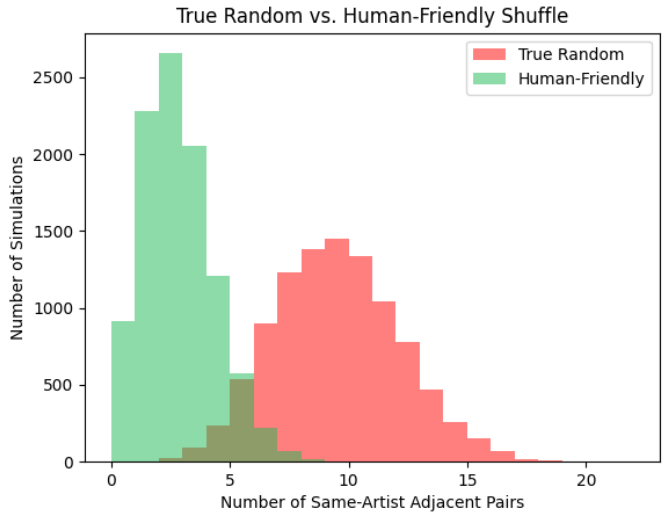
Using the linearity of expectation:

$$E[X] = \text{E[I\_1]+}\ldots\text{+E[I\_49]} = \sum_{\text{i=1}}^{49} \text{E[I\_i]} = 49\left(\frac{9}{49}\right) = 9$$

In contrast, the "human-friendly" algorithm I made greatly reduces the probability of consecutive songs by the same artist by using weights. For example, as we iterate, if the remaining playlist is ["A", "A", "B", "C"], and the previous artist was "A", the weights become

[0.2, 0.2, 1.0, 1.0], so that each A will receive 1/5 the selection weight of every other artist when we run np.random.choices(). I repeated this multiple times for the same number of simulations as our random shuffle and the average was often close to 2.3. This indicates that the simulation reflected our theoretical expectation, and that the man-made shuffle suppresses repetitions that we observe are actually mathematically more frequent than we thought.

Results are illustrated in the histogram below.



From this diagram you can see the full distributions of X (the number of same-artist adjacent pairs) for the two shuffles. The true random shuffle is centered near the theoretical expectation of 9 consecutive pairs, while the human-friendly one is concentrated around 2-3 pairs. Relating to the graphs we saw for sum of two dice in class, here we can also see that there are many more shuffle arrangements that can result in X being a moderate number, than more extreme values of X. Although each sequential ordering of the playlist is equally likely, the values of X are not equally likely.

Next, I used the Monte Carlo technique to further demonstrate the nuances between the two kinds of shuffling. I defined an event X >= 10 where the variable X represents the number of consecutive artist pairs. After counting the times the number of pairs was greater than or equal to 10 across our simulations, we get values usually approximate to these:

$$\mathrm{P}(\mathrm{X}\geq 10)\ \text{for true random}\ \approx 0.4234$$

$$\mathrm{P}(\mathrm{X}\geq 10)\ \text{for human}-\text{friendly}\ \approx 0.0001$$

It turns out that out of 10,000 true random shuffle simulations, roughly 42.34% (for this particular run) had at least 10 adjacent same-artist pairs, whereas for the human manipulated

algorithm there is nearly none. Similarly, I compare the calculations for the probability of having at least 3 consecutive songs by the same artist and it was around 73% for the random shuffle and about 10% for the human-friendly shuffle. This helped answer my question about why it sometimes felt like there was so much repetition in Spotify shuffled playlists. The experiment results show that under true randomness, having a streak is not uncommon at all.

Moreover, I decided to find the conditional probability of the next song being by an artist A, given that the current song is by the same artist. Example experiment values below:

True random shuffle:

$$P(next = A|current = A) \ = \ 0.18316$$

Human friendly shuffle:

$$\mathrm{P}(next = A|current = A) \ = 0.04796$$

The results for the random shuffle can be confirmed by a theoretical calculation as well. We can write out the conditional probability like so:

$$P(\text{next} = A \mid \text{current} = A) = \frac{P(\text{current} = A \text{ and next} = A)}{P(\text{current} = A)}$$

We get the numerator by using counting: consider permutations without replacement

$$P(\text{current} = A \text{ and next} = A) = \frac{10 \cdot 9}{50 \cdot 49}$$

$$P(\text{next} = A \mid \text{current} = A) = \frac{\frac{10 \cdot 9}{50 \cdot 49}}{\frac{10}{50}} = \frac{9}{49} \approx 0.1837$$

This value matches what I observed from running the Python program. Under real randomness, there is in fact about an 18% probability of hearing the same artist on the next song. For the other shuffle, we forcefully manipulated it to avoid repeats, so that significantly reduced the probability of observing consecutive artists in a playlist.

All in all, this experiment illustrated that true randomness produces natural clustering and more repetition than what people intuitively expect. The contrast between the values from the two shuffles demonstrated how a manipulated algorithm that intentionally avoids repetition and produces an artificially more balanced playing order is surprisingly less mathematically random. This was an interesting insight that reminded me of the Galton board and two dice examples we covered in class, and I can appreciate how this behavior of true randomness has some philosophical nuance in the real world.

*Note: Please refer to the other attached file for the code. Thanks!