PHSX815 - Project 1

Ryan Scott

March 1, 2021

Do you want to play a game?

This project is based on a game of chance. It utilizes a fictional deck of n cards, each given a value ranging from 1 to n. The game works like this: two players are dealt from a shuffled deck of cards. The cards are then played against each other sequentially, each duel is called a "hand". As players reveal their cards, the winner for each hand is determined by whose card value is greater. For example, if Alice plays a 4 and Bob plays a 6, Bob wins the hand. After all of the cards have been played, the total number of wins for Alice is recorded.

The goal of this experiment is to determine whether or not cheating can be detected reliably over a large number of games. For this we use a Z test statistic, because our data closely resembles that of a normal distribution.

Alice, the cheater

Alice, however, has decided that she wants to cheat. She can use the "gimme" command to use a cheater deck of cards that she brought with her. This deck of cards will always have values higher than the fair deck that Bob *thinks* Alice is using. By using the cheater cards, she takes a number of cards out of her cheater deck and replaces that number of cards from her fair deck based on how they had been shuffled and distributed. She does not know which cards she is replacing.

Now that Alice has cheated, she has guaranteed herself winning all of the hands that she cheats with. We will see that this modifies the overall distribution of data. We make the assumption that Alice has chosen to play fair most of the time and is not replacing her entire deck with cheating cards.

The distribution of data

In this project, we are witness to two distributions. In any given hand, Alice will have a Bernoulli distribution if the hands are fair, and a Binomial distribution (where p=1) if she has chosen to cheat. More specifically:

$p_0 = .5$

$p_1 = 1(\frac{g}{n}) + .5(1 - \frac{g}{n})$

Where g is the number of cards Alice cheats with, and n is the total number of hands in any one game. For our analysis, we are less concerned with the distribution of data related to hands. Our interest lies in the distribution of data the games that she wins.

As we accumulate data, we see that our data resembles a categorical, or multinomial, distribution. Over many many games, the data approaches a normal distribution. If we assume that only fair games are played, the distribution will be centered at a 50 percent win rate, as expected. The number of games won will be a maximum at n/2 (n being the number of hands/game). In other words, if 50 cards are used, we would expect to see her win 25 games, plus or minus some standard deviation.

When Alice chooses to cheat, her normal curve begins to shift to the right. She is more likely to win more games than if she had been playing fair (See Figures 1 and 2). Because we can obtain all of the data (average values, standard deviations) associated with the fair and cheated games, we can apply a Two-Sample Z Test statistic to our data.

The Two-Sample Z Test

The Two-Sample Z Test, or Z test, is a straightforward method of determining a likelihood estimate. It is ideal for large sample sizes and distributions that are normal. In our case, we have the power to iterate over hundreds and thousands of trials, making the simplicity of the Z test undeniably attractive. Other methods of obtaining a likelihood require a "choose" function, which requires more computing power than some computers (see: mine) were able to provide. The product of factorials to obtain a probability is problematic and over large amounts of data, as simplified test statistic is desired to save precious computing power.

In this experiment, we use the Two-Sample Z Test, which assumes we have information from both of our distributions. The Z test is as follows:

$$Z = \frac{\mu_0 - \mu_1}{\sqrt{\frac{\sigma_0^2}{n_0} + \frac{\sigma_1^2}{n_1}}} \tag{1}$$

Where μ_i are the average values of the data, σ_i are the standard deviations, and n_i are the number of games played. This Z value is shown on the title of each plot. Once we have the Z value, we can compare it with the Z value table (see Figure 4) to determine the probability that this data matches our fair game. We have the resources to test if this Z test is viable when put under pressure.

We will examine three scenarios: one with an obvious cheater, one with a subtle cheater, and one with a fair game. We can iterate each of these over 100 games to determine a Z value. These games are Figures 1, 2, and 3, respectively. These clearly demonstrate that even with 1 cheater card and a Z score of -1.79, we see .0367 or 3.67 percent. This suggests that we were not using the standard cards.

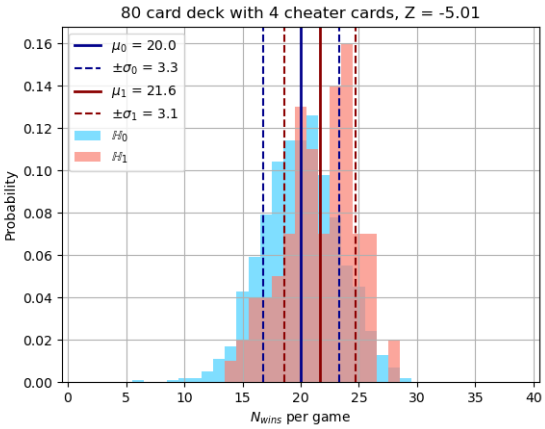

Figure 1: The obvious cheater

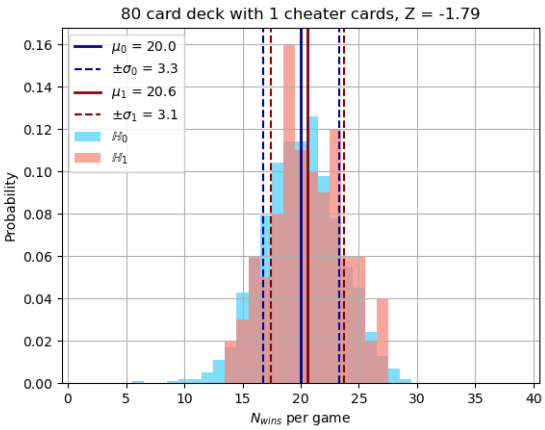

Figure 2: The subtle cheater

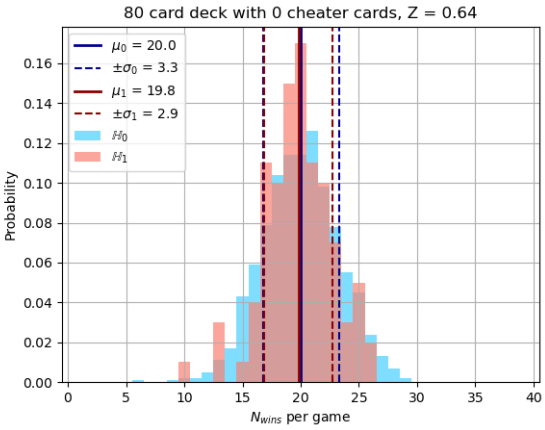

Figure 3: The fair player

z	.00	.01	.02	.03	.04	.05	.06	.07	.08	.09
–3.4	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0002
–3.3	.0005	.0005	.0005	.0004	.0004	.0004	.0004	.0004	.0004	.0003
–3.2	.0007	.0007	.0006	.0006	.0006	.0006	.0006	.0005	.0005	.0005
–3.1	.0010	.0009	.0009	.0009	.0008	.0008	.0008	.0008	.0007	.0007
–3.0	.0013	.0013	.0013	.0012	.0012	.0011	.0011	.0011	.0010	.0010
–2.9	.0019	.0018	.0018	.0017	.0016	.0016	.0015	.0015	.0014	.0014
–2.8	.0026	.0025	.0024	.0023	.0023	.0022	.0021	.0021	.0020	.0019
–2.7	.0035	.0034	.0033	.0032	.0031	.0030	.0029	.0028	.0027	.0026
–2.6	.0047	.0045	.0044	.0043	.0041	.0040	.0039	.0038	.0037	.0036
–2.5	.0062	.0060	.0059	.0057	.0055	.0054	.0052	.0051	.0049	.0048
–2.4	.0082	.0080	.0078	.0075	.0073	.0071	.0069	.0068	.0066	.0064
–2.3	.0107	.0104	.0102	.0099	.0096	.0094	.0091	.0089	.0087	.0084
–2.2	.0139	.0136	.0132	.0129	.0125	.0122	.0119	.0116	.0113	.0110
–2.1	.0179	.0174	.0170	.0166	.0162	.0158	.0154	.0150	.0146	.0143
–2.0	.0228	.0222	.0217	.0212	.0207	.0202	.0197	.0192	.0188	.0183
–1.9	.0287	.0281	.0274	.0268	.0262	.0256	.0250	.0244	.0239	.0233
–1.8	.0359	.0351	.0344	.0336	.0329	.0322	.0314	.0307	.0301	.0294
–1.7	.0446	.0436	.0427	.0418	.0409	.0401	.0392	.0384	.0375	.0367
–1.6	.0548	.0537	.0526	.0516	.0505	.0495	.0485	.0475	.0465	.0455
–1.5	.0668	.0655	.0643	.0630	.0618	.0606	.0594	.0582	.0571	.0559
–1.4	.0808	.0793	.0778	.0764	.0749	.0735	.0721	.0708	.0694	.0681
–1.3	.0968	.0951	.0934	.0918	.0901	.0885	.0869	.0853	.0838	.0823
–1.2	.1151	.1131	.1112	.1093	.1075	.1056	.1038	.1020	.1003	.0985
–1.1	.1357	.1335	.1314	.1292	.1271	.1251	.1230	.1210	.1190	.1170
–1.0	.1587	.1562	.1539	.1515	.1492	.1469	.1446	.1423	.1401	.1379
–0.9	.1841	.1814	.1788	.1762	.1736	.1711	.1685	.1660	.1635	.1611
–0.8	.2119	.2090	.2061	.2033	.2005	.1977	.1949	.1922	.1894	.1867
–0.7	.2420	.2389	.2358	.2327	.2296	.2266	.2236	.2206	.2177	.2148
–0.6	.2743	.2709	.2676	.2643	.2611	.2578	.2546	.2514	.2483	.2451
–0.5	.3085	.3050	.3015	.2981	.2946	.2912	.2877	.2843	.2810	.2776
–0.4	.3446	.3409	.3372	.3336	.3300	.3264	.3228	.3192	.3156	.3121
–0.3	.3821	.3783	.3745	.3707	.3669	.3632	.3594	.3557	.3520	.3483
–0.2	.4207	.4168	.4129	.4090	.4052	.4013	.3974	.3936	.3897	.3859
–0.1	.4602	.4562	.4522	.4483	.4443	.4404	.4364	.4325	.4286	.4247
–0.0	.5000	.4960	.4920	.4880	.4840	.4801	.4761	.4721	.4681	.4641

Figure 4: Z-Score Table

The Algorithm

Our algorithm uses two python files "CardgameSim.py" and "CardgameAnalysis.py". The first file, CardgameSim.py, is designed to take user inputs over multiple variables, such as how large the deck is, how many cheater cards are used, and how many games are iterated over. The program then takes that data and puts it into a "Cards" class, which shuffles the deck using the default python library, random. We then define our gameplay.

Our gameplay first deals out the cards to players using a list of lists. We can take one of those lists and replace the number of cheater cards indicated in the "gimme" variable. These cheater cards are given values higher than those within the deck. This hand is then shuffled.

The algorithm then compares index values in the two lists and takes the difference, always using Alice's deck first. If the difference is positive, we say that Alice won the hand and the win result increases by one. This is then iterated over each item in the list of cards until all cards have been played. The result of all of this is that we have a unique distribution of data that resembles a normal curve.

Finally, this script writes each outcome onto a txt file for use in the CardgameAnalysis.py script.

The CardgameAnalysis.py algorithm first accepts user inputs similar to the previous algorithm. The file then looks at the txt files indicated by the user and splits the list into values, which are then placed into a list of outcomes. This list of outcomes gives us access to all of the necessary data for our Z test statistic. We can evaluate average value, standard deviation, and number of games played from this data. Once that dats is received, we plot both data files and indicate vertical lines for the average value and standard deviations of the data.

Conclusion

Our home-made game allows us to analyze the data of a normal distribution using the Z test statistic. The game has embedded in it two distributions. First, each hand comparison is a Bernoulli distribution. Second, each game resembles a normal distribution over many iterations. We have seen that over many games, our data easily reveals when someone has cheated. Once we have a Z score for our data, we can use a table of Z values to determine what percentile our data falls in.