Michael Chillemi

3/17/2022

Probability & Applied Stats

Professor Hoy

Explaining The Submissions

This paper is going to be explaining all the submissions pieces in project 2 bundle. First, I will be explaining the additions I made to the stats lib. One of the methods I had to add was the binomial distribution. For this method I too kin inputs as p, q, n, and y. From there I took the combination of n and y. Next, I took p and raised it to y and raised q to n – y. Once those calculations were made then you just multiply them together to get the binomial distribution. After that method was the geometric distribution method. To calculate this as inputs you need p, q and y. First you must raise q to y - 1. After you multiply p and q to achieve the geometric mean. The next distribution that was needed to be added was hyper geometric distribution. This method was not hard but just needed a couple extra steps. First as inputs you need r, y, n and N. Once we have those values you will need to take the combination of r, y and you will also need the combination of N – r, n – y. In addition, you need the total amount of combinations which is the combination of N, n. Once all those calculations are made then you will need to multiply the first two combinations and divide by the total amount of combinations to achieve the answer. Another distribution we needed to add to the stats library is the Poisson distribution. To calculate this the only inputs, you need are lambda and y. Next, we need to run a for loop going up to the value of y. In every iteration of the loop, we will raise lambda to our variable I. Then take the factorial of I and raise a variable e to lambda which was already provided to be 2.7182818. The next step would be to take the top and divide by the bottom and multiply to our e. Once that step is complete add that number every time at the end of the loop and return it when the loop is complete. Next, we needed to add Chebyshev. To complete this method, we needed to take in as input the upper bound, lower bound, mean, and standard deviation. Once we have those inputs, we subtract the mean from the upper and lower bound. Then we run a check to make sure both differences are the same which will be the within number. Once it is verified then we divide the within number to our standard deviation to get our k value. After this step we plug k into the formula which is one minus one over k raised to the power of 2. After that calculation is compete then it will return your answer. The last method was a method to take in a number and return the factorial of that number as a big integer. Basically, it was the same method I did for calculating the factorial which is recursively calling the method and adding a base case. But the only difference is I had to take the input number and convert it to a big integer which was not that difficult after reading the proper documentation. Last of this program I updated my results method to test and display the results of all the methods in the library.

The next program that was asked to be completed was the graph function program. The first file we needed to create was a plotter. The purpose of the plotter was for us to pick a function and then print out a csv file that had one column of x values and the other column of y values. The function I picked was . Once that file was created the next step was to salt the data. The next program salter would take in a csv file and salt the y values. To do this first I created a method that would fill two array list. One list having all x values and another having all y values. Once the array lists were populated then we can run a function on the y value list to salt the values. To complete this I made a for loop that would loop through the y list. At every iteration of the loop, I would have two random numbers generated one being a random number from 1-100,000. The other generated 0 or 1. If the second random number is one then the program will add the first random number to whatever value at position I. If the value is 0 then the program will subtract the random number to the value at position I. Once that step is done it will add it to a new array list and return that list. After the list is populated then the function will create a new csv file with the salted data. The next program that we needed to create was smoother. The purpose of smoothing would be to smooth out the data. To accomplish this first we must reuse the csv to array list method. Once that is complete and we have our array lists populated then we will start to smooth the data it will take in an integer which will describe the window size. The window size is the number of numbers before and after the current value you want to smooth. To smooth that data, you will take the window and get the average of the window and that will be the new y value. Once you do this it will add the new values to the list. Then finally you can print out the data to another csv file. After I created a java file that had a method to create all three csv files with 3 second delays in between each call. Last I created a tester to call my results methods.

The last set of programs we had to create was the poker hand tester. First, we created a card object. Every card object has a suite as a string and a number value represented as a int. After creating the card class, we needed a deck class. In the deck class it needed to represent a deck of cards. First, we made a method to draw a card which would take an array list remove the top element and return that card object. The next method we needed was to replicate shuffling cards. To do this I just created a method that would take in an array list of card objects as input. To complete this task, I simply called the collections sorts method to shuffle the array list. The last method I had in the java file was a print deck method which was used to test and make sure the arrays were populated properly. Next was the hand evaluator class. In this class where I created all the testing methods for each type of hand. First, I created check methods and then test methods to run the proper number of runs. The number of runs is adjustable to the user but for my testing purposes I ran each test 10,000 times independently. The test I ran are as follows one pair, three of a kind, full house, two pairs, flush, straight, four of a kind, royal flush, straight flush, and the probability of getting no pairs. I also created a test all method that tests all the probability and display the results in percent. Last I created a tester method to call my test all to display the results.

This last section of the paper will describe the extra work I put into the project two bundle. In the stats library I created a couple extra methods that handled different cases for binomial distribution. It would handle the cases if you needed to do a number greater than or equal to y. Also, in all of my programs I used java docs to explain every method. There was not much I could for extra credit on the graph function programs. But on the poker hand tester I added a couple extra methods that I tested. The first method I added was the royal flush test. This method would calculate the probability of getting a royal flush in 10,000 hands. Next, I had a method that would calculate a straight flush in 10,000 hands. Last instead of creating a whole new method to calculate the probability of not getting a pair in your hand. All I did was take the compliment of the probability of getting a pair to achieve my results. To do this I just recalled my pair test method and subtracted by it by 1.