Dominic Verrichia

Project 2 - Probability Distribution Programs with Github

The scope of this project covers the use of various Probability Distributions including Binomial, Geometric, Hyper-Geometric, and Poisson Distributions. Each has their own circumstances for proper application. The javadoc comments in the programs along with this document will further explain the application of these distributions with instructions for using the programs effectively.

The binomial distribution is applied on an experiment with identical trials that yield exactly one of two outcomes. Some common examples of this are pass/fail, hit/miss, and compliant/non-compliant. This type of experiment has the following properties: a fixed number of trials (n), the probability of success (p), and the probability of failure (q). Note that the terms success and failure are used loosely and simply indicate the probability of one or the other outcome. Since there are only two possible outcomes, the sum of p and q is always 1. This property simplified the program BinomialDistribution where the constructor requires parameters for the integer 'n' number of trials and the double 'p' probability of the desired 'success' result. It is important to note that the chosen 'success' outcome must match with the parameter input y when calculating the probability of exactly 'y' successes. This leads to the pmf method that calculates the probability of 'y', a given number of successful outcomes throughout the Binomial Distribution. This method models the probability mass function for this distribution that can be found in the formula sheet on GitHub. Notice that the method makes use of the PermutationsAndCombinations program created in the last project since it has a combinations method.

Additional work was done to the Binomial Distribution program to handle the cases where the user needs a probability of the successful outcome happening at least or at most y amount of times. This was done by creating a duplicate method that accepted one more parameter as a bound. Depending on whether 'y' or 'bound' is higher, the method will recursively add the probability of each integer from y then up or down to the bound.

A Geometric Distribution is very similar to the Binomial Distribution in terms of the setup of the experiment with identical trials and only two possible outcomes. However, this Distribution determines the probability of finding the first success after a given number of trials. This assumes the experiment will end after the first success, and all trials before the first success were failures. Because of this, the parameter 'n' for the number of trials seen before is not necessary. The only parameter needed for a geometric distribution is the probability of finding success in a trial. The pmf method will then calculate the probability of the first success occurring on the 'y^{th}' try. The variance and expected value of this distribution are not dependent on y, so they do not need parameters as the pmf method did.

A random variable is a Hypergeometric Probability Distribution when a random sample size 'n' is taken from a larger population 'N' that consists of a total 'r' desirable elements where there are exactly two possible outcomes. Binomial and Geometric distributions work well when N and r are very large compared to n, but we can apply the Probability Mass Function of a Hypergeometric Distribution to experiments where 'n' is relatively large compared to 'N'. The HypergeometricSolver object accepts three primitive type integer parameters that are converted into BigInteger objects to populate the global fields during construction. The global variables represent the three variables mentioned above which are N, r, and n. This program has a method pmf that returns the probability of an integer parameter 'y' or the number of desired elements which is less than or equal to 'r' will be selected in the sample set 'n' in the larger population 'N'. This description holds a lot of information, but the global variables are constant to each problem which means the probability depends solely on the parameter y that ranges from 0 to 'r'. In other words, if there are 5 desirable elements in the population of 20 people, and we want to fill a group of 10 individuals with exactly 3 desirable elements, we would need to calculate $pmf(3)$. In this example, the population size 'N' is 20, the sample size 'n' is 10, and the amount of desirable elements 'r' is 5 which would all be assigned during construction of the distribution. Methods for the expected value and variance are also included based on the equations in the Formula Sheet.

Some extra work was done to both improve the versatility of the PermuatationsAndCombinations class from the last project in order to be reused in this program. An alternative combinations method was created to handle and return BigInteger objects. This provides an upgrade from the original long primitive types so the program will not fail due to overflow. Writing this program to handle BigInteger objects along with displaying them properly was challenging especially when trying to divide. All bugs were ironed out before submission, so the program successfully handles BigInteger values.

When incidents occur independently, in continuous time, and at a constant rate, we can model with a Poisson Distribution. This type of experiment is simple in the sense that it only requires one variable which is the average amount of times that a given event happens in a given time frame. This rate is labeled with the Greek letter Lambda. The variance and expected value of this distribution are always equal to lambda, while the probability mass function only depends on lambda and the parameter y. This program includes pre and post conditions in the javadoc comments specifying that lambda and y are always positive inputs that will return a value between 0 and 1. The pmf method returns the probability of a given event such as a car crash happening 'y' amount of times in one day given there are 35 per year. This means there is an average of 35/365 car crashes per day making lambda equal to approximately 0.095 for this problem. We could then use the pmf function to calculate the probability of (y = 1, 2, 3, 4, 5…) car accidents happening in one day.

The plotter program consists of three classes: Plotter, Salter, and Smoother where Plotter is the parent class to Salter and Smoother. These three classes work together to demonstrate salted datasets and smooth datasets. The function modeled in plotter is hard coded for the sake of simplicity, and it represents the graph of

$$y = e^{-(x-2)^2}.$$

This graph resembles a traditional bell curve with domain 0 to 4. The Plotter object accepts four parameters during construction that include a String for the file name and double values for the domain bounds and step values for the domain. These inputs determine the amount of Ordered Pairs in the OrderedPair array called plot. In the constructor, plot is filled with x values ranging from xMin to xMax and incrementing by delX with respective y values determined by the method 'f'. The internal protected class OrderedPair is accessible by Plotter and its subclasses. It simply serves as a two dimensional data container to assist the Plotter in generating a CSV file. The method sendToCSV writes every OrderedPair in the array 'plot' on its own line with a comma separating the values x and y. After constructing a Plotter with domain bounds and step size, the sendToCSV method will generate the CSV file that can be opened with excel and shown as a graph. Note that the smaller size of delX, the more points will appear in the CSV file and graph.

After the initial construction of the Plotter, the Salter is intended to use the already existing dataset generated by the Plotter. This is done by utilizing the second constructor in Plotter that accepts an OrderedPair array along with a string for the file name. Salter has one unique method not inherited from Plotter that is named salt. This method acts as a mutator to its own global variable 'plot' as it visits every OrderedPair and increases or decreases the y value by a random double between 0 and 0.2. The process for determining whether to add or subtract the random value is decided by a random selection from 0 or 1. This method aims to slightly adjust each y value in the plot with random magnitude and direction. The resultant salted plot created during testing can be found in the folder named PlotterOutputExcelFiles.

Similarly, the Smoother program extends Plotter, and it is intended to be constructed using the plot obtained from salting the data with the Salter object. This class has a method called smooth that calls on the internal method 'average' to adjust the y value in each OrderedPair closer to the average between itself and the two surrounding OrderedPair y values on the left and right. Notice that this method needs to handle the edge cases at the first and last index of the OrderedPair array since there is either no left or no right OrderedPair. This problem is solved with conditional cases that will find the average using the two OrderedPair objects to the right at the first index, and the two OrderedPair objects to the left at the last index. Every OrderedPair in the middle of the array will be treated normally as described above.

Finally, the PokerHand program contains the most classes that include Card, Deck, HandEvaluator, ProbabilitySimulation, and TestHandProbabilities. Each class is well documented with javadoc comments for every method and constructor that explain in detail the functionality of each method and the reasoning behind some of the logic. In addition to the poker hands assigned in class, the few remaining special hands that include a straight flush and royal flush were added to the simulation. The tester class ran this simulation 1 million times for accuracy, and I found it interesting that the Royal flush only occurred one time out of 1 million hands!

Extra work for this program included the extra poker hands, and extensive javadoc comments explaining the logic behind the boolean methods to determine the presence of the exact poker hand being tested. Some work has been done to create a ranking system for the poker hands using the HandEvaluator class, however it did not make the deadline to be submitted with the project.