Semester Project - Pooled Testing

Veronica Longley

December 15, 2021

Our Goal: In this assignment we will simulate pooled COVID-19 testing similar to what was conducted here at Marist last year. We will assume an infection rate of 2 percent, 100 percent testing accuracy, group sizes of 8, and a population of 1000 people to start. We will infect the population, shuffle the positive cases, and test the population. As we test we will count the number of testing cases: case 1 indicates the group of 8 had no positive tests, case 2 indicates 1 positive test, and case 3 indicates 2 or more positive tests. Case one requires one test for each group of 8. Case two requires one test for the group of 8, two tests for each group of four and four individual tests, 7 in all. Case three requires one test for the group of 8, two tests for each group of four and eight individual tests, 11 in all. Note: If two, or more, positive tests are in the same group of four (both in first half or both in second half) we will count this as a case 2 as it will require 7 tests though technically it is a case 3 (2 or more positive tests). We will start with a population of 1000 people then see how our results vary with larger population sizes.

1 Simulation

```
import java.util.Random;
import java.util.Arrays;
import java.util.stream.*;

public class main
{
        public static float case1;
        public static float case2;
        public static float case3;
        public static float numofTests;
        public static void main(String[] args)
        {
                int popSize = 1000;
                double infectedAmt = popSize * 0.02;
                int [] population = new int [popSize];
                int groupSize = 8;

                for(int j = 0; j < popSize; j++)
                {
                        population[j] = 0;
                        //System.out.println(population[j]);
                }//for

                for(int i = 0; i <infectedAmt; i++)
                {
                        population[i] = 1;
                }//for

                shuffle(population, popSize);

                int [] samplePool = new int [groupSize];

                for(int k = 0; k < popSize; k = k+ 8 )
                {
                        samplePool[0] = population[k];
                        samplePool[1] = population[k+1];
                        samplePool[2] = population[k+2];
                        samplePool[3] = population[k+3];
```

```
                        samplePool[4] = population[k+4];
                        samplePool[5] = population[k+5];
                        samplePool[6] = population[k+6];
                        samplePool[7] = population[k+7];
                        runtest(samplePool);
                }//for

                System.out.println("Case (1): " +  case1 + " instances requiring "
                + (case1 * 1) + " tests. ");
                System.out.println("Case (2): " + case2 + " instances requiring "
                + (case2 * 7) + " tests. ");
                System.out.println("Case (3): " + case3 + " instances requiring "
                + (case3 * 11) + " tests. ");
                System.out.println("-----------------------------------");
                System.out.println(numofTests + " tests to screen a population of "
                + popSize + " people with an infection rate of 2%. ");

        }//main

        static void shuffle (int values [], final int TOTNUM)
        {
                Random random = new Random();
                int temp = -1;
                random.nextInt();
                for (int i = 0; i < TOTNUM; i++)
                {
                        int change = i + random.nextInt(TOTNUM - i);
                        temp = values[i];
                values[i] = values[change];
                values[change] = temp;
                }//for
        }//shuffle

        static void runtest(int pool [])
        {
                boolean posneg = false;
                boolean posneg2 = false;
                boolean posneg3 = false;

                posneg = test(pool);

                if(posneg == false)
                {
                        case1++;
                }//if
                else
                {
                        int[] firstHalf = new int [4];
                        firstHalf[0] = pool[0];
                        firstHalf[1] = pool[1];
                        firstHalf[2] = pool[2];
                        firstHalf[3] = pool[3];
                        int[] secondHalf = new int [4];
                        secondHalf[0] = pool[4];
                        secondHalf[1] = pool[5];
                        secondHalf[2] = pool[6];
                        secondHalf[3] = pool[7];
                        posneg2 = test(firstHalf);
```

```
97                          posneg3 = test(secondHalf);
98
99                          if(posneg2 == true && posneg3 == false)
100                         {
101                                 singleTest(pool[0]);
102                                 singleTest(pool[1]);
103                                 singleTest(pool[2]);
104                                 singleTest(pool[3]);
105                                 case2++;
106                         }//if
107                         if(posneg3 == true && posneg2 == false)
108                         {
109                                 singleTest(pool[4]);
110                                 singleTest(pool[5]);
111                                 singleTest(pool[6]);
112                                 singleTest(pool[7]);
113                                 case2++;
114                         }//if
115                         if(posneg2 ==true && posneg3 ==true)
116                         {
117                                 case3++;
118                                 singleTest(pool[0]);
119                                 singleTest(pool[1]);
120                                 singleTest(pool[2]);
121                                 singleTest(pool[3]);
122                                 singleTest(pool[4]);
123                                 singleTest(pool[5]);
124                                 singleTest(pool[6]);
125                                 singleTest(pool[7]);
126                         }//if
127                 }//else
128
129         }//runtest
130
131         static boolean test(int pool[])
132         {
133                 numofTests ++;
134                 boolean result = false;
135                 if(IntStream.of(pool).sum() != 0)
136                 {
137                         result = true;
138                 }//if
139                 return result;
140         }//test
141
142         static boolean singleTest(int value)
143         {
144                 numofTests++;
145                 boolean result = false;
146                 if(value == 1)
147                 {
148                         result = true;
149                 }//if
150                 return result;
151         }//singleTest
152 }//main
```

2 Breaking Down the Code

To start, we introduce global variables to count the number of case 1 instances, the number of case 2 instances, the number of case 3 instances, and the total number of tests needed to test the population. We initialize the population size to 1000; however, this will be altered later to see how our simulation adjusts. First, we create an array called population to hold an index for each person. We initialize each index array to be zero. Then we loop through changing the zero to one for the number of people we need to infect (2 percent times the population size). Then we shuffle the array. Next, we have a for loop to break the population into groups of 8, store them in another array, and call runtest() to test them. Skipping down to runtest() (line 72), we start by introducing three booleans all initialized to false. Runtest() then calls a helper method test which takes in the array, increments the number of tests, checks if the sum of the array is zero and if it is not changes return boolean to true, and finally returns that boolean. If false is returned on the first test we know there are no infections in this group, so we increment the number of case 1 instances and move on to the next group. If true is returned we know we have at least one case. We introduce two sub arrays of size four, fill them with the first and second half of the group, and test each half, storing the boolean test() return. If posneg2 is true and posneg3 is false, we must single test each of the first four people in the group as we know the case is in the first half of the group. Similarly, if posneg2 is false and posneg3 is true, we must single test each of the last four people in the group as we know the case is in the second half of the group. For both of these cases we increment the case 2 counter. Lastly, if both posneg2 and posneg3 are true we know we have at least one case in both halves of the group, so we must single test everyone in the group. We do this and increment the case 3 counter before moving to the next group. Note: singleTest() works the same as test() except instead of summing the array we are just checking if the one index is 0 or 1. Once these tests have run on the entire population we can print out the number of case 1 instances, the number of case 2 instances, the number of case 3 instances and the total number of tests needed for the population.

3 Results by Population Size

The below table shows how our case counters and number of tests needed for the population changes as the population grows. The infection rate will remain constant at 2 percent.

Population Size	Case 1	Case 2	Case 3	Total Number of Tests
1,000	106, 105..	18, 20..	1, 0..	243, 245..
10,000	1067, 1056 ..	177, 192 ..	6, 2..	2372, 2422 ..
100,000	10638, 10648 ..	1794, 1775 ..	68, 77..	23944, 23920 ..
1,000,000	106309, 106381 ..	17963, 17840 ..	728, 779..	240058, 239830 ..

For population size 1,000 we know we have 125 groups of eight with case one showing 85 percent of the time, case two about 14.96 percent of the time and case three about 0.04 percent of the time. This results in about 106.25 instances, 18.7 instances and 0.05 round up to 1 instance for each case respectively. This is extremely consistent with our results. For population size 10,000 we should be getting about 1062.5 ((10,000/8) * 0.85) case 1 instances, 187 ((10,000/8) * 0.1496) case 2 instances, and 0.5 ((10,000/8)*0.0004)

case 3 instances. This again is fairly consistent with our results. Looking at a population of 100,000, we expect 10,625 ((100,000/8) * 0.85) case 1 instances, 1,870 ((100,000/8) * 0.1496) case 2 instances, and 5 ((100,000/8)*0.0004) case 3 instances. Here we see the number of case one instances is fairly consistent, but the case 2 and 3 instances are rather far off from our expected values. Lastly, for a population of one million, we expect 106,250 ((1,000,000/8) * 0.85) case 1 instances, 18,700 ((1,000,000/8) * 0.1496) case 2 instances, and 50 ((1,000,000/8)*0.0004) case 3 instances. Again we are seeing many more case 3 instances in a population of 1,000,000 from our simulation than we expect based on these numbers.

Looking at binomial distributions and hypergeometric distributions, we see for binomial distributions the probability for an event to occur is constant for each trial while hypergeometric distributions adjust probability after each trial as there is no replacement. Because we have a set number of positive cases and a set number of groups, we are looking at a hypergeometric distribution because as the positive cases are found, the probability for them being in the next group changes. If we were randomly selecting 8 from the population, testing them, and putting them back into the population, then we would be dealing with a binomial distribution as this is with replacement. When we calculated the expected percentages for each case we used a consistent probability 0.98 to the 8th and assumed each individual would have the same likelihood of being infected. This is fine for smaller populations, but it explains why our numbers begin to vary with larger populations. Once a positive case is found the probability for the next individual to be positive is less, and as population sizes increase this difference beings to affect our expected values even more. So, while we are modeling the distribution as binomial, we would likely have much more accurate results for larger populations if we modeled it as a hypergeometric distribution.