Spring Internship 2020 with Prof. Saurabh Sinha & Shounak Bhogale
Mayank Hirani
February 2020 - June 2020

Introduction

The purpose of the project was to create a simulator that created sample DNA sequences containing a repeating pattern, and to create an algorithm that would find the pattern. This project was to learn about the algorithms and methods that researchers use to find patterns in real DNA sequences across various organisms. Initially, I created a simulator that created random sequences, and generated a random pattern that it would insert into each of these sequences. I then created a pattern-finding algorithm that would read through the sequences and determine the pattern.

After making this more basic simulation, I made a newer simulator which included two features not present in the original simulation: P, a variable that determines the chance a pattern appears in a sequence, and E, a variable that determines the chance of a one-character error in a pattern. With this updated simulator, I was able to measure the accuracy of my previous pattern-finding algorithm. I varied P and E separately to measure the results of the accuracy and plot on a graph. I then varied P and E at the same time and plotted the results on a heatmap.

By the end of the internship the skills I learned were: Reading through and writing files using Python, plotting graphs and heatmaps using matplotlib and pandas and organizing and working with large amounts of data in Python. I also learned that the purpose of finding patterns in sequences is to find similarities, compare, and analyze patterns in various organisms. Pattern-finding algorithms are applied to real-world DNA sequences of organisms in order to determine repeating patterns.

Methods

The first simulator was created using a Simulator class with multiple methods. Two variables, number of sequences and pattern length, were stored as data variables of the class, and within the constructor method all the methods of the class are called. It begins by calling a method, get_input(), that receives the input from the user for the number of sequences and pattern length. Then the create_pattern() function generates a random pattern of the length given by the user in get_input(). The create_sequences() function generates the user input number of sequences with random characters A, T, C, and G. This function writes these sequences to a file, and inserts the generated pattern into each sequence at a random location and writes them to a separate file. One file contains sequences without the pattern, and another contains the sequences with the pattern.

There are two pattern-finding algorithms. The first, a more brute-force method than the second, iterates through the file containing the sequences with the patterns, iterates through each sequence, and saves each possible pattern of the pattern length in a set. A possible

pattern is every section of the sequence that has a length equal to the pattern. A count_patterns() method then iterates through the set, and with each possible pattern, iterates again through the sequences and keeps a dictionary of counts for every possible pattern. The pattern with the highest counts in the dictionary is returned as the pattern.

The second pattern-finding algorithm is a more advanced version of the first, which only has to iterate through all the sequences once. It iterates through all the sequences, and iterates through each character of each sequence, saving each possible pattern to a dictionary. The keys of the dictionary are all the possible patterns, and the values of the dictionary are the count for each possible pattern. After the iteration is complete, the pattern with the highest count is returned.

The final part of the project was to recreate the original simulator, the program that created the sequences and ran the pattern-finding algorithm, but to make it more realistic by adding two new variables in. These two variables were P and E. P is the probability that a pattern is present in a sequence, and E is the probability of error (one of the characters has been altered) in a pattern. To add these variables to the existing simulator, the random library was used for probability. In the insert_pattern() method, a random value between 0 and 1 was drawn using random.random(). If the number was less than or equal to the value of P, the pattern would be inserted into the sequence. For the E variable, a method was created called alter_pattern(), which would return an altered version of the pattern in which one of the characters was randomly changed to a different character. This value was called in the insert_pattern() pattern if a random value drawn was less than E.

The next part was to observe what happens when P and E are varied. To accomplish this, I first varied P from 0 to 100, inclusive, and ran the simulation, as well as the pattern finder algorithm, 100 times for each value of P. A method determine_accuracy() would return whether the pattern finder returned the correct pattern or not. Each value of P would have an accuracy result, how many of the 100 runs were successful. This accuracy result was the dependent value, so when plotted on a graph using matplotlib, the y-axis displayed the accuracy, and the x-axis displayed the value of P. The same procedure was done for E.

After plotting each variable separately, both were varied together. This was done using a heatmap in matplotlib. I used nested loops to first vary P, then inside that loop vary E, and for each combination of values an accuracy value was determined. A numpy array was created to hold the results, with the rows being values of P and the columns being values of E. Using pandas, the numpy array was converted to a dataframe, and then matplotlib.pcolor() was used to plot this dataframe as a heatmap. After doing, multiple tests were done changing different factors to make a variety of heatmaps. First, P and E were only varied by only values of 0.2, but for each combination, the program was run a 1000 times (See figure 3). Next, P and E were varied by 0.01, and each combination was run only once (See figure 4). Afterwards, P and E were varied by 0.05, and ran 100 times (See figure 5), and finally, varied by 0.01 and ran 100 times (See figure 6).

Results

For the first two graphs, Figure 1 and Figure 2, as P increased, the accuracy increased at an increasing rate, and slowed down as P neared 1.00. The opposite was true for E, as E increased, the accuracy decreased at an increasing rate, and slowed down as E neared 1.00. In the heatmaps, the highest accuracies were observed at higher values of P and lower values of E. As the number of runs increased, the graphs appeared to be smoother.

Figure 1: Effect of Varying P from 0.00 to 1.00 on the Accuracy of the Pattern-Finding Algorithm

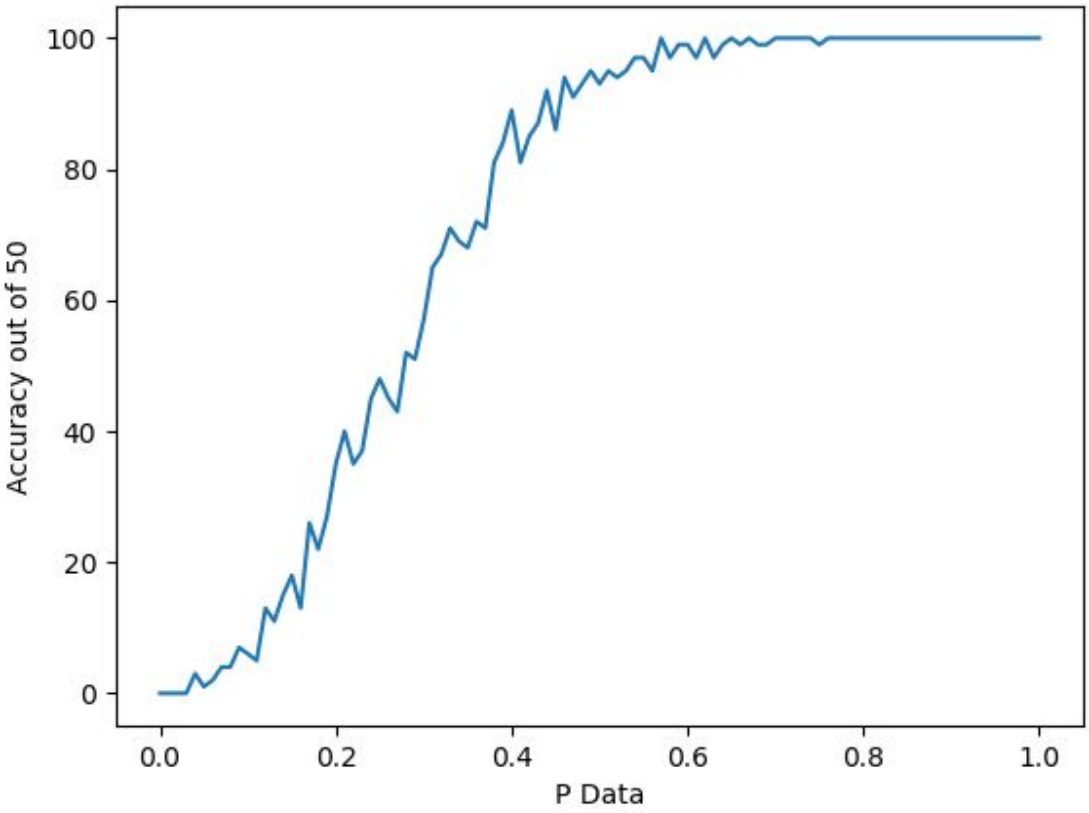


Figure 2: Effect of Varying E from 0.00 to 1.00 on the Accuracy of the Pattern-Finding Algorithm

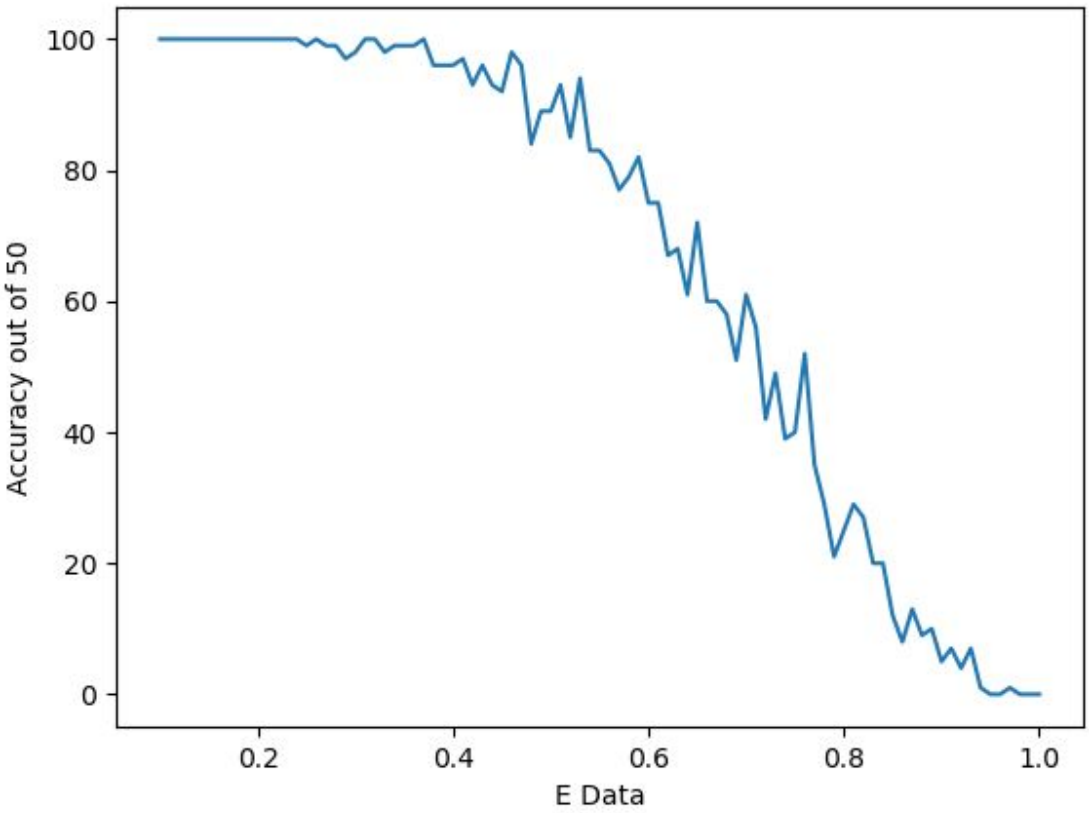


Figure 3: Heatmap result of Varying P and E by 0.2 and running each combination 1000 Times

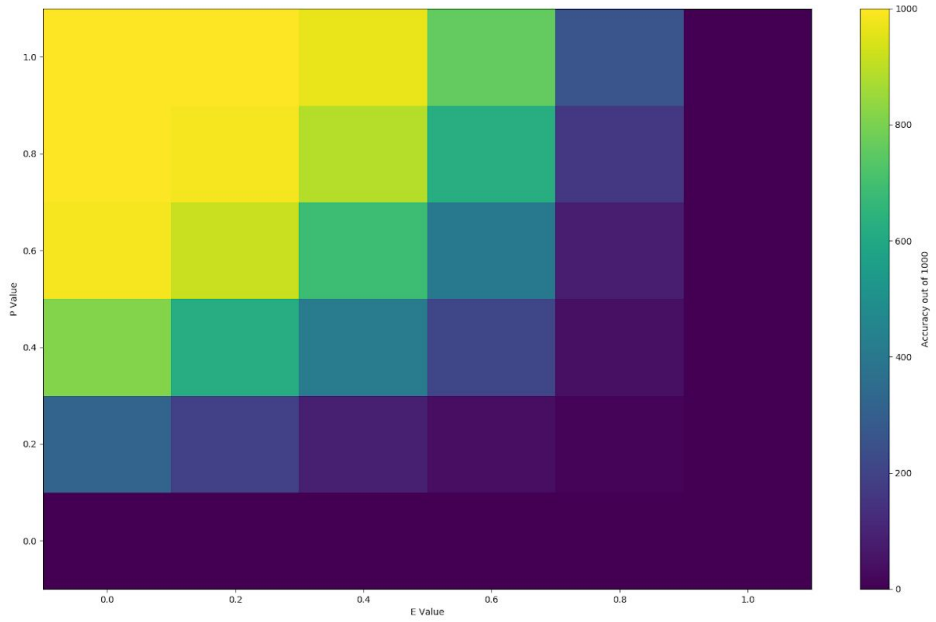


Figure 4: Heatmap result of Varying P and E by 0.01 and running each combination 1000 Times

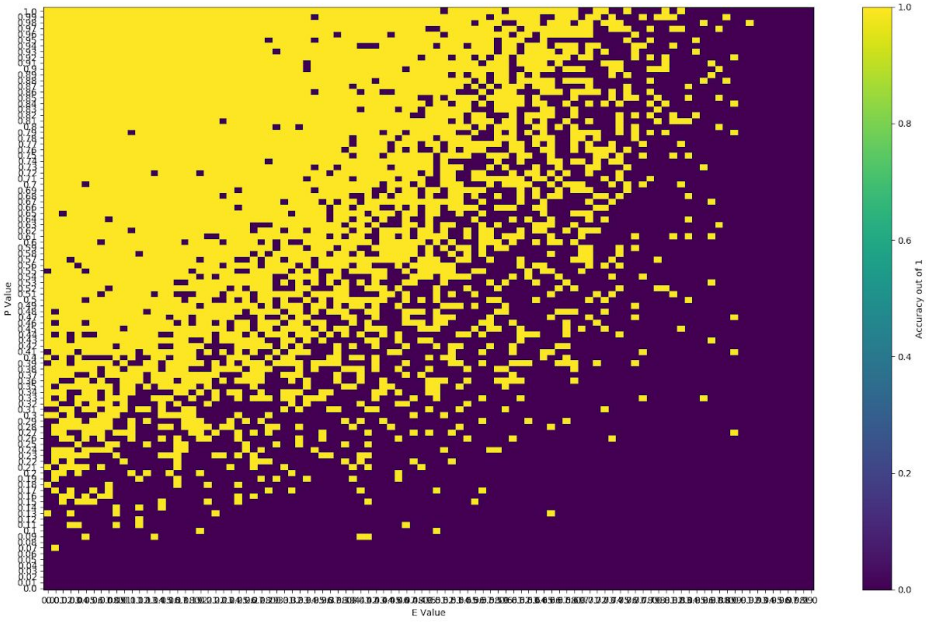


Figure 5: Heatmap result of Varying P and E by 0.05 and running each combination 100 Times

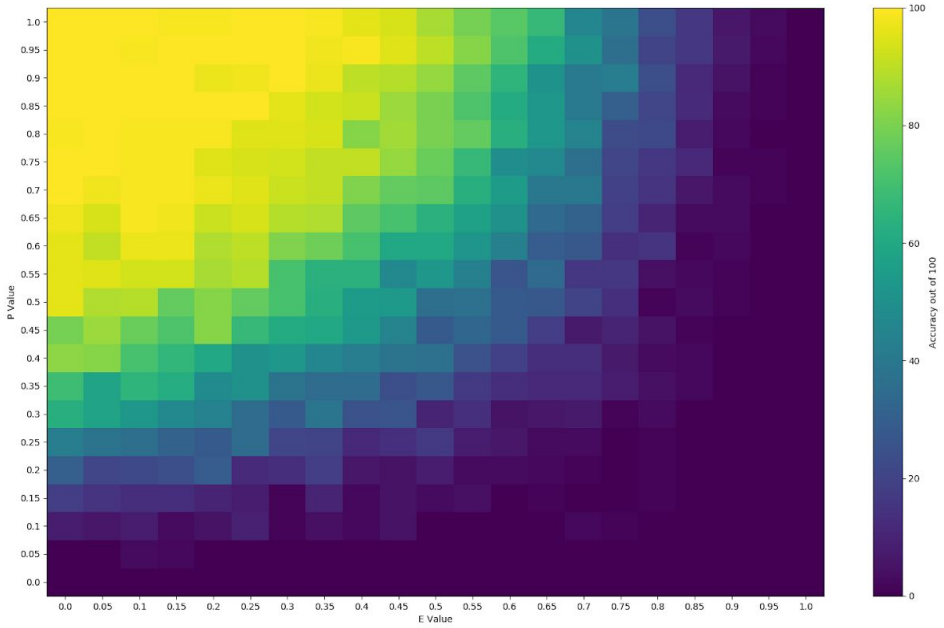
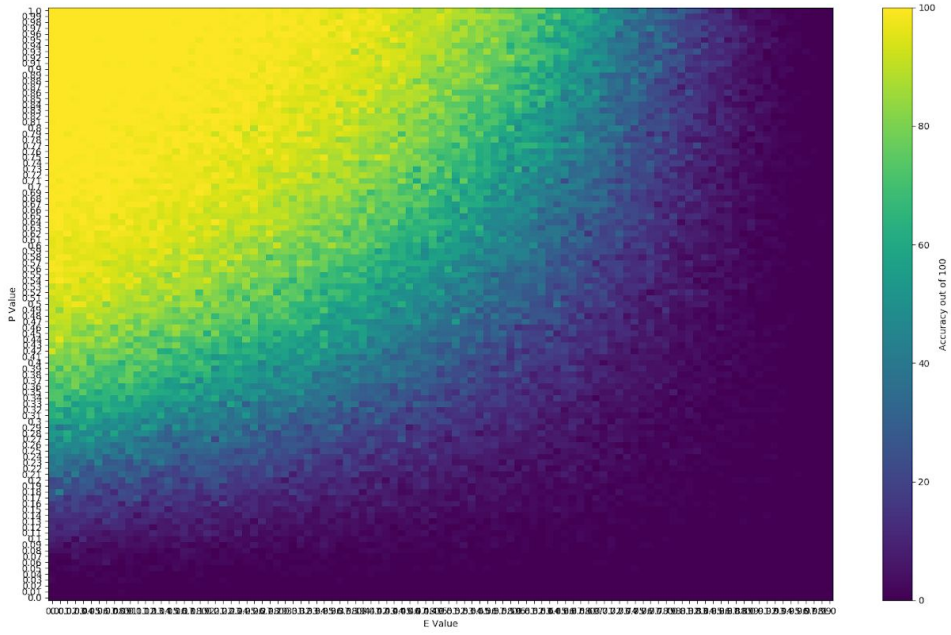


Figure 6: Heatmap result of Varying P and E by 0.01 and running each combination 100 Times



Discussion

The results of the original pattern finder showed that a brute force method of finding patterns always works when the length of the pattern is known and the pattern is present in every sequence. However, this method has very long run times, and must store every single possible pattern from the sequences in a dictionary. The graphs obtained from varying P and E individually, Figures 1 and 2, show that the accuracy changes abruptly as P and E starts to change, due to the fact that as soon as the pattern begins to appear in more and more sequences, the pattern would have a higher count than other patterns. The reason for the plateau as P approaches 1.00, and as E approaches 0.00, could be attributed to the reason that the chance of another pattern occurring more times than the desired pattern is extremely unlikely as P approaches 1.00 and when P is 1.00.

The heatmap results came out as expected, the highest levels of accuracy were associated with higher values of P (the pattern is present in more sequences), and lower values of E (the patterns have fewer errors). This logically makes sense because using the pattern-finding algorithm, if a pattern is present in fewer sequences, the pattern-finder would have a lower count for that pattern, which could cause another pattern to accidentally appear more times. In addition, as E increases, more patterns would have errors in them, and thus will not count towards the pattern's count. The reason that changing P has the same effect as changing E is that with the current pattern-finding algorithm, a value of P is the same as 1 - the same value of E. This is because if there is an error in the pattern, the pattern finder would see that pattern as a completely different pattern than the desired pattern, as if the pattern is not present in the sequence. This means that E becomes the opposite of P. A high value of E is the same as a low value of P, and vice versa. Due to this, both variables have the same influence on the accuracy. This observation and realization was made after the heatmaps were created.

The purpose of plotting the heatmaps was to notice and analyze how P and E differ in their influence on the pattern-finding algorithm. The results from the heatmaps show that P and E have the same amount of influence on the accuracy of the pattern-finding algorithm. This can be determined because the heatmap is symmetrical on the diagonal from (P = 1.00, E = 0.00) to (P = 0.00, E = 1.00). What this means is that changing the value of P would result in the same accuracy as changing the value of E in the opposite direction.

There are three "zones" visible on the heatmap that the results lie in. The highest accuracy results, showing mostly yellow color, resulting in values of P > ~0.50 and values of E < ~0.5. This zone has generally high values of P and generally low values of E, and contains accuracy values of ~90-100%. Following this zone there is another zone, in which accuracy values are between ~40-80%, displayed as a greenish/blueish color. It is important to notice that there is a balance in the values of P and E, so that it is not as optimal as the high-accuracy zone, but still accurate some of the time. The final zone is outside the second, with accuracy values of 20% and less, displayed as a purple color. In this zone, P and E are both either very high, or both very low.

In conclusion, through the internship, I was able to learn about pattern-finding in DNA sequences and how the algorithms work. I wrote my own algorithm, and improved its efficiency

over the course of the internship. I then changed settings for the simulation that the pattern-finding algorithm ran on to measure the effect of changing two different variables on the accuracy of my pattern-finding algorithm. Throughout the internship, I learned about the reason for finding patterns in DNA sequences, the real world application of this topic, and how pattern-finding algorithms work. I was also able to apply my Python skills to this project and improve my knowledge, and learn how to work with libraries like pandas and matplotlib.