CSCE 689 : Algorithms for Big Data

Programming Assignment 1 Report

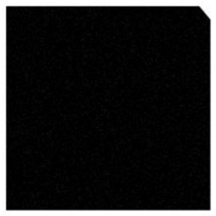
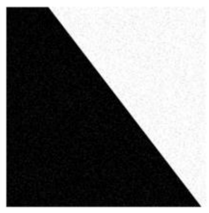
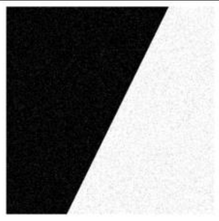
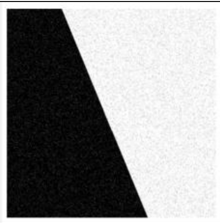
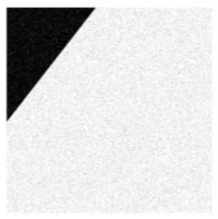
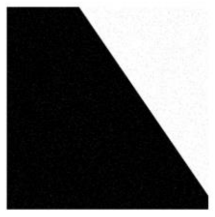
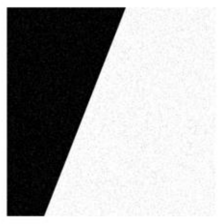
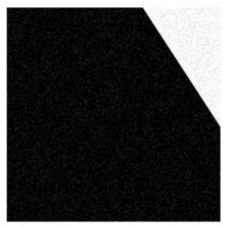
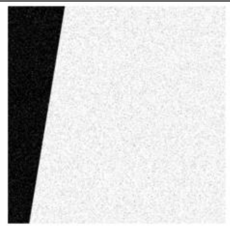
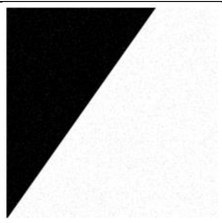
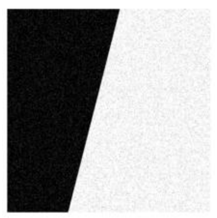
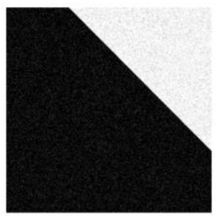
Due: Sept 20 **Name: Vasu Naman Verma**

Half Plane Tester

Generation of Instances:

The `generate_binary_matrix` function is used to create test instances for evaluating the algorithm. We are creating an image with dimensions of 1000x1000 pixels. To begin, we initialize a matrix of size 1000x1000 with all values set to 1, representing white pixels. Next, we construct a line by randomly selecting a slope and intercept.

We then iterate through each pixel in the matrix and determine its position relative to the line. If a pixel falls on the left side of the line, we change its color to black. Additionally, we randomly flip pixels with a probability of q.

Epsilon/q	0.01	0.02	0.03	0.04	0.05
0.001					
0.1					
0.3					

Experiment:

The experiment is conducted for each epsilon value. For each epsilon, there are 5 different q values considered. For each q value, an image is generated using the `generate_binary_matrix` function. Subsequently, the algorithm is executed 50 times for each generated image.

The algorithm begins by calculating how many sides of the image have endpoints with the same or different colors. If all four sides have endpoints with different colors, the algorithm always returns False. If all four sides have endpoints with the same color, the algorithm samples ln(1/1-p)/epsilon pixels independently at random. If all of these pixels have the same color, the algorithm returns True; otherwise, it returns False.

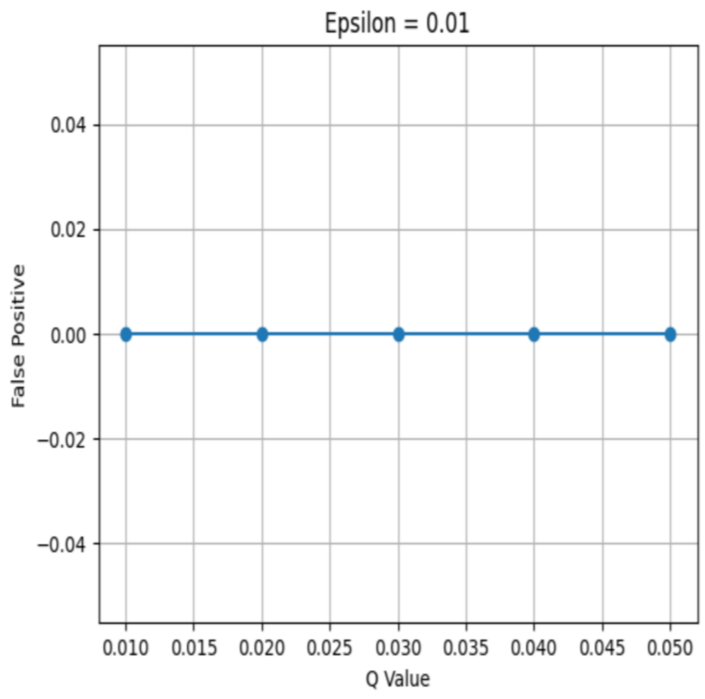
In the case where two sides have endpoints with different colors, the algorithm first identifies which two sides these are. Subsequently, it conducts a binary search on these sides to find a white and black pixel pair with a distance of less than (epsilon*n)/2 between them on both sides. The algorithm repeats this process for both sides.

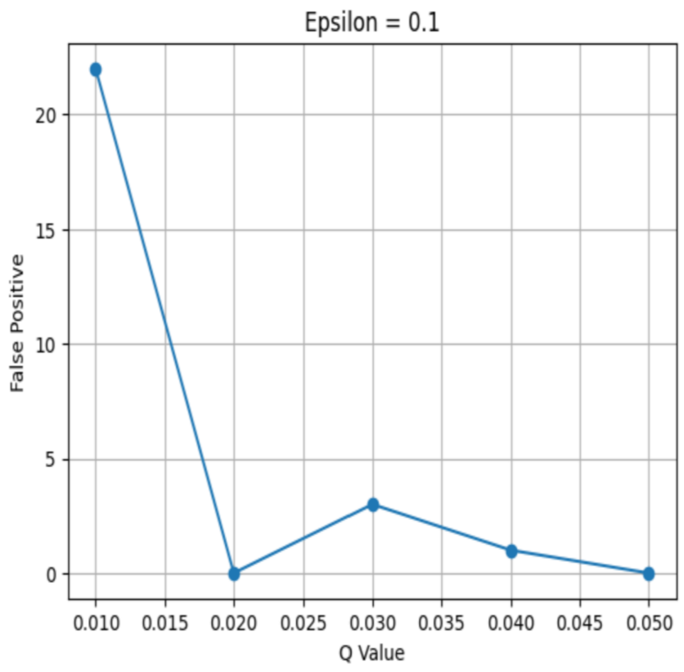
Then, two lines are constructed by connecting both the white pixels and both the black pixels found earlier. Afterward, the algorithm randomly selects pixels 2*np.log(1/(1-p))/epsilon times. For each selection, it checks whether the pixel falls on the side with different colored endpoints. If it does not, the algorithm returns True; otherwise, it returns False.

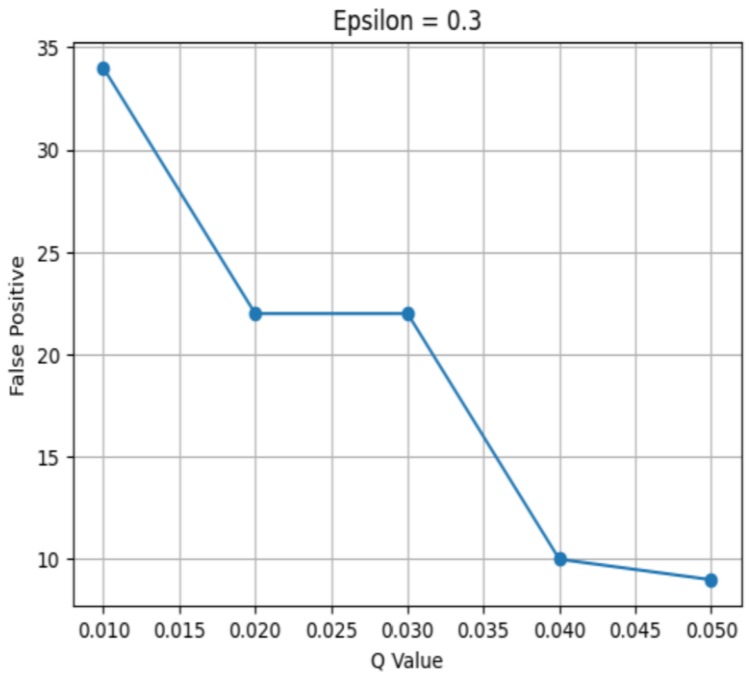
Since we are aware that the images contain flipped pixels, the expected outcome should be False. Therefore, we keep track of the number of times the algorithm returns True, which signifies a false positive. This count is recorded as the false positive count.

Finally, we create a plot showing the relationship between the false positive count and the various q values for each epsilon value.

Results:







<u>Analysis:</u>

As the value of q increases, the number of pixels that flip in the generated binary matrix also increases.

The algorithm's sample size is determined by the formula:

```
Sample size = 2*ln(1/1-p)/epsilon
```

For very small epsilon values, the sample size becomes significantly large. The algorithm is able to detect whether the image satisfies the half-plane property with high accuracy because it processes a large number of pixels. So, in case where epsilon = 0.01, we are able to achieve 100% accuracy.

When epsilon is relatively large and q is small, the algorithm may yield more false positives. This is because the sample size is smaller, making it challenging to accurately estimate the image properties.

As the value of q increases, there are more flipped pixels, and the algorithm is able to detect those pixels even with a small sample size. So as q increases, the accuracy of the algorithm increases.

The runtime of the algorithm is sublinear in nature because it processes a relatively small sample size of pixels. The exact runtime performance depends on the values of the probability (p) and epsilon parameters.