

Slovenská Technická Univerzita

Fakulta informatiky a informačných technológií

Klasifikácia

Zadanie 3.

Umelá Inteligencia

Martin Čajka

AIS ID: 116158

Školský rok 2022/2023

Description of the assignment:

The task of the assignment was to create a classifier in a 2D space ranging from (X,Y) -5000 to +5000. Its job is to classify points generated in the given range into four categories: Purple, Blue, Red and Green. At the beginning the space contains 5 points of each colour at the given positions:

R: [-4500, -4400], [-4100, -3000], [-1800, -2400], [-2500, -3400] a [-2000, -1400]

G: [+4500, -4400], [+4100, -3000], [+1800, -2400], [+2500, -3400] a [+2000, -1400]

B: [-4500, +4400], [-4100, +3000], [-1800, +2400], [-2500, +3400] a [-2000, +1400]

P: [+4500, +4400], [+4100, +3000], [+1800, +2400], [+2500, +3400] a [+2000, +1400]

The classifier will implement a function `classify(int X, int Y, int k)`, which classifies the new point with coordinates (X, Y) and returns the assigned colour. A k-NN algorithm has to be used for the purpose of classification, with k-values of 1, 3, 7 or 15.

For the demonstration of the program a testing environment has to be created. Generate 40 000 points (10 000 of each). The coordinates will be random with each following points colour to be different than the previous ones. All points have a 99% chance to spawn in the proximity of their colour quadrant (max. 500 in another quadrants range) Repeat the test 4 times for each k-value. The end results of each classification should be visualised.

Representation of points:

```
212 class POINT:
213
214     def __init__(self, x_position, y_position, givenColor):
215         self.x = x_position
216         self.y = y_position
217         self.color = None
218         self.givenColor = givenColor
219         self.distance = 0
```

Each point contains the following attributes:

X – The x coordinate value.

Y – The y coordinate value.

Colour – The assigned colour from the classifier.

givenColour – The assigned colour given at point generation.

distance – The calculated distance from the currently classified point.

k-NN Algorithm:

The k nearest neighbour's algorithms passed k value represents the number of nearest neighbours the classification result is going to be based on. For example, for k-value 7, the algorithm searches for the closest 7 points to the coordinates of the currently classified point. The classified points colour is going to be the most occurring colour between his neighbours.

Classify function:

```
161 def classify(x, y, k):
162
163     for point in POINTS_LIST:
164         getDistance(point, x, y)
165
166     sortedList = sorted(POINTS_LIST, key=lambda d: d.distance, reverse=False)
167
168     KNN_LIST = []
169
170     purpleCounter = 0
171     blueCounter = 0
172     redCounter = 0
173     greenCounter = 0
174
175     i = 0
176
177     while i != k:
178         KNN_LIST.append(sortedList[i])
179         i += 1
180
```

The classification function first calculates the distance from the classified point (x, y coordinates) for each already existing point. Afterwards it sorts the list of existing points by distance, and iteratively adds the closest k points to the KNN_LIST. Existing points are points from the starting dataset and already generated and classified point.

```
181     for point in KNN_LIST:
182
183         match point.color:
184             case "Purple":
185                 purpleCounter += 1
186             case "Blue":
187                 blueCounter += 1
188             case "Red":
189                 redCounter += 1
190             case "Green":
191                 greenCounter += 1
192
193     highestOccur = max(purpleCounter, blueCounter, redCounter, greenCounter)
194
195     if highestOccur == purpleCounter:
196         return "Purple"
197     elif highestOccur == blueCounter:
198         return "Blue"
199     elif highestOccur == redCounter:
200         return "Red"
201     else:
202         return "Green"
203
```

Next, the program iterates the KNN_LIST and counts number of occurrences of each colour. Then it finds the highest occurring colour and returns it as the colour of the classified point.

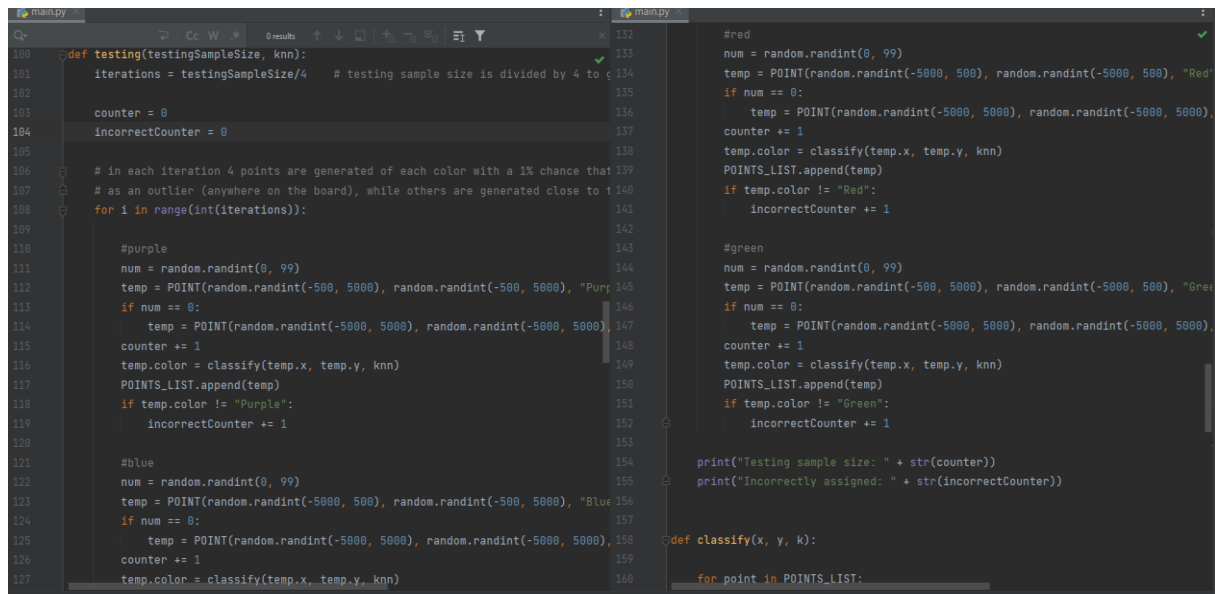
getDistance function:

```
206 def getDistance(point, target_X, target_Y):
207
208     distance = math.sqrt((point.x - target_X)*(point.x - target_X) + (point.y - target_Y)*(point.y - target_Y))
209     point.distance = distance
210
```

The getDistance function calculates the distance of the given point, from the passed coordinates using the Pythagoras theorem $\sqrt{(x - x)^2 + (y - y)^2}$.

Testing:

Testing has taken the form as described in the description part. For each k-value 40 000 points have been generated. The points have been generated iteratively in order of the colour quadrants: Purple – Blue – Red – Green.



```
def testing(testingSampleSize, knn):
    iterations = testingSampleSize/4 # testing sample size is divided by 4 to
    counter = 0
    incorrectCounter = 0

    # in each iteration 4 points are generated of each color with a 1% chance that
    # as an outlier (anywhere on the board), while others are generated close to
    for i in range(int(iterations)):

        #purple
        num = random.randint(0, 99)
        temp = POINT(random.randint(-5000, 5000), random.randint(-5000, 5000), "Purple")
        if num == 0:
            temp = POINT(random.randint(-5000, 5000), random.randint(-5000, 5000))
            counter += 1
        temp.color = classify(temp.x, temp.y, knn)
        POINTS_LIST.append(temp)
        if temp.color != "Purple":
            incorrectCounter += 1

        #blue
        num = random.randint(0, 99)
        temp = POINT(random.randint(-5000, 5000), random.randint(-5000, 5000), "Blue")
        if num == 0:
            temp = POINT(random.randint(-5000, 5000), random.randint(-5000, 5000))
            counter += 1
        temp.color = classify(temp.x, temp.y, knn)

        #red
        num = random.randint(0, 99)
        temp = POINT(random.randint(-5000, 5000), random.randint(-5000, 5000), "Red")
        if num == 0:
            temp = POINT(random.randint(-5000, 5000), random.randint(-5000, 5000))
            counter += 1
        temp.color = classify(temp.x, temp.y, knn)
        POINTS_LIST.append(temp)
        if temp.color != "Red":
            incorrectCounter += 1

        #green
        num = random.randint(0, 99)
        temp = POINT(random.randint(-5000, 5000), random.randint(-5000, 5000), "Green")
        if num == 0:
            temp = POINT(random.randint(-5000, 5000), random.randint(-5000, 5000))
            counter += 1
        temp.color = classify(temp.x, temp.y, knn)
        POINTS_LIST.append(temp)
        if temp.color != "Green":
            incorrectCounter += 1

    print("Testing sample size: " + str(counter))
    print("Incorrectly assigned: " + str(incorrectCounter))

def classify(x, y, k):
    for point in POINTS_LIST:
```

The testing function receives the sample size and the k-value as parameters. Afterwards it calculates the number of iterations (sample Size/4) and generates and classifies 4 points (1 of each colour) each iteration.

Parameter k=1:

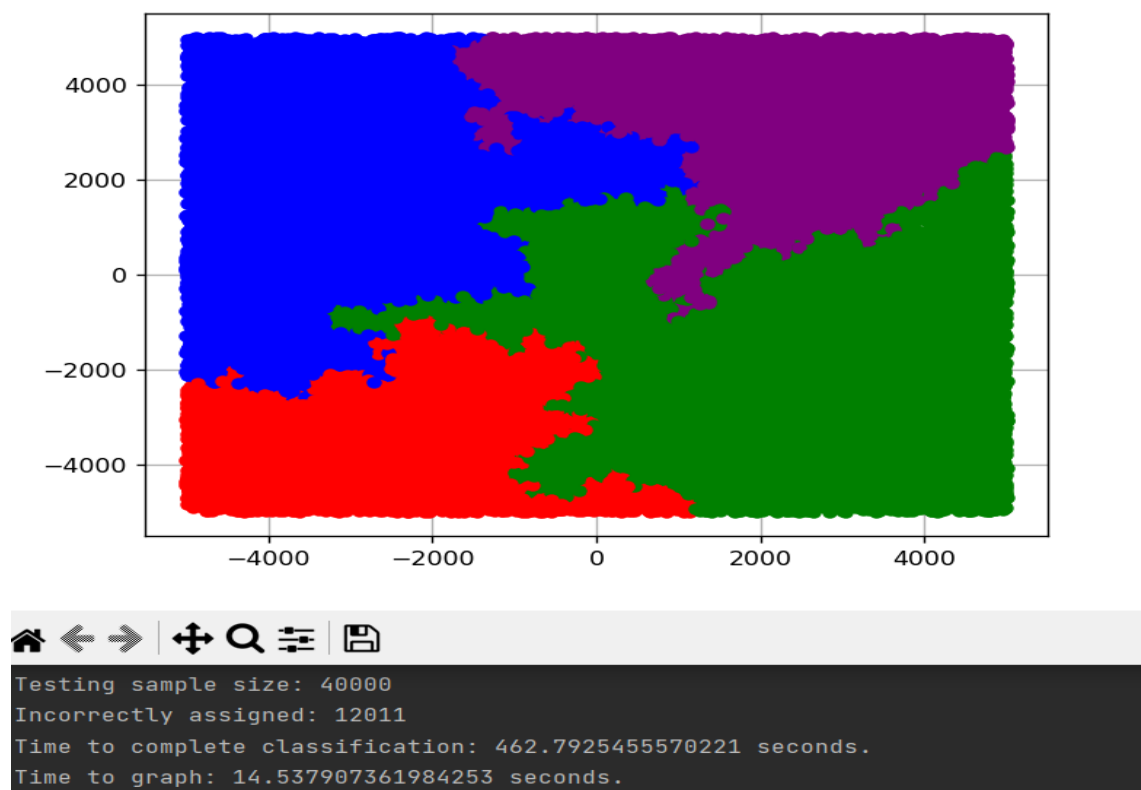


Figure 1 - The classification result for k=1.

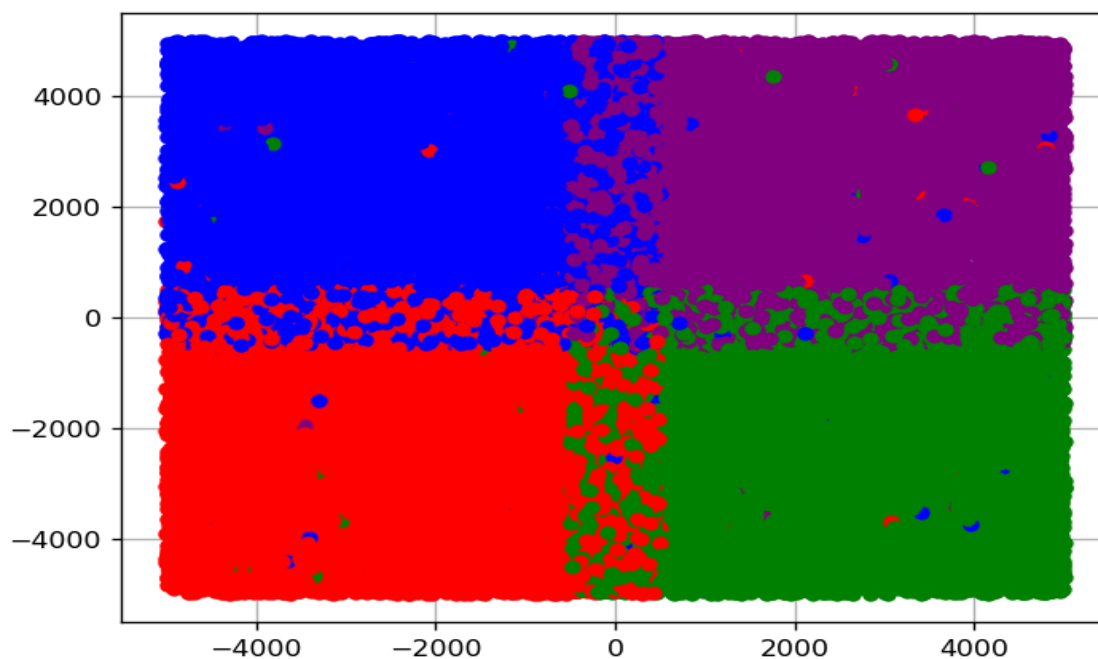
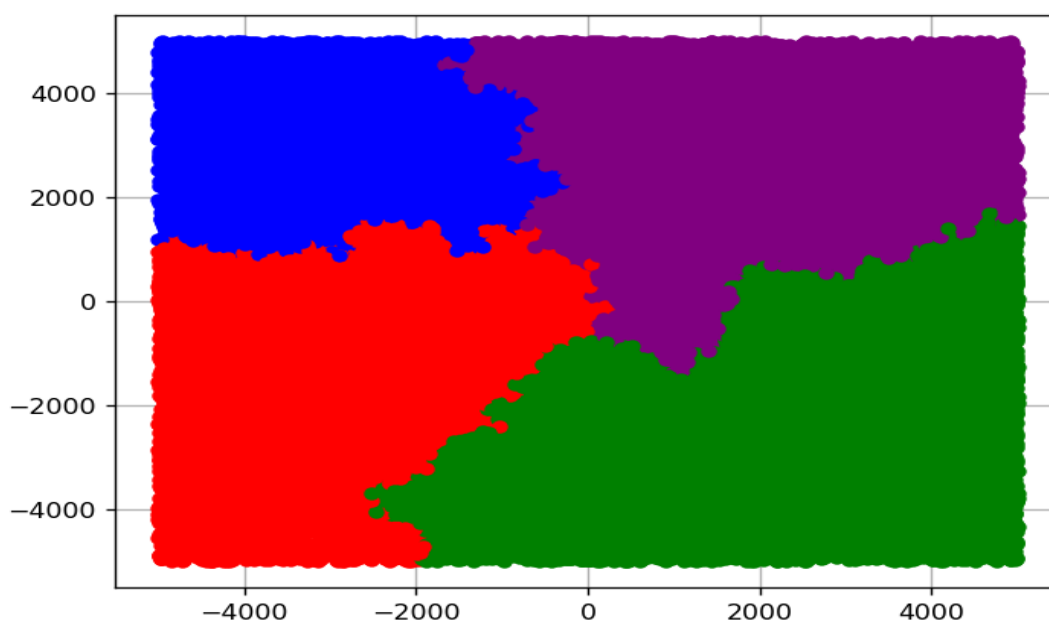


Figure 2 - The generated points for k = 1

Parameter k=3:



```
Enter number of nearest neighbours: 3
Testing sample size: 40000
Incorrectly assigned: 10906
Time to complete classification: 467.2785096168518 seconds.
Time to graph: 14.525989770889282 seconds.
```

Figure 3 - The classification result for k=3.

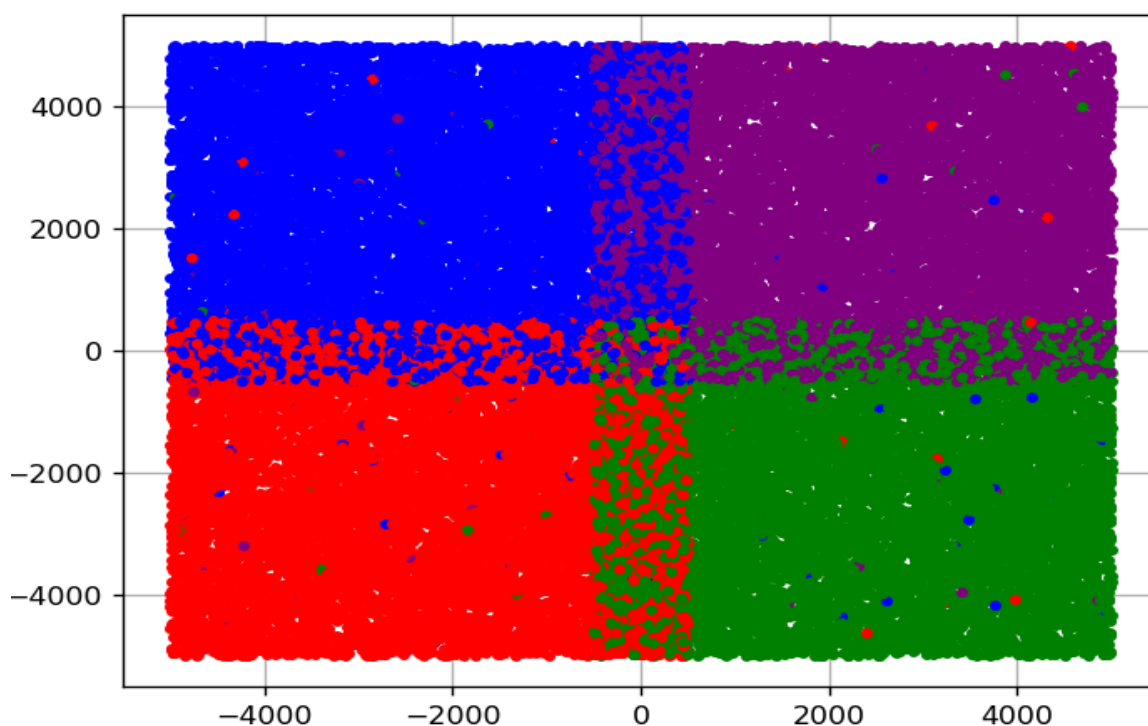


Figure 4 - The generated points for k = 3

Parameter k=7:

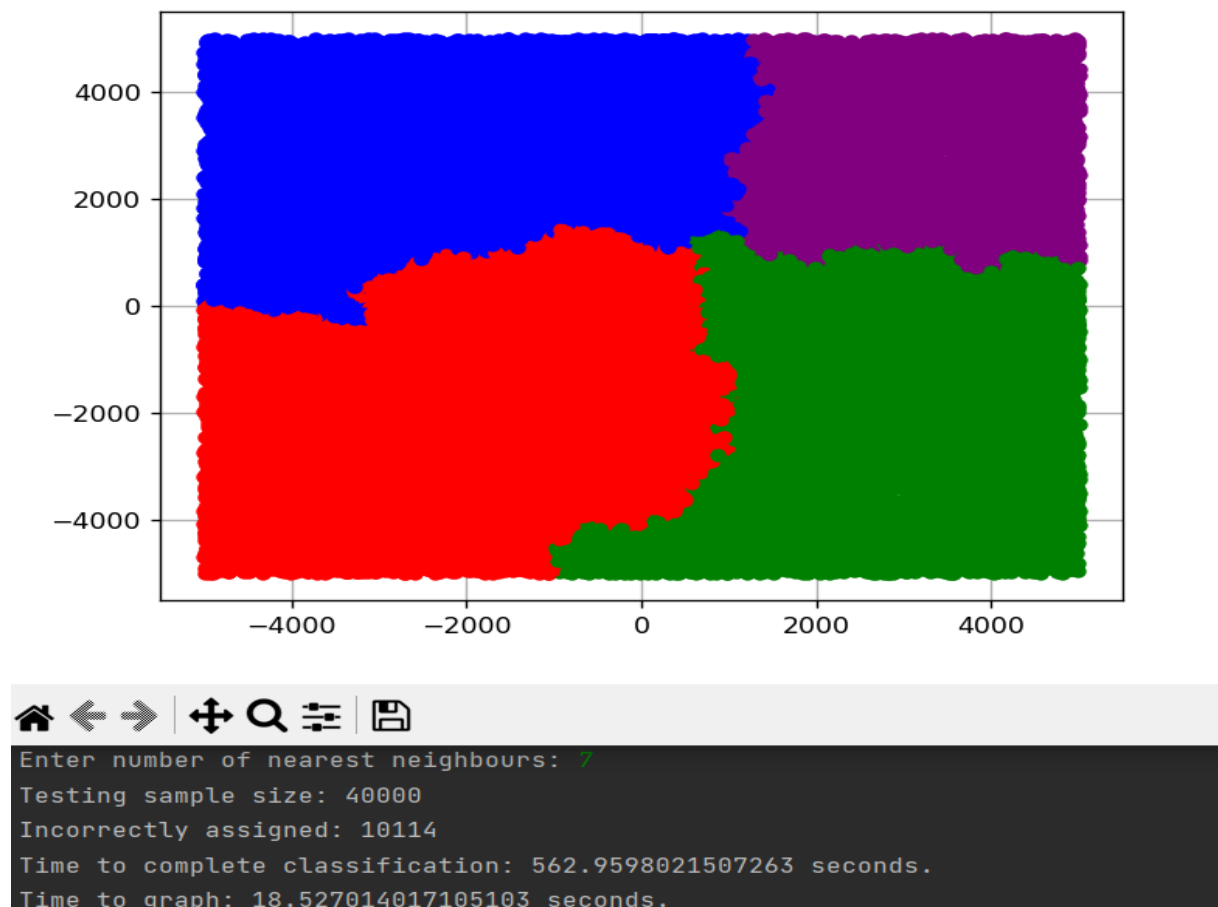
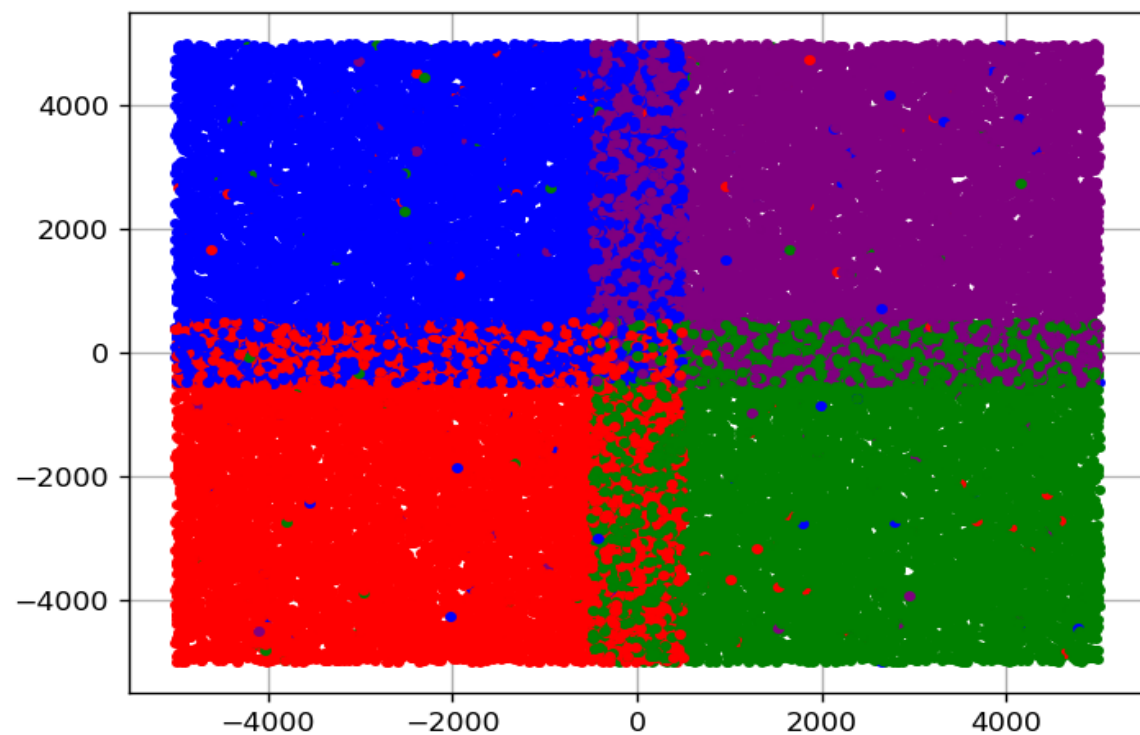


Figure 5 - The classification result for k=7.



Parameter k=15:

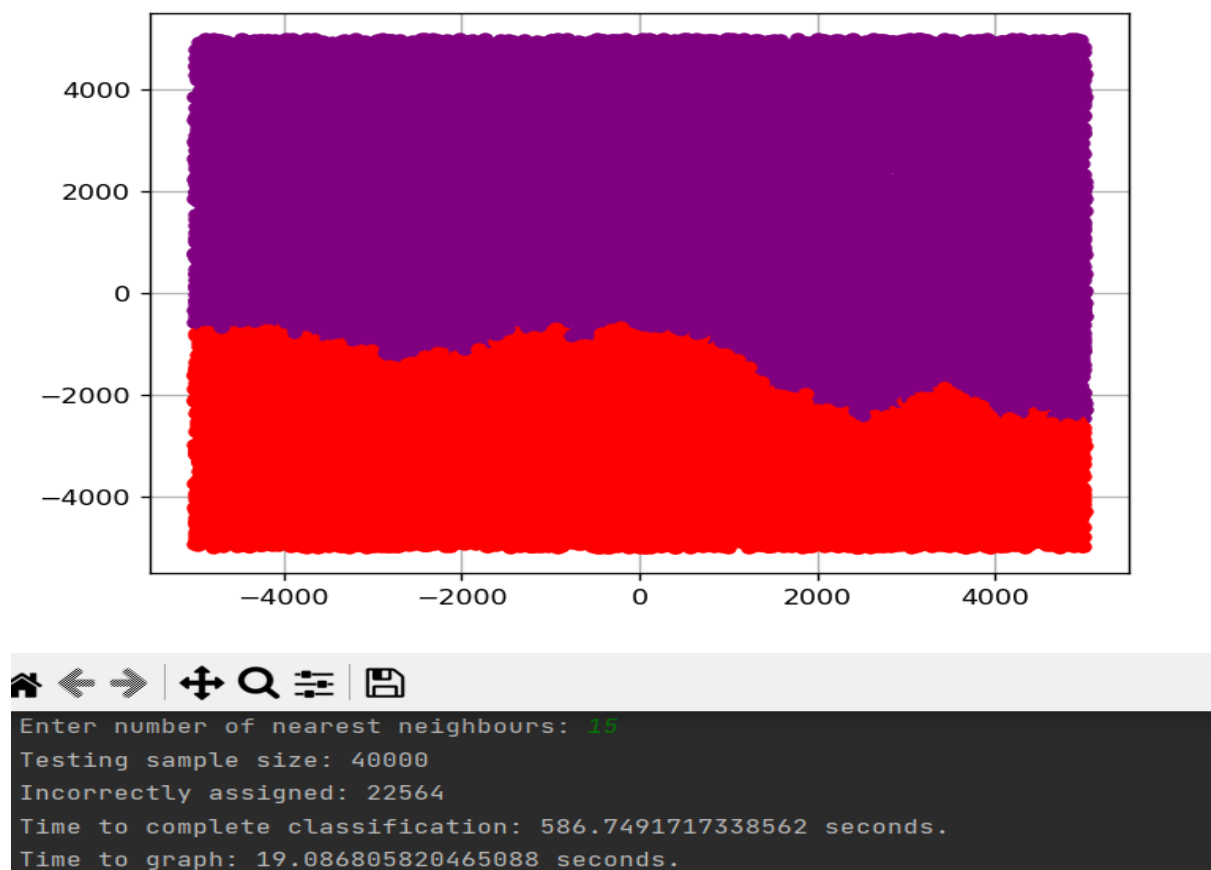


Figure 7 - The classification result for k=15

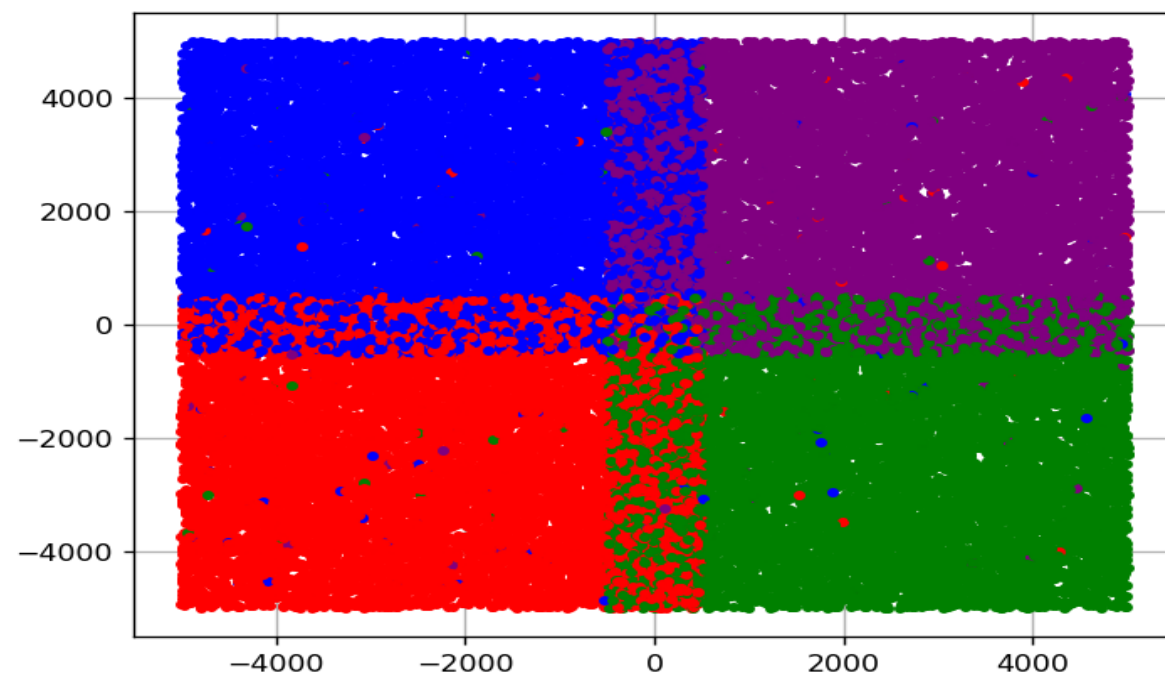


Figure 8 - The generated points for k = 15

Parameter k=15 (2):

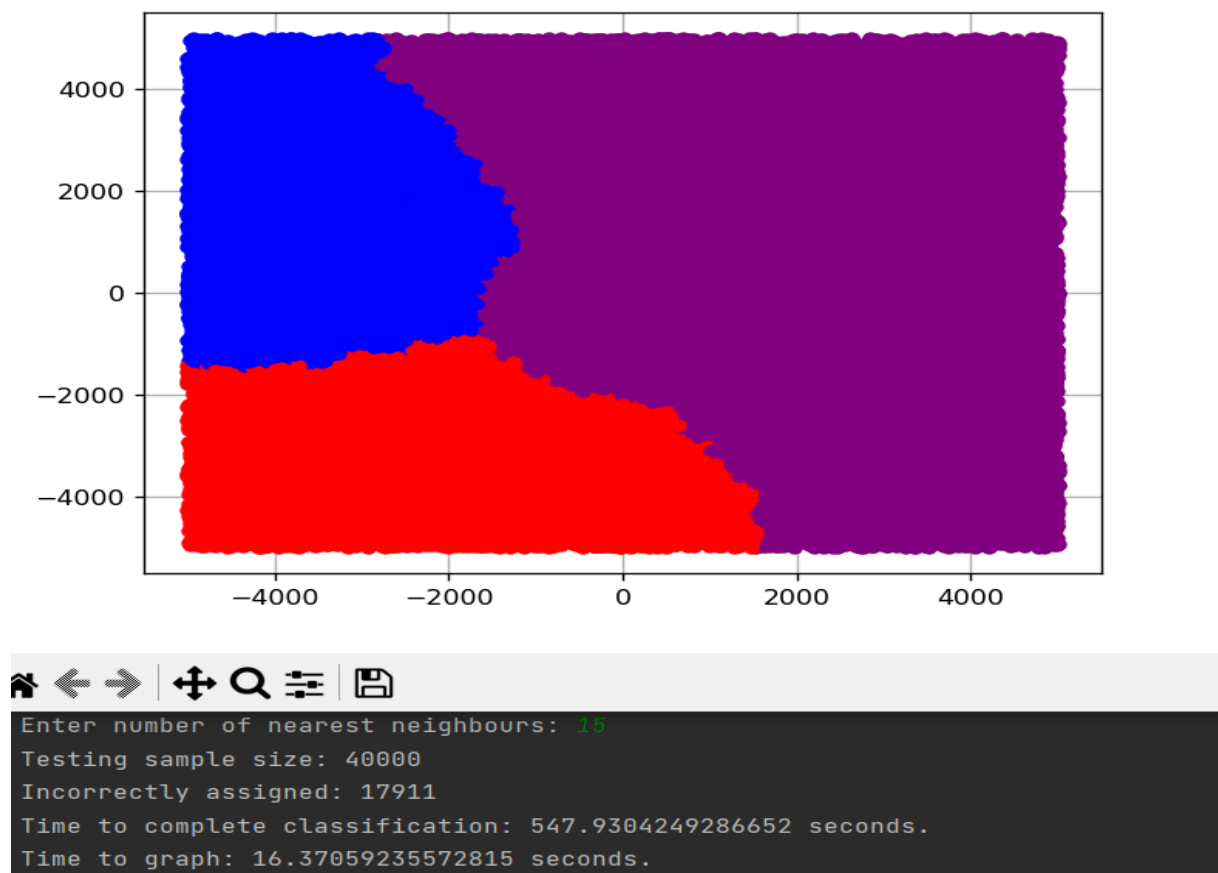


Figure 9 - The classification result for k=15

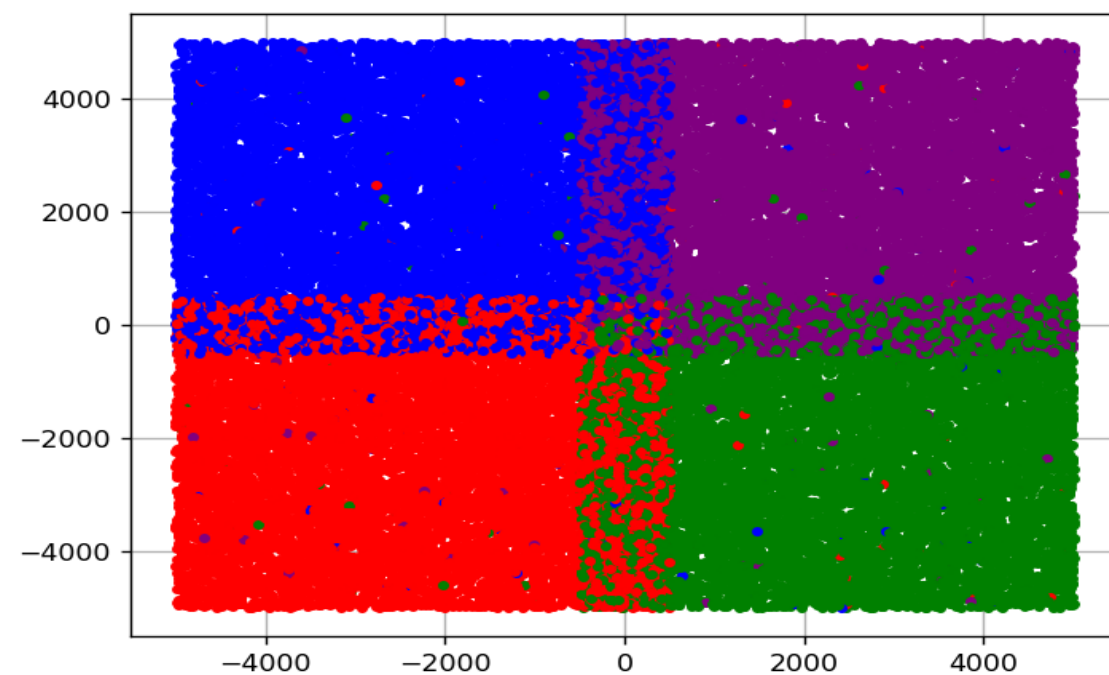


Figure 10 - The generated points for k = 15

Conclusion:

At the end of our testing, we can see that the classification differs in behaviour depending on the k-value. For k=1 the graph shows that the colour classification is quite dynamic and certain colours venture into other ones, as the new point colour is going to be the same as the closest point. As the k-value raises, the borders between colours are more defined – for k=3 there is not so much variation at the borders as in k=1, and for k=7 even less so. For k=15 graphs we can see that there is a possibility for one or two colours to completely disappear, as the classification of the first couple of points has very decisive consequences – we only have 20 points of training data at the beginning, so the occurrences of colours are going to equal - the purple colour is given to the point as it is the first mentioned. Therefore, for bigger k-values it would be important to enlarge the starting data size.

The accuracy of the classifier is a little less than 75% - for edge k-values (1,15) the accuracy is a bit lower compared to k-values (3,7) as there is a lot more variability. We can see that for some cases of k=15 the accuracy can be worse than 50%.

The efficiency of the program is decent, the program runs the classifier for 40 000 points in around 9 minutes without any additional optimization. The program does not use any arrays for space representation, instead the points have coordinates and the distance between them is calculated during classification using the Pythagorean theorem.