ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικό και Καποδιστριακό
Πανεπιστήμιο Αθηνών

Computational Geometry

Name: Foteini Tsavo

Registration Number: 1115201500206

Email: sdi1500206@di.uoa.gr

Exercise 1

Implement from scratch the k-NN algorithm in python programming language. Present your method and how you worked, conclude by discussing the disadvantages of the k-NN algorithm, if any.

The goal of machine learning is to make as accurate predictions as possible , using supervised or unsupervised learning. One of the most popular classification and regression algorithms used, is k-Nearest Neighbors algorithm.

The logic behind the classification of a query point is simple. An object is being classified based on the plurality vote of its neighbors, and to be more precise, based on the mode of the labels of its k neighbors. To determine the neighbors, a distance metric should be used, for instance : Euclidean Distance, Manhattan Distance, Hamming Distance, Cosine Distance or other metrics. Given a training set and a test set, the classification might be very different depending on the metric we choose to use.

For the implementation of the k-NN for classification we had followed some certain steps and those are written below:

1. Take a training set (points that are already labelled) and a query set (points that need to be labelled based on training examples)
2. Define k's value.
3. Determine the distance function that will be used.
4. Take test points, one by one, and compute their distances with all the training points.
5. Sort the distances and take k-nearest data samples.
6. Take the most popular label of those k elements that were chosen, and assign it to test point.

Also, the pseudocode written below is a great way of describing the logic of this classification method:

```
Input(X,Y) //X for queries, Y for training points
Input_Val(k)
List ← Initialized
For x in X: //for each test point:
        For y in Y: //for each training point:
                Distance = DistanceMetric(x,y)
                Add Distance to List
        End for
        Sort List
        Find k first elements of List
        L = Popular(k) //find which label is popular
        Classify(x, L) //assign this label to current test point
End for
```

Explaining the code:

To begin with, for the implementation of this project we have separated the code to three modules. The exercise01.py contains our main function, whereas the two other modules have the classes we used, in Dataset.py, and general functions that helped us, functions.py . The algorithm has been implemented for d dimensionality.

As the program runs successfully, the user is welcomed and is asked to determine if he/she desires to continue with the Iris.csv as dataset or the datasets that we provide, by choosing 1 for Iris, or else for our trainingSet.csv and testSet.csv . The difference between these two is that Iris.csv must be splitted, however the second way gives the opportunity to just insert two different files and do the work. After these action, the user is once again asked to enter the k value.

To continue with, in the Dataset.py module we have created some classes and some functions that are associated to these particular classes. The class named Element contains the information that was extracted from reading each row of the csv file. Furthermore, this class is part of Dataset class, which consists of all the elements. Furthermore, the classify function is responsible for labelling the query. Every time a test point is passed as a parameter, the computations happen, using the Euclidean distance metric, and the point takes the label that is most common. We do this for every query point in predictions function, and consequently we achieve the classification of the points we desired, presented in a new file, that has been created from the same function, predict.csv .

In the last module, functions.py we have written some functions that their use is for general purposes. Each function helped to build the algorithm. Finally , all of these are combined and the k-Nearest Neighbors classifies the data correctly, as well as writes the output to the file that we have mentioned before, predict.csv .

To sum up, we noticed that k-NN's algorithm performance was influenced by the distance function, the decision rule used to derive a classification and the number of neighbors, k which is inserted by the user. It is surely an easy algorithm to implement and understand. But there are some disadvantages. Firstly, the storage of the training set might be really large. Secondly, there are a lot of calculations need to be done so this influences the complexity and thirdly, as the dimension increases it becomes harder to classify correctly.

Exercise 2

Using the code provided at the class (or implementing your own) explain what is the curse of dimensionality and how is related to the k-NN algorithm.

The curse of dimensionality is a term, that was firstly, used by Bellman, in 1957. It refers to high dimensional spaces, and by that we mean a couple of hundreds or thousands dimensions. This particular situation arises in dynamic programming, data mining, machine learning, numerical analysis, combinatorial problems, time dependent problems, sampling, as well as control theory. The main problem regarding increased dimensionality, occurs when the volume of space increases really fast, therefore the data we have become sparse. If object appear to be sparse, they also will be dissimilar and that is not in advantage of algorithms such as k-NN, whose goal is to group the data properly. It makes their work inefficient .

Moreover, in Kevin Murphy's book "Machine Learning- A probabilistic perspective", the curse of dimensionality is explained:

"Consider the inputs are uniformly distributed along a D-dimensional unit cube. Suppose we estimate the density of class labels by growing a hyper cube around x until it contains the desired fraction f of the data points. The expected edge length of this cube is $e_D(f) = f^{1/D}$."

In this paragraph above, it is mirrored the unexpected behavior of distances in high dimensions. So when this happens, the points are not concentrated as they were in a lower dimension, as a result the length of the cube should be increased. Also, the most points seem to be placed at the boundary of the cube.

Furthermore, the code provided at class explains why the k-nearest neighbor algorithm is not that efficient when it comes to higher dimension. Firstly, it generates random points in a d-dimensional cube and then their distances will be computed but for different dimensions from 1 up to 200.

Based on the printed results that we have, we notice that as the dimension increases, the minimum distance increases gradually, which the point that we had above, that data becomes sparse. Even more, when we take a look at the mean(distance) column, it is shown that with increase of the dimension the mean distance of points increases a bit faster. The min distance ratio with mean distance, looks like having an unexpected behavior, but what is mostly observed is that when the dimension increases the ratio will be closer to 1. As a result, it seems that for higher dimensions mean distance is closer to minimum distance.

All in all, the curse of dimensionality is an obstacle for k-NN algorithm because the phenomenon that arise is that the ratio between the nearest and the farthest point approaches 1. So this makes it hard for the algorithm to distinguish which points are relevant, because its criteria is based on the distance. Normally the points that are closer are more relevant and those, which are further are not. But in the case of higher dimensionality, this might be difficult to accomplish, however the metric that is used.

Exercise 3

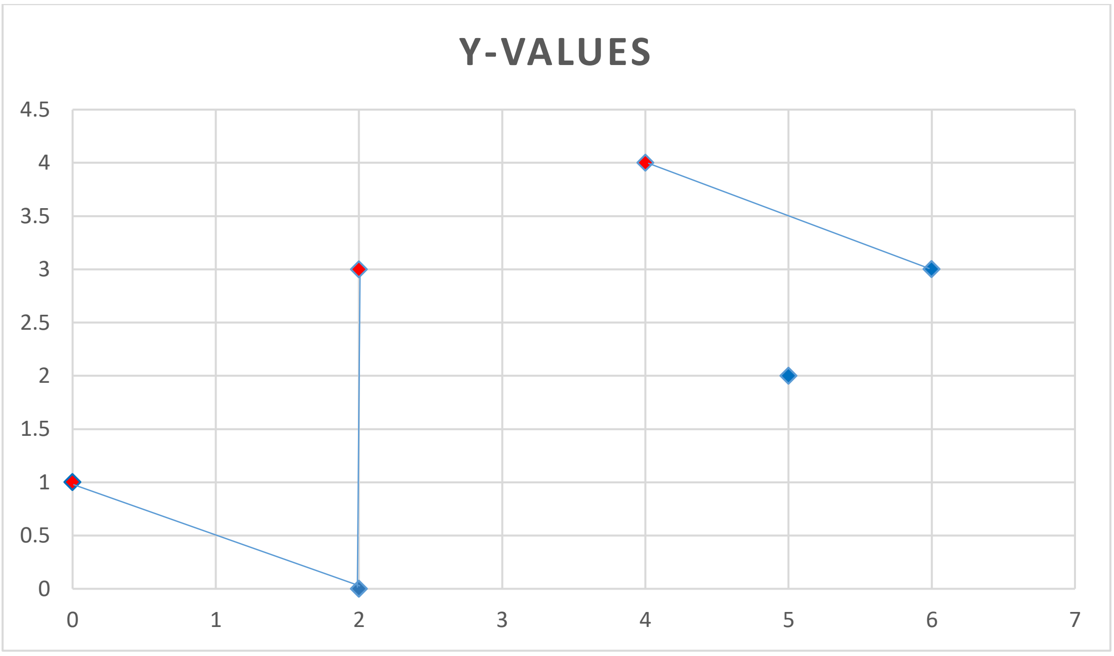
1) Suppose there is a set of points on a two-dimensional plane from two different classes. Points in class Red are (0, 1), (2, 3), (4, 4) and points in class Blue are (2, 0), (5, 2), (6, 3). Draw the k-nearest-neighbor decision boundary for k = 1 as we discussed in the lecture. Experiment yourself with two or more different distance metrics. Present your results.

The Nearest Neighbor algorithm decides how the current query point will be labelled, based on the distance of the training point that happens to be closer to it. When we classify a certain point, it automatically becomes part of a class that is in one side of the decision boundary. Specifically, a decision boundary is the region of a problem space in which the output label is ambiguous. To explain this further, this decision boundary is created from all the lines that are exactly in the middle of two points of opposite classes.

Firstly, in order to draw a decision boundary for two classes we have to examine the local information. By that we mean the points of a class that appear to be close with points of the opposite class. So we will focus on the regions where we think decision boundaries should occur. Then, we are going to connect the opposite labels we have found, forming a line. After that we will draw perpendicular bisectors of these lines, extend them all and join bisectors.

In our case, we are given points that are part of two classes, Red or Blue. We have drawn them in the Cartesian coordinate system of two dimensions, below. The color of the points represents the class they belong to. Next, we will draw the straight lines between red and blue points. We connect the opposite points that seem to be closer based on **Euclidean Distance.**

$d = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2}$	(0,1)	(2.3)	(4,4)
(2,0)	**2.236068**	**3**	4.472136
(5,2)	5.09902	3.162278	**2.236068**
(6,3)	6.324555	4	**2.236068**



Now our task is to find the equation of the perpendicular bisector. Math formulas we are going to use to achieve that are written below:

$M = (\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$ → The point that is in the middle of a red and blue point

$m = \frac{y_2-y_1}{x_2-x_1}$ → The slope formula

m'→ The slope of the perpendicular line, we need to take the opposite reciprocal

$y = m'x + b$ → The equation of a straight line

To start with, we will calculate the perpendicular line for the first two pairs that we have (0,1) and (2,0).

$M = (\frac{0+2}{2}, \frac{1+0}{2}) = (1, 0.5)$

$m = \frac{0-1}{2-0} = -\frac{1}{2}$

m' = 2

$y = 2x + b$ What we need to do now is to find the value of b. As we know point M is part of this line so we are going to solve the equation:

$0.5 = 2 + b$ → $b = -1.5$

As a result the equation of the line is $\boldsymbol{y = 2x - 1.5}$

For the second pair of points (2,3) and (2,0) we should do the same calculations.

$M = (\frac{2+2}{2}, \frac{3+0}{2}) = (2, 1.5)$

In this case, we need no further calculations because it is obvious that the perpendicular line is $\boldsymbol{y = 1.5}$

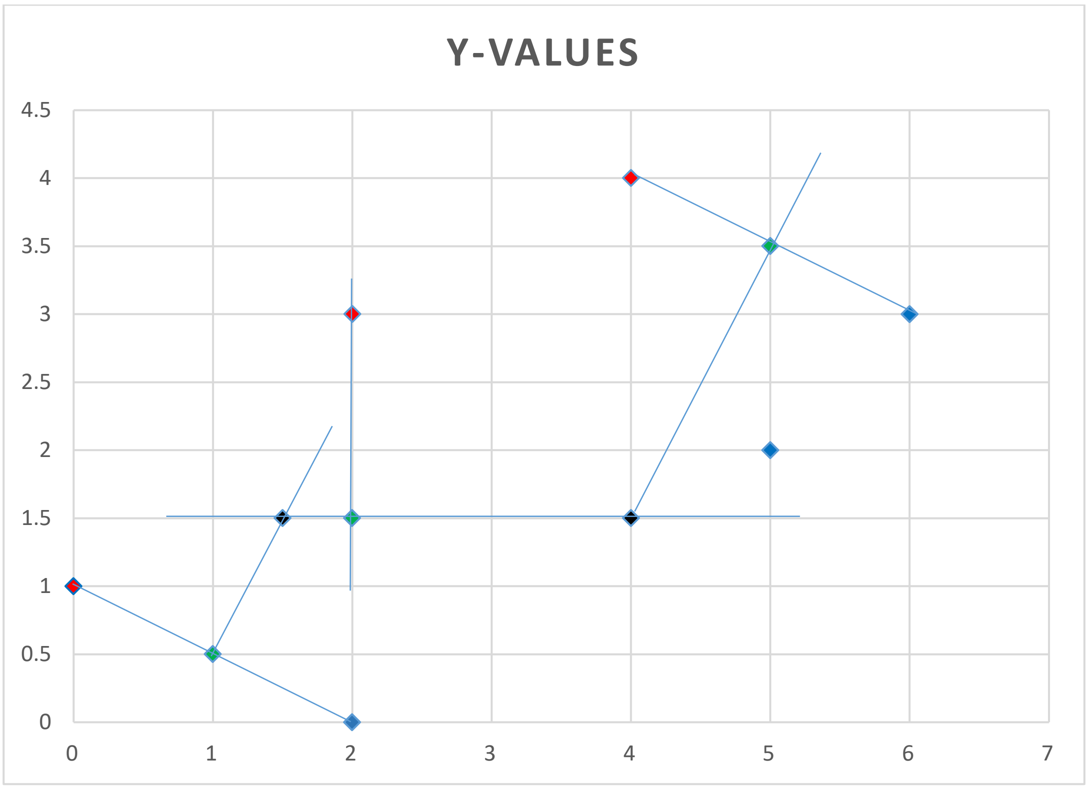
Finally, we will calculate for the third pair (4,4) and (6,3).

$M = (\frac{4+6}{2}, \frac{4+3}{2}) = (5, 3.5)$

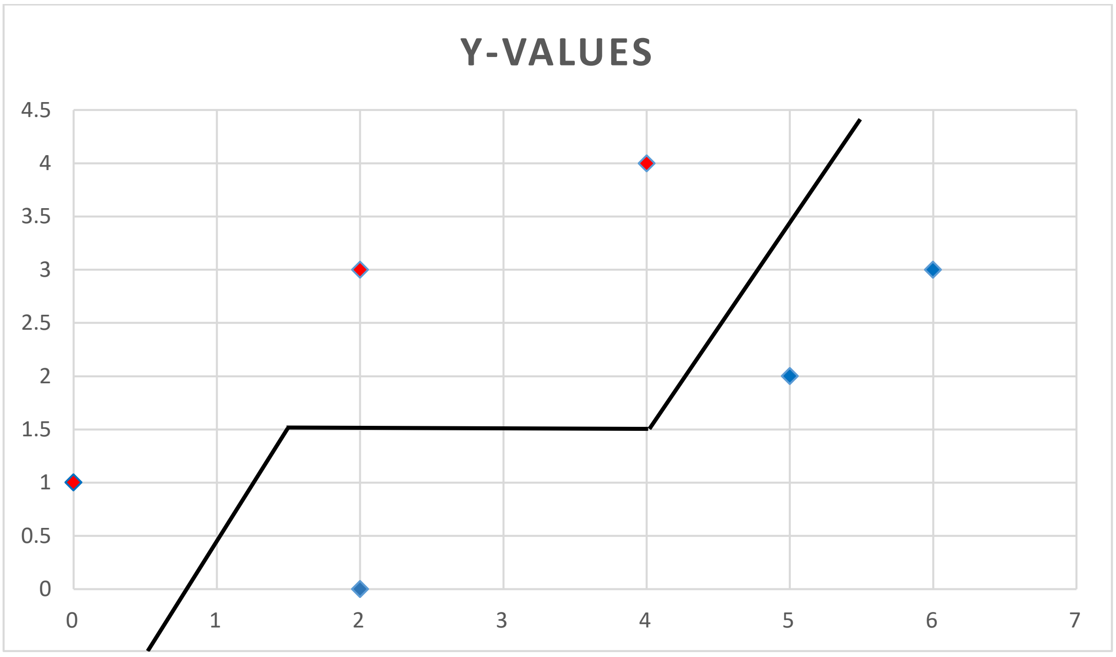
$m = \frac{3-4}{6-4} = -\frac{1}{2}$

$m' = 2$

$y = 2x + b \rightarrow 3.5 = 2 * 5 + b \rightarrow b = -6.5 \rightarrow \boldsymbol{y = 2x - 6.5}$



As you can see, black points are those where the perpendicular lines intersect and this helps us understand the decision boundary. Moreover, it helps us to draw it as follows:



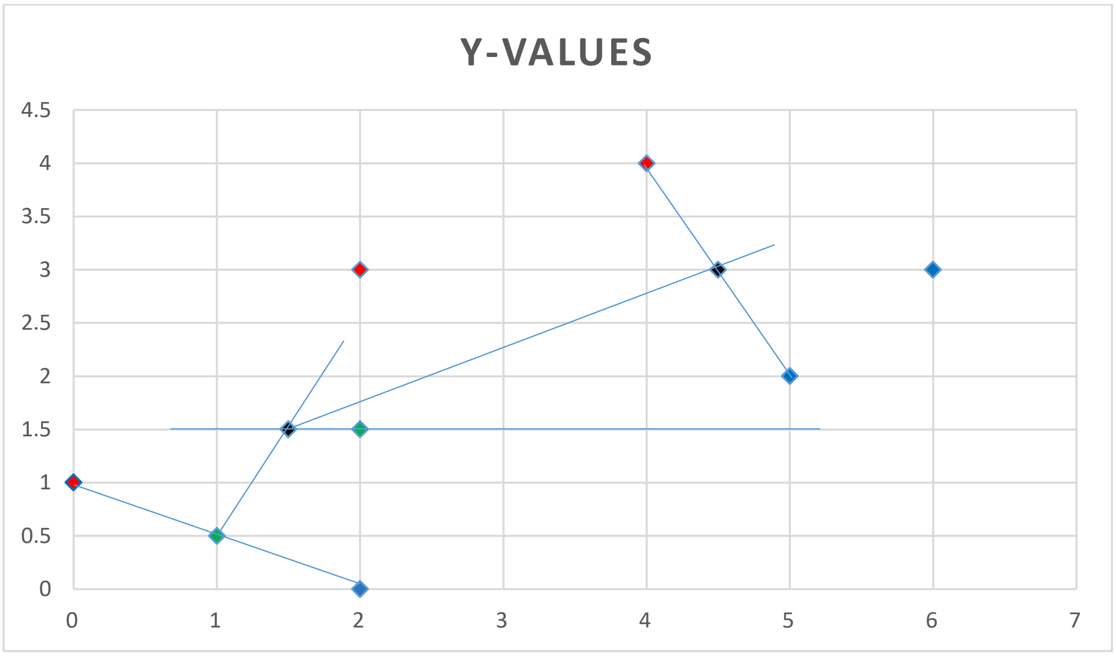
As we notice in the Euclidean Distance board below, we have two blue points that their distances to the red point (4,4) are equal. Now we will examine the case where (5,2) is chosen over (6,3).

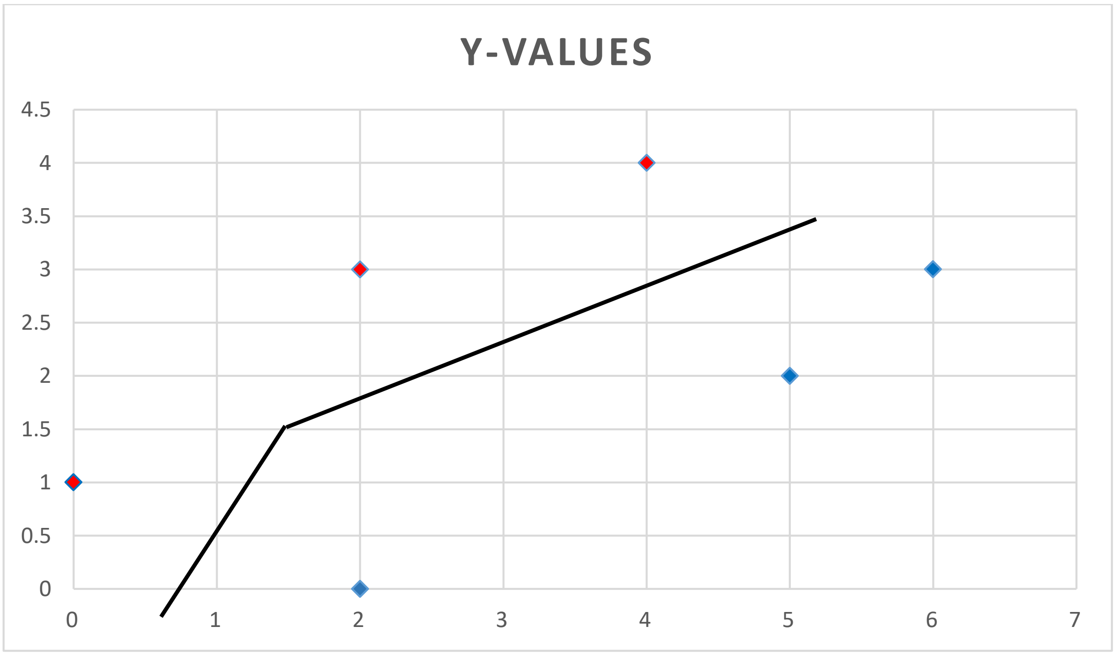
$M = (\frac{4+5}{2}, \frac{4+2}{2}) = (4.5,3)$

$m = \frac{2-4}{5-4} = -2$

$m' = \frac{1}{2}$

$y = \frac{1}{2}x + b \rightarrow 3 = \frac{1}{2} * 4.5 + b \rightarrow 3 = 2.25 + b \rightarrow b = 0.75 \rightarrow \boldsymbol{y = \frac{1}{2}x + 0.75}$





This decision boundary looks smoother than the first one.

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικό και Καποδιστριακό
Πανεπιστήμιο Αθηνών

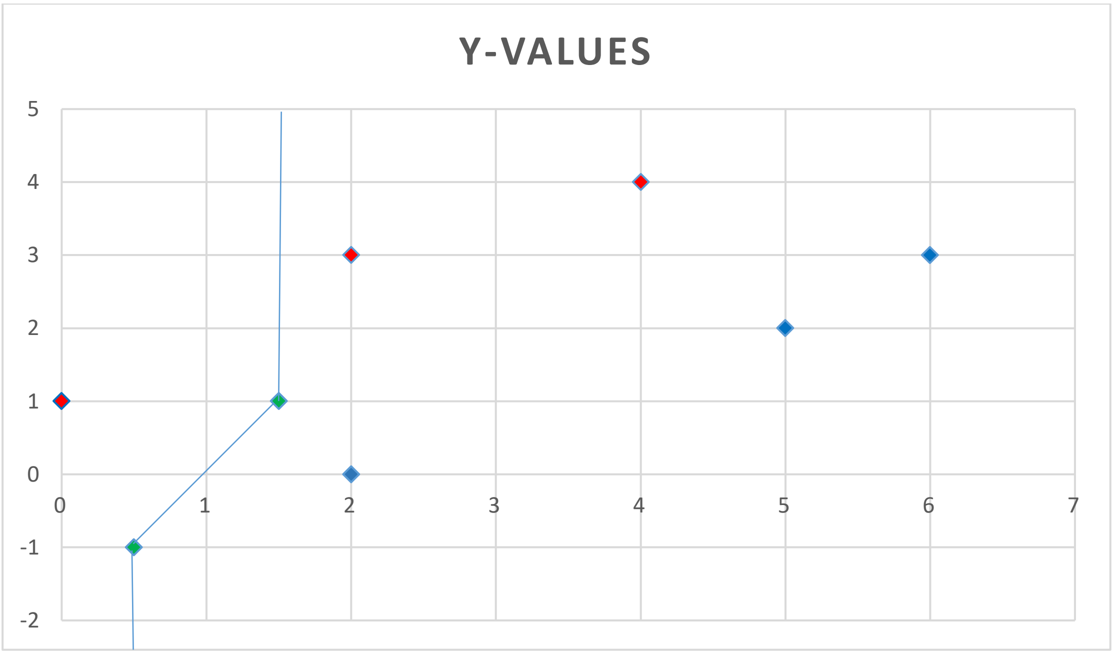
Using **Manhattan Distance** as a metric:

$\lvert x_1 - x_2\rvert + \lvert y_1 - y_2\rvert$	(0,1)	(2.3)	(4,4)
(2,0)	**3**	**3**	6
(5,2)	6	4	**3**
(6,3)	8	4	3

In Taxicab Geometry the perpendicular bisector is defined as the set of all points that are equidistant from two certain points. The equation of the line we will be searching is the following: $\lvert x - x_1\rvert + \lvert y - y_1\rvert = \lvert x - x_2\rvert + \lvert y - y_2\rvert$ for the example points $A(x_1, y_1)$, $B(x_2, y_2)$.

For the first pair of points (0,1) and (2,0).

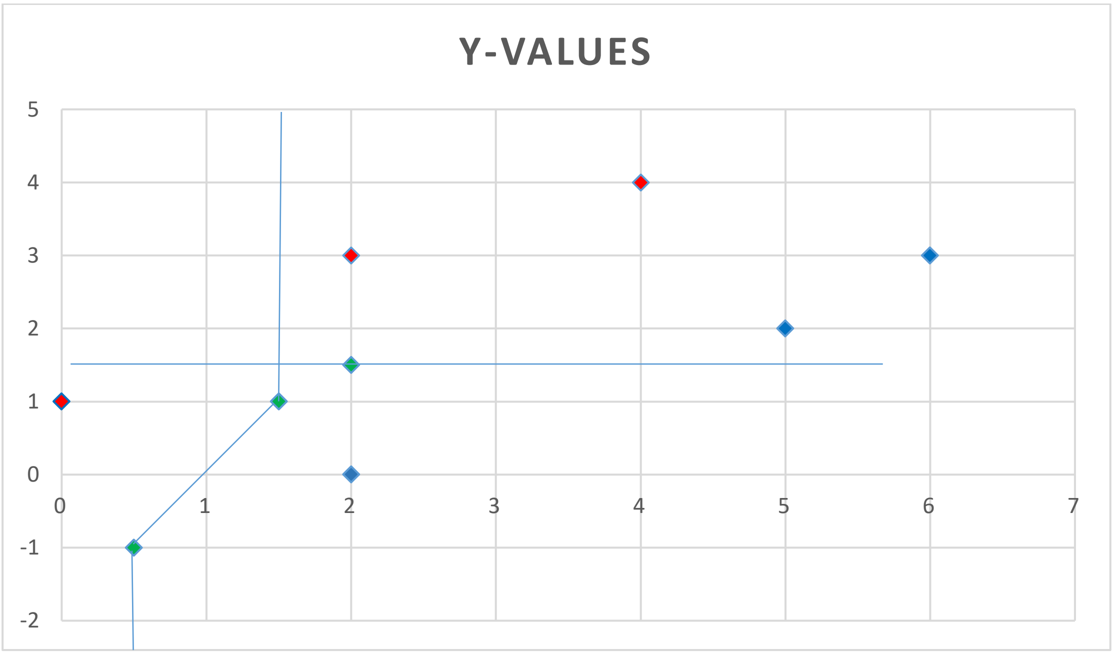
$$\lvert x\rvert + \lvert y - 1\rvert = \lvert x - 2\rvert + \lvert y\rvert$$



Next pair of points (2,3) and (2,0).

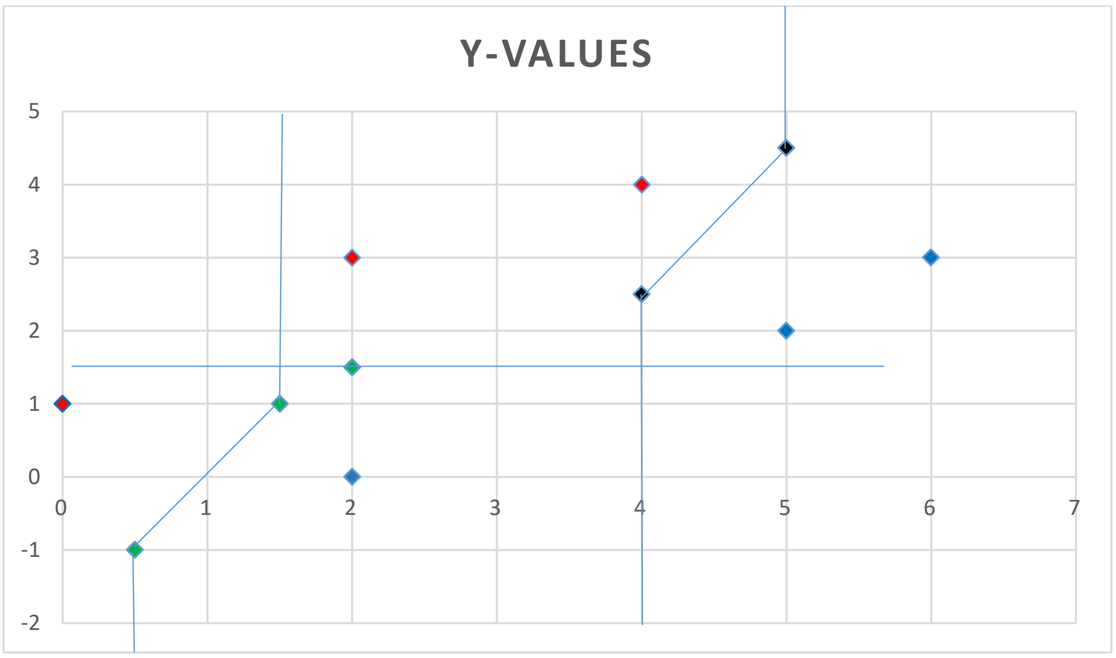
$$|x-2|+|y-3|=\ |x-2|+|y|$$

In this case we have Euclidean's distance perpendicular bisector same as Taxicab's due to the pair's positions.

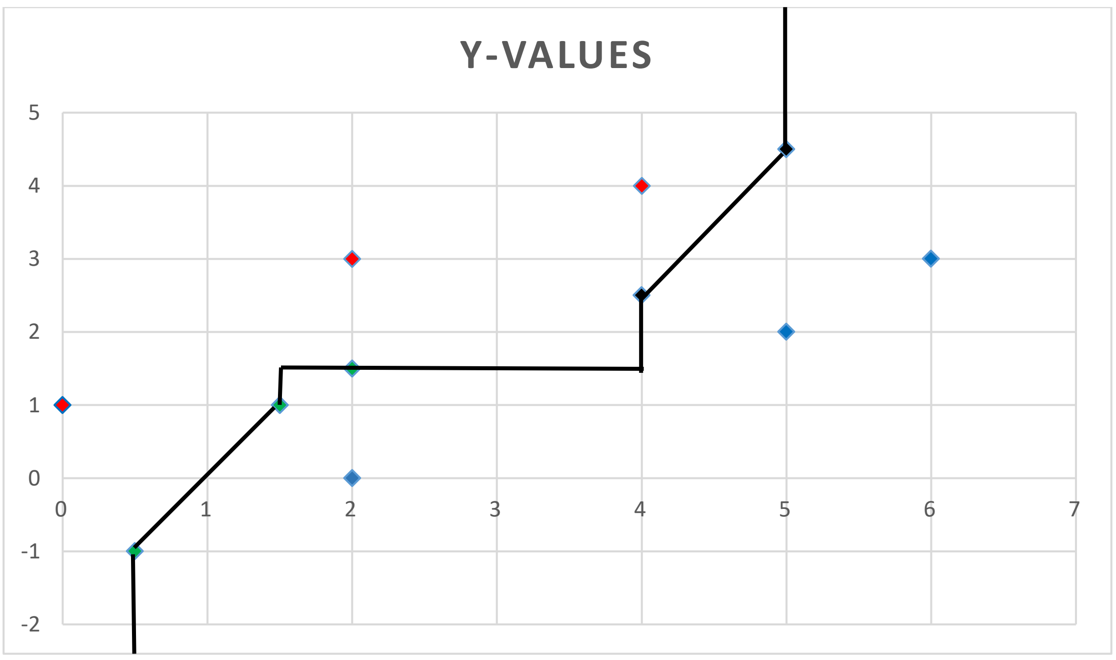


Finally, we present (4,4) and (5,2).

$$|x-4|+|y-4| = |x-5|+|y-2|$$



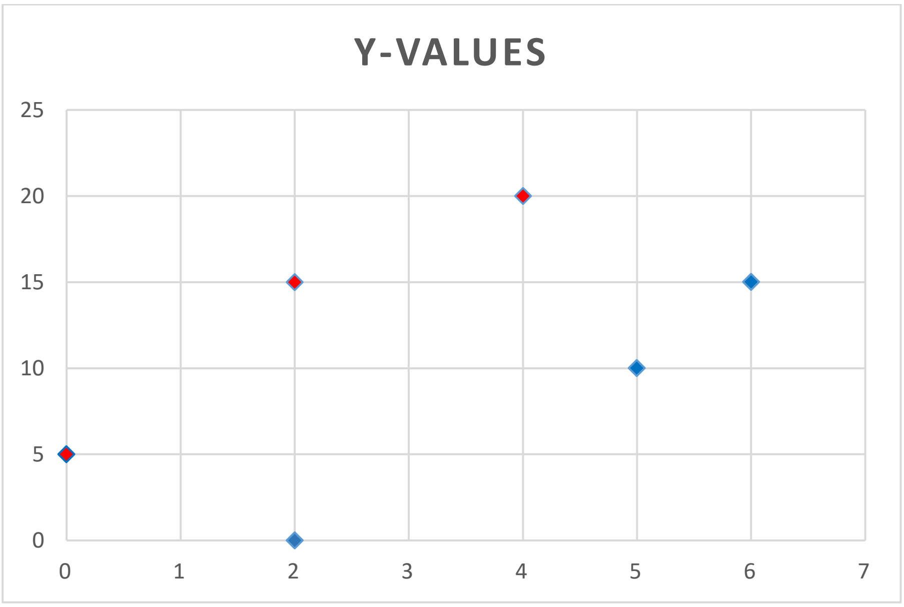
We are now in the position to draw the decision boundary using Manhattan Distance metric.



The conclusion is , that the distance metric we choose affects the decision boundary. Different metrics will have different boundaries and as a result the classification will not be quite similar.

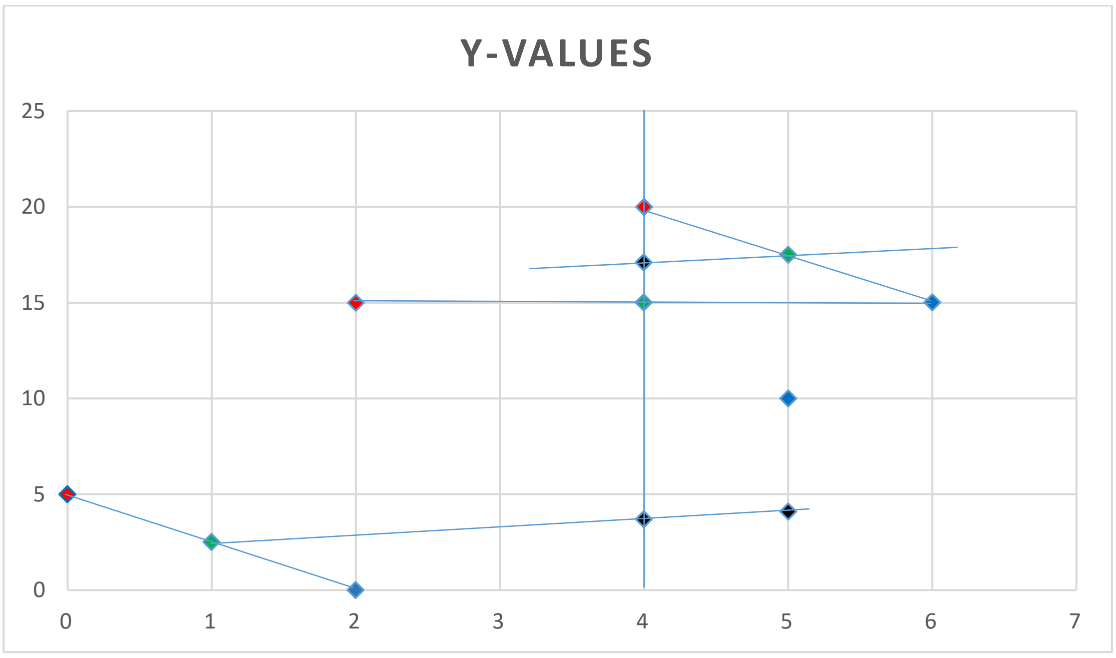
2) If the y-coordinate of each point was multiplied by 5, what would happen to the k = 1 boundary? Draw a new picture. Explain whether this effect might cause problems in practice.

To begin with, we have multiplied the coordinate y of each point by 5 and presented the results in the chart below. Now let's us draw the board of distances, in order to distinguish which points are closer or further.



Euclidean Distance:

$d = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2}$	(0,5)	(2.15)	(4,20)
(2,0)	**5.385165**	15	20.099751
(5,10)	7.071068	5.830952	10.049876
(6,15)	11.661904	**4**	**5.385165**



Very first pair of Red and Blue points:

$M = (\frac{0+2}{2}, \frac{5+0}{2}) = (1{,}2.5)$

$m = \frac{0-5}{2-0} = -\frac{5}{2}$

$m' = \frac{2}{5}$

$y = \frac{2}{5}x + b \rightarrow 2.5 = \frac{2}{5} + b \rightarrow b = 2.1 \rightarrow y = \frac{2}{5}x + 2.1$

For the pair of points (2,15) and (6,15), it is quite obvious that $M = (4{,}15)$ and the perpendicular is the one drawn above.

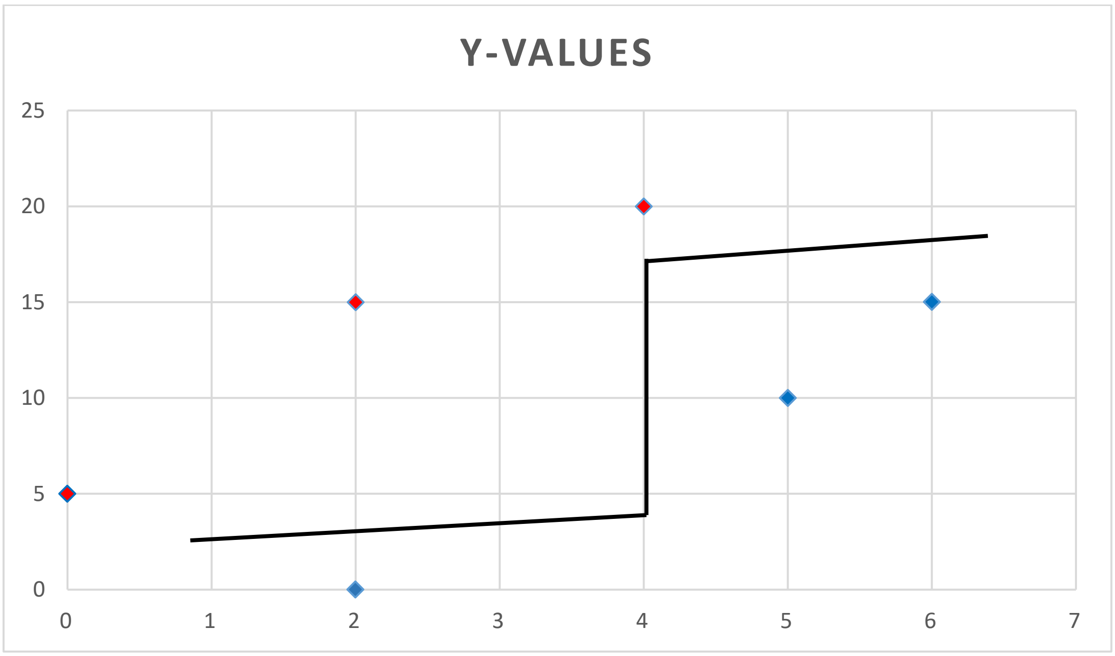
Final pair:

$M = (\frac{4+6}{2}, \frac{20+15}{2}) = (5{,}17.5)$

$$m = \frac{15-20}{6-4} = -\frac{5}{2}$$

$$m' = \frac{2}{5}$$

$$y = \frac{2}{5}x + b \rightarrow 17.5 = \frac{2}{5} * 5 + b \rightarrow b = 15.5 \rightarrow y = \frac{2}{5}x + 15.5$$

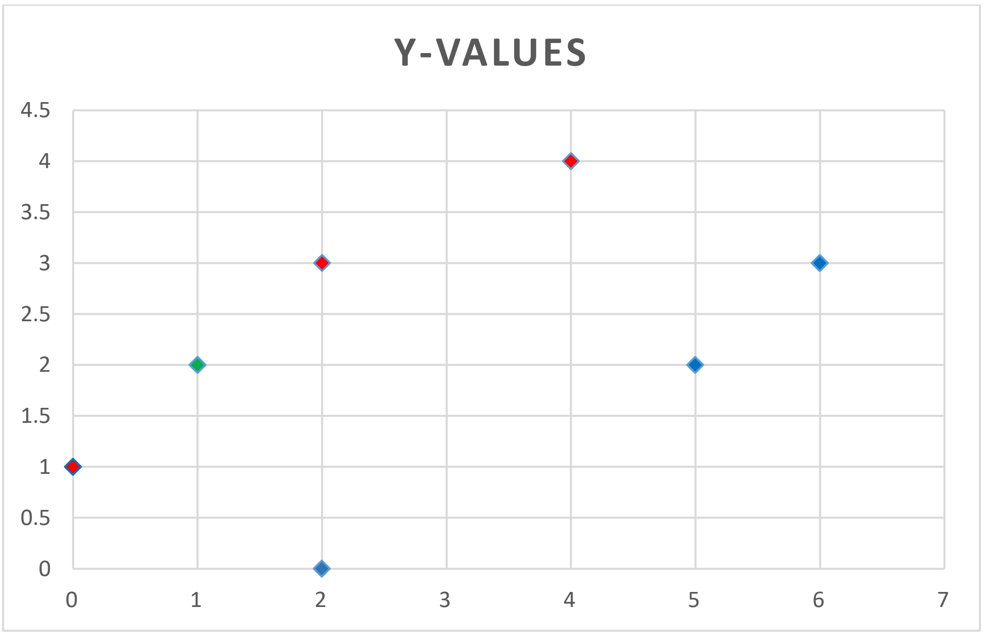


Firstly, as we increase the value of y coordinate for each point, our distances are being increased too. As a result, these points are far from each other, so when a new query needs to be inserted in one of the two neighborhoods using 1-NN, the classification might not be very precise. If the data are far, the classification is not very real and the boundary will be not that helpful to distinguishing the neighborhoods.

In general, the 1-NN decision boundary is not that smooth, but it is flexible. This means that it can drastically change when points are inserted. For instance, if a point is falsely classified and is a Red one in the Blue neighborhood, than if another point is inserted in this neighborhood but its closest point is the red one, it will get its label. This example is perfect for describing the noise that appears in the data during the classification.

4) Suppose now we have a test point at (1, 2). How would it be classified under 3-NN? Given that you can modify the 3-NN decision boundary by adding points to the training set in the diagram, what is the minimum number of points that you need to add to change the classification at (1, 2)? Provide also the coordinates for these new points and justify your answer.

In this case, we are going to classify the given point using the 3-NN algorithm. The main difference between 1-NN and 3-NN is obviously the k value. This time we are not searching only for the closest point, but for the three closest points that exist near our query point, and the label that will be assigned to it will be the one that is most common among its neighbor.



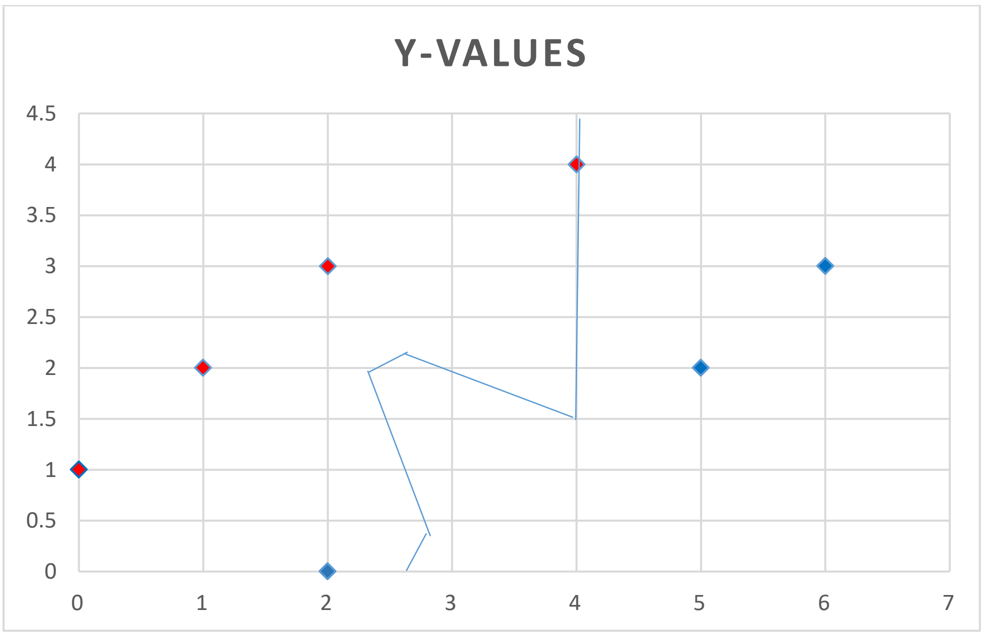
By just looking at the Cartesian coordinate system above, we are able to predict that the point will be inserted as a member of the Red Class, as he seems closer to Red points. On the other hand, the algorithm does not have this ability and reads only coordinates. Consequently, the k-NN will compute the distance of the point to all the training points, then, will take into consideration only the 3 points that have the smallest distance. Finally, it will decide which label is appropriate for the new point based on the labels of its neighbors.

Using **Euclidean Distance** metric:

$d = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2}$	(1,2)
(0,1)	**1.414214**
(2,3)	**1.414214**
(4,4)	3.605551
(2,0)	**2.236068**
(5,2)	4
(6,3)	5.09902

The k points that are neighbors with query point (1,2) are (0,1) , (2,3) and (2,0). The two of them belong to the Red Class and the last one is Blue. As a result, the point (1,2) will be inserted to the Red class, as we have predicted.

As far as the decision boundary is concerned, it is a useful information that helps us distinguish immediately whether the point belongs to the Red or the Blue class. If we just draw the point in the Cartesian coordinate system, we can see from which side of the boundary the point is. Therefore, we are able to answer really fast about its classification.



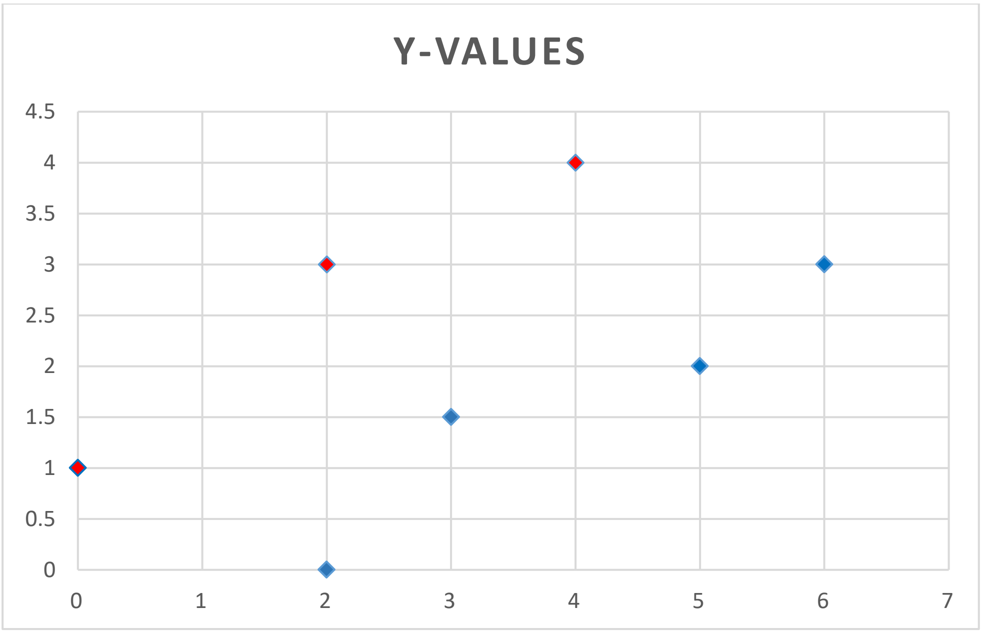
Let us suppose that we have not inserted (1,2) yet and there are some points before this particular one, that need to be classified. The task we have is simple and clear: When the time comes for the (1,2) to be inserted as a test query , it should be labelled as a Blue one. By adding new points, probably, blue ones, in certain positions, we will be able to change the flexible boundary of 3-NN and reach our goal.

To start with, at least two blue neighbors must be placed at distance smaller than **1.414214** from the test example. This fact is based on the study of the Euclidean Distance board above, where (1,2) appears to be in the middle of two red points. But if we insert directly a point that is in the left side of the boundary, it will be classified as red immediately and we will achieve nothing.

So we start by adding the point (3,1.5) and measuring its distance from every other point of the training set.

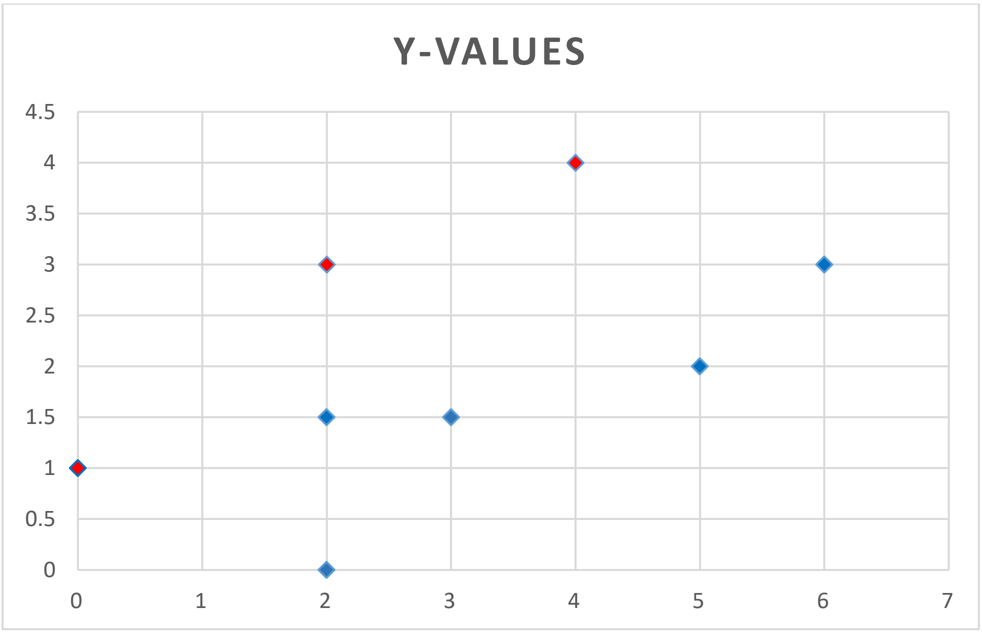
$d = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2}$	(3,1.5)
(0,1)	3.041381
(2,3)	**1.802776**
(4,4)	2.692582
(2,0)	**1.802776**
(5,2)	**2.061553**
(6,3)	3.354102

It seems that the new point with coordinates (3,1.5) will be classified as Blue, thus the decision boundary will change.



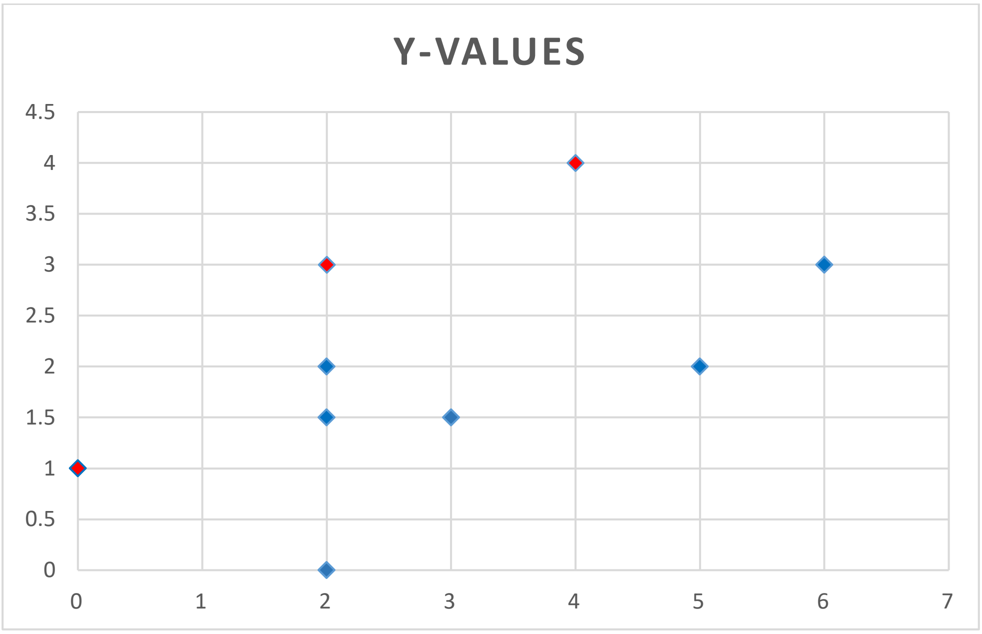
Next , (2,1.5) will be inserted.

$d = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2}$	(2,1.5)
(0,1)	2.061553
(2,3)	**1.5**
(4,4)	3.201562
(2,0)	**1.5**
(5,2)	3.041381
(6,3)	4.272002
(3,1.5)	**1**

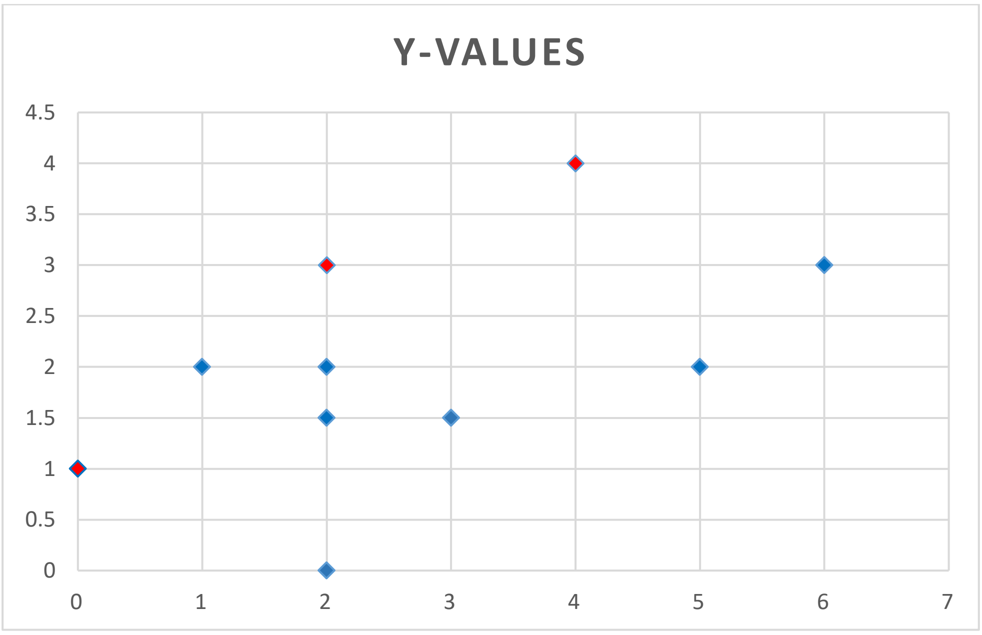


Third point waiting for insertion is (2,2).

$d = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2}$	(2,2)
(0,1)	2.236068
(2,3)	**1**
(4,4)	2.828427
(2,0)	2
(5,2)	3
(6,3)	4.123106
(3,1.5)	**1.118034**
(2,1.5)	**0.5**



Finally, let us insert (1,2) and see how it will be classified now. Its three closest neighbor will be (0,1) , (2,1.5), (2,2) with distances 1.414214, 1.118034 and 1 accordingly.



All in all, the test point (1,2) was labelled as Blue, after the insertion of **three** blue points [(3,1.5) , (2,1.5) , (2,2)] and with at least two of them placed as near as possible, or at least nearer than the two red elements. Note that we have not illustrated the boundary because it needs to be precise. But it is important to mention that the boundary has really changed, and was modified with each insertion we have implemented.

Exercise 4

How long does it take for k-NN to classify one point? Or in other words what is the testing complexity for one instance? Assume your data has dimensionality d, you have n training examples and use Euclidean distance. Assume also that you use a quick select implementation which gives you the k smallest elements of a list of length m in O(m).

As a lazy learning algorithm, k-NN does not need any training data points in order to generate a model. The training phase is only about storing the data as well as the labels. Those will be used in the testing phase, where the query point will be classified by assigning the label that appears more frequently in the k nearest points of the training set. Consequently, the testing phase will be slower than training phase, due to more computations.

As long as the testing phase is concerned, the distances of the query point from all the points that are included in the training set, should be computed using the Euclidean distance. Each distance computation requires $O(d)$ runtime, therefore $O(nd)$ time is needed for all the distances to be calculated, from the n training points to the current test point. Also, using a quick select implementation gives us the advantage of finding the k elements with the

```
For each point p in training set:                          $O(n)$
        Find Euclidean distance with test point q          $O(d)$
        Store the distances in a list
                                                           $O(nd)$

Run a quick select algorithm:                              $O(m)$
        Return the k smallest elements
                                                   $O(nd + m)$ = $O(nd)$
```

smallest distances in $O(m)$ time. This leaves us with total complexity of $(nd + m)$, where m ≤ n, because this particular list contains the distances which are n in number. In this case, we can simplify our answer, because of the fact that m ≤ n, to $O(nd)$.

The approach of k-NN with quickselect, is an algorithmic improvement that takes advantage of the fact that, there exist efficient ways to find the k^{th} smallest value in an array which might be unsorted or an unordered list. This algorithm is similar to QuickSort algorithm, but the main difference between them is that instead of checking for both sides, which pivot defines, it only recurs for the one part of the array that contains the k^{th} element.

Exercise 5

To see an application of the k-NN algorithm in a real world classification problem consider the data found at https://www.kaggle.com/uciml/iris. Download from there the Iris.csv file. Ignore the id column and consider the columns: SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm as point coordinates in a four dimensional space. Consider the column Species as the class/label column. The file contains 150 rows. Run the algorithm on the first 100 rows and make predictions for the rest 50. How your predictions are compared to the actual? Explain your methodology.

The Iris.csv file was downloaded and is placed inside the Erxercise01 folder with all the python files that we had implemented for the first exercise. Therefore, we not only can see just the application of k-NN in a real world problem, but we also test the algorithm written by us. The results of the classification are included in a file named predict.csv .

If we open this particular file, we are able to see that all these point are classified as iris-versicolor, which is not what Iris,csv file suggests. This might have happened because the label iris-virginica is not known from the algorithm at the moment. No point of the training set, meaning the first hundred rows, is member of this class. As a result, k-NN will just do the calculations and find a label that seems closest to these objects. It looks like the iris-versicolor class has a greater similarity with iris-virginica class. This is based on the Euclidean distance of the features that we have.

Summarizing, the k-NN algorithm has chosen a label that already existed in the training set. To be more precise, it has considered the characteristics/features of the elements as vectors with the dimension equal to four and has done the computations based on Euclidean Distance. All of this had led to the result we see in the prediction.csv file.

References:

- Prof. Bob Berwick, **K-NN and ID Tree Notes, Cliff Notes Version,** MASSACHUSETTS INSTITUTE OF TECHNOLOGY Department of Electrical Engineering and Computer Science 6.034 Artificial Intelligence, Fall 2011.
- Panagiotis Cheilaris, Sandeep Kumar Dey, Maria Gabrani, and Evanthia Papadopoulou, **Implementing the L∞ Segment Voronoi Diagram in CGAL and Applying in VLSI Pattern Analysis,** Faculty of Informatics, Universit`a della Svizzera Italiana, Switzerland 2 IBM Zurich Research Laboratory, Switzerland
- Olga Veksler, **Machine Learning in Computer Vision,** CS840a
- Stack Exchange website: https://stats.stackexchange.com/questions/219655/k-nn-computational-complexity
- Towards Data Science website : https://stats.stackexchange.com/questions/219655/k-nn-computational-complexity
- Marine Carpuat, **Classification with Nearest Neighbors**, CMSC422
- Towards Data Science website : https://towardsdatascience.com/importance-of-distance-metrics-in-machine-learning-modelling-e51395ffe60d
- Sebastian Raschka, **Machine Learning Lecture Notes,** Department of Statistics University of Wisconsin–Madison
- Analytics Vidhua, **30 Questions to test a data scientist on K-Nearest Neighbors (kNN) Algorithm**
- Ebook: Viren Tellis, **Manhattan Prep GMAT, Geometry**
- Hiroshi Shimodaira, **Classification and K-Nearest Neighbours**
- Jim Wilson, **Taxi Cab Geometry with Technology : Some exploration materials**, The University of Georgia