python

Class: Machine Learning

Topic

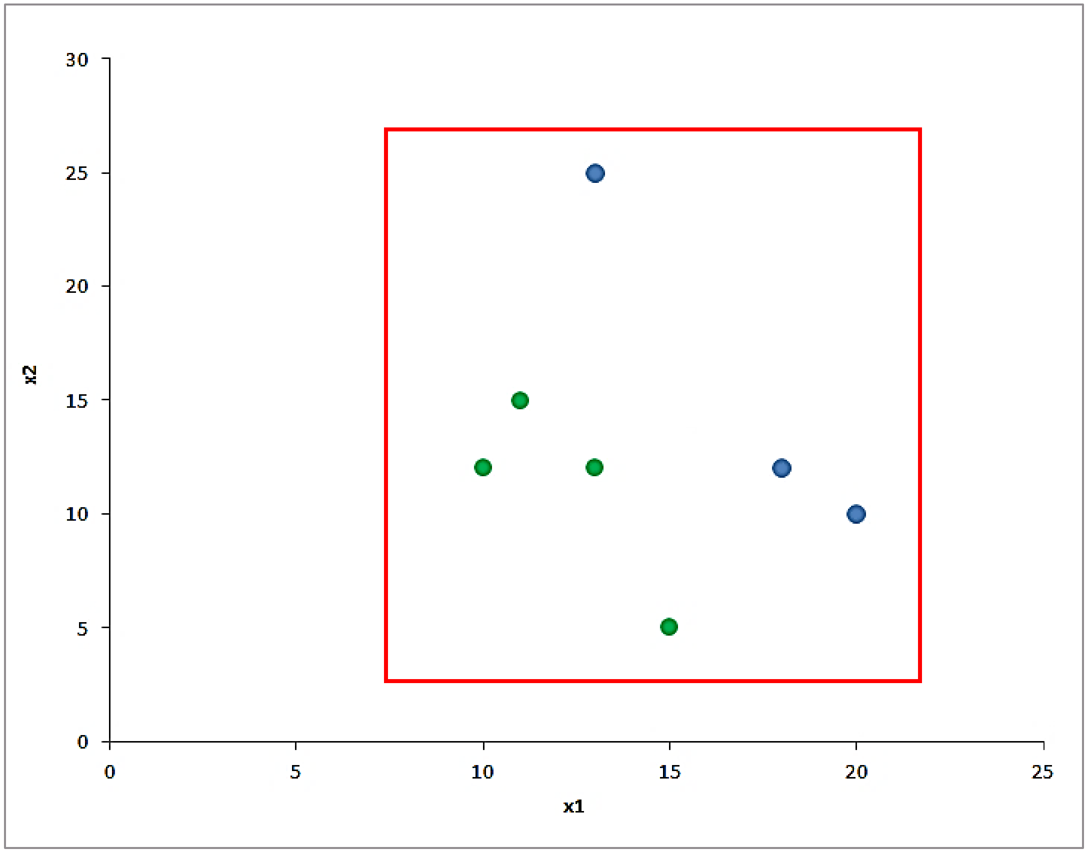
K-Nearest Neighbor and Summing Up the End-to-End Workflow

Example of a Classification Algorithm

K-Nearest Neighbor

K-Nearest Neighbor

Example

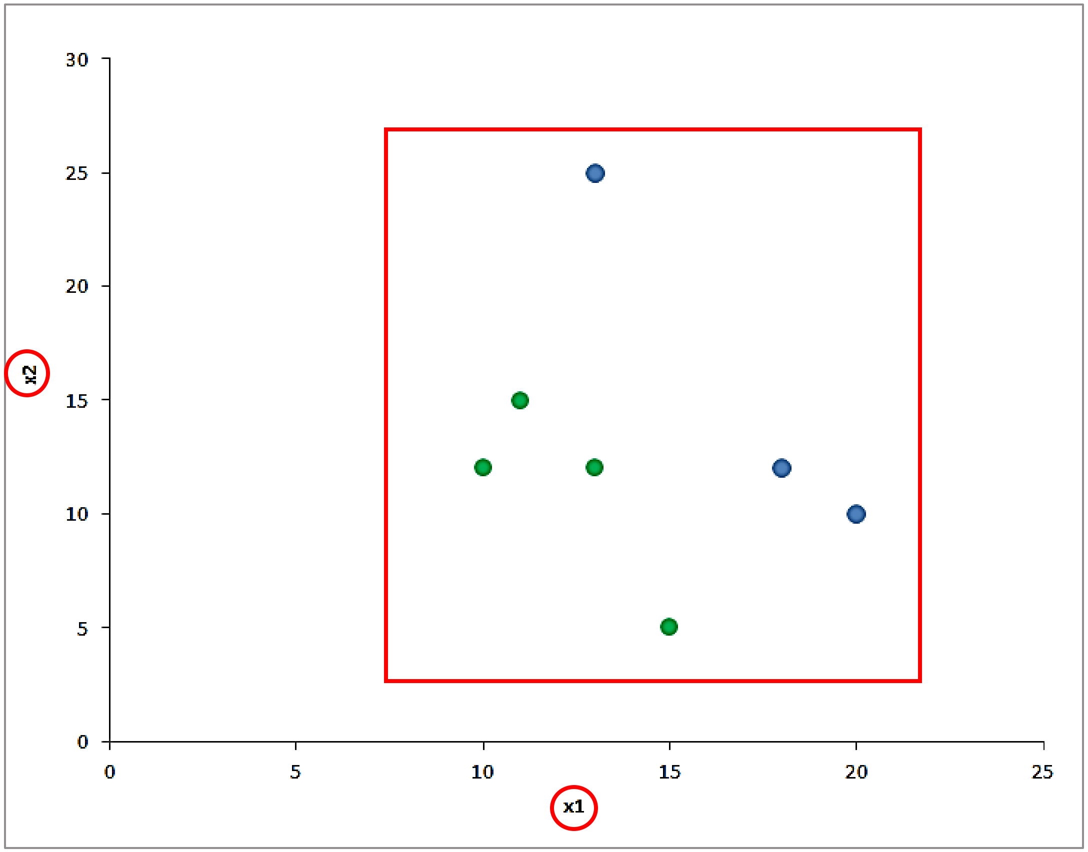


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

Observation on 7 data points

K-Nearest Neighbor

Example

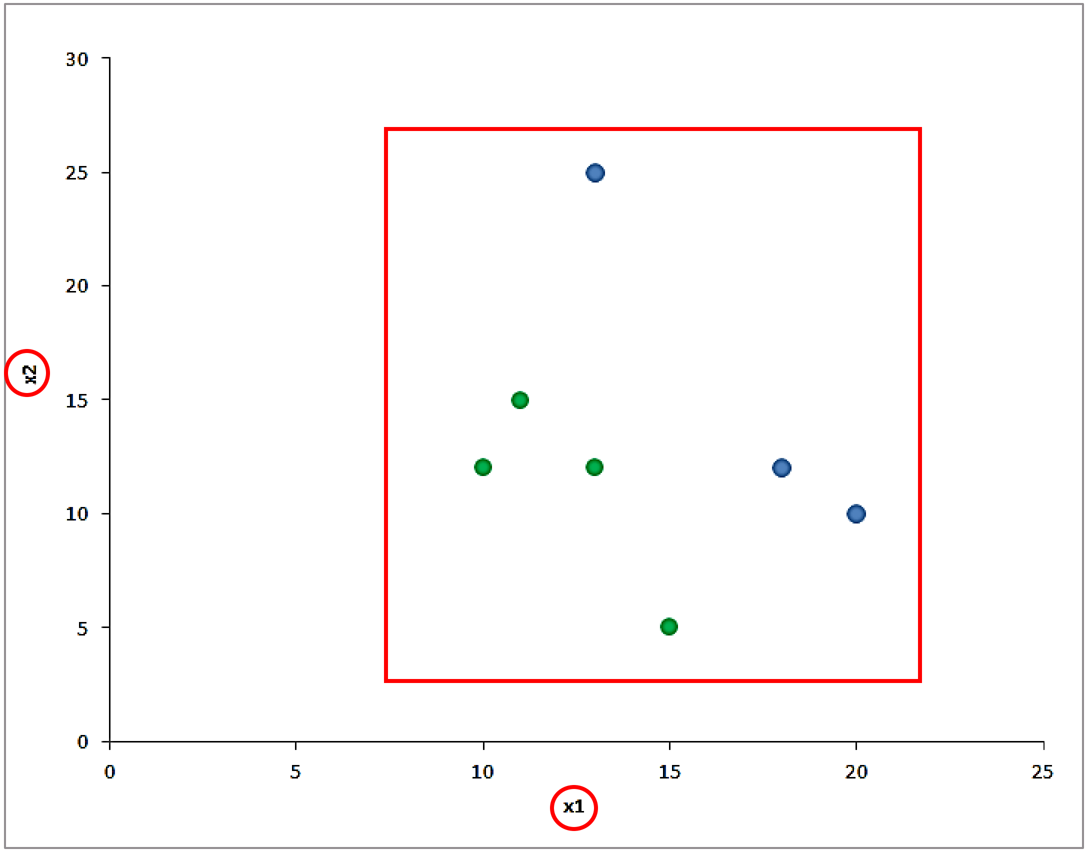


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

There are 2 features **x1** and **x2** for each of these 7 points

K-Nearest Neighbor

Example

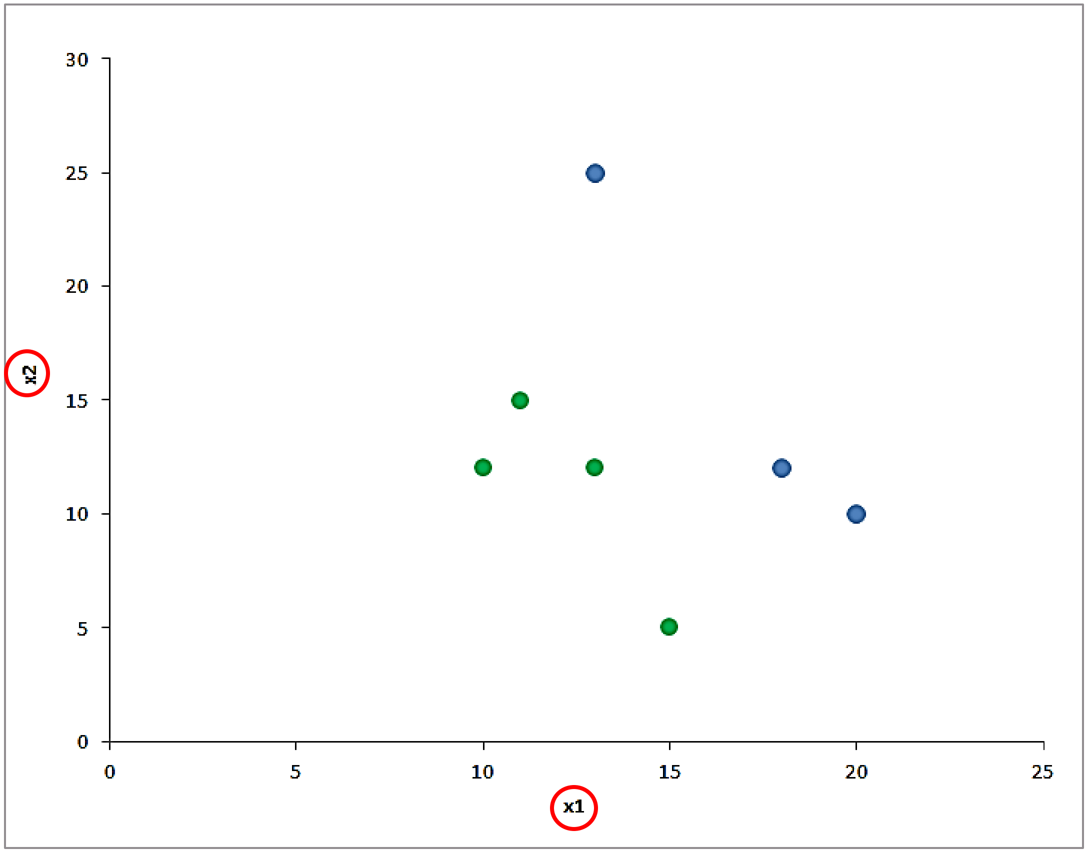


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

Response is a binary variable – takes one of green or blue

K-Nearest Neighbor

Example

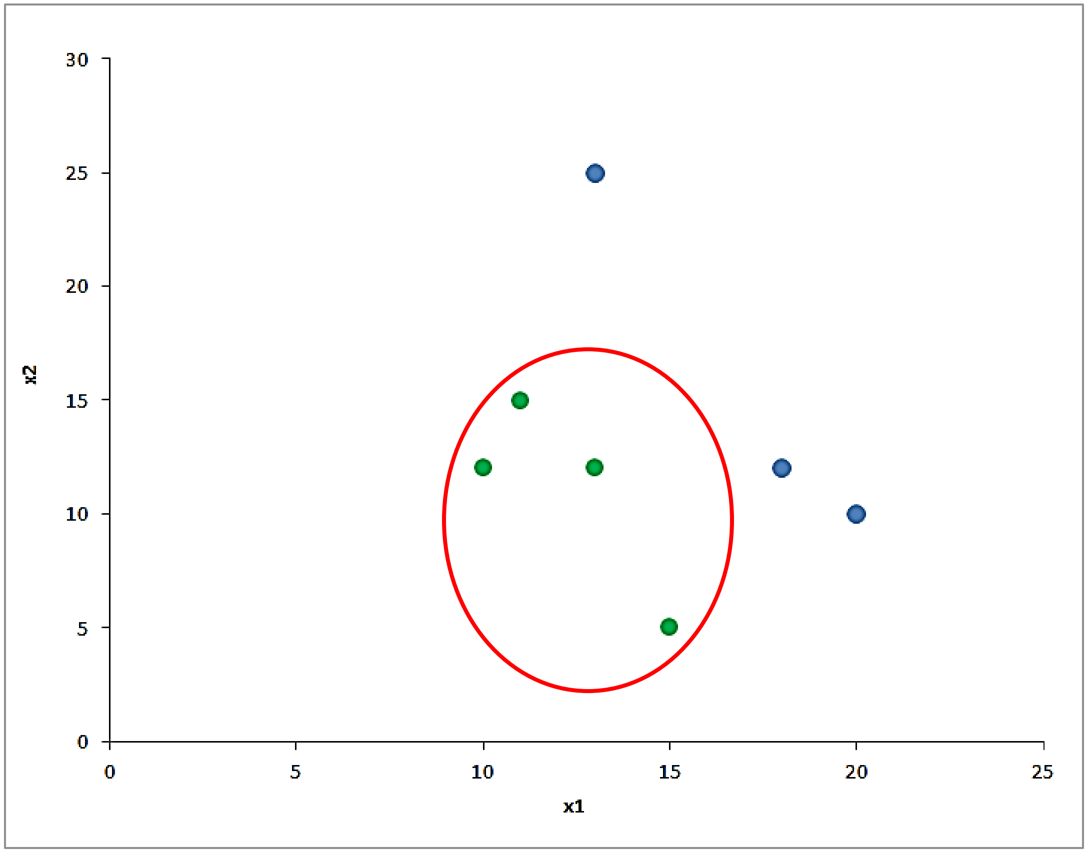


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

A scatterplot of the given data points can be plotted with **x1** on the **x** axis and **x2** on the **y** axis

K-Nearest Neighbor

Example

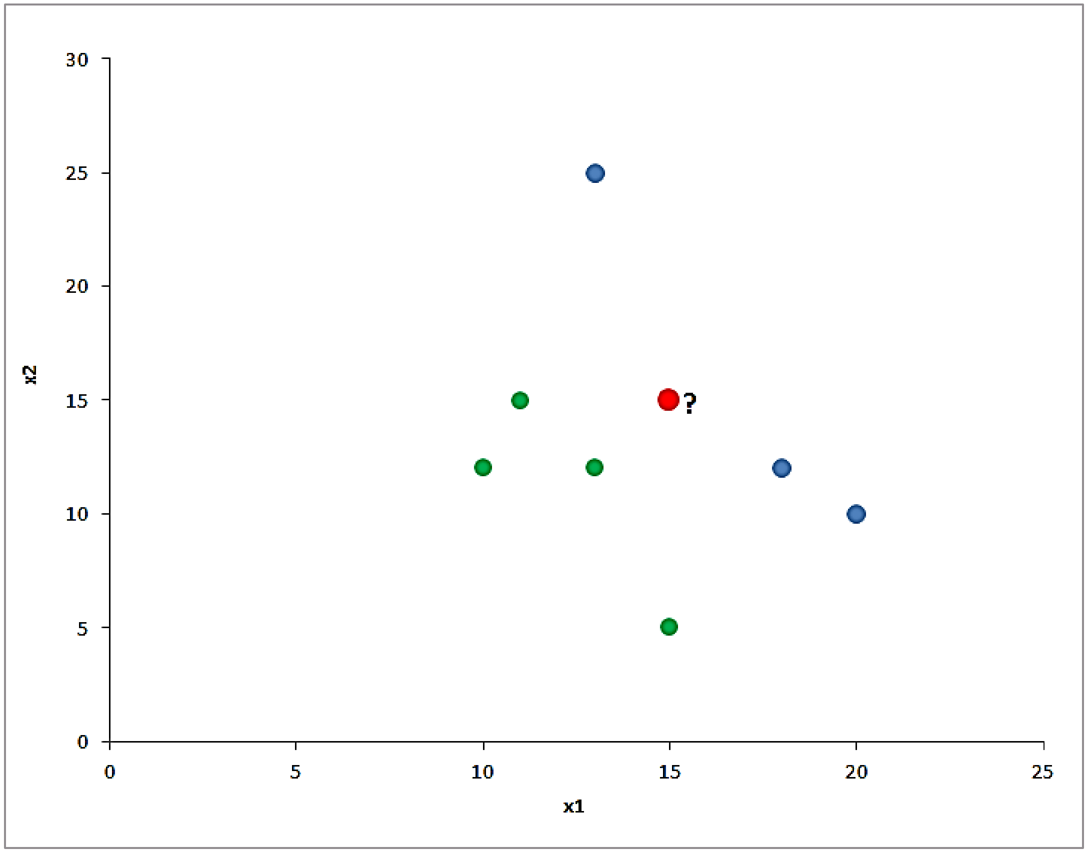


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

Each point is colored according to the color in the response; if the response is green in color, the point is green

K-Nearest Neighbor

Example

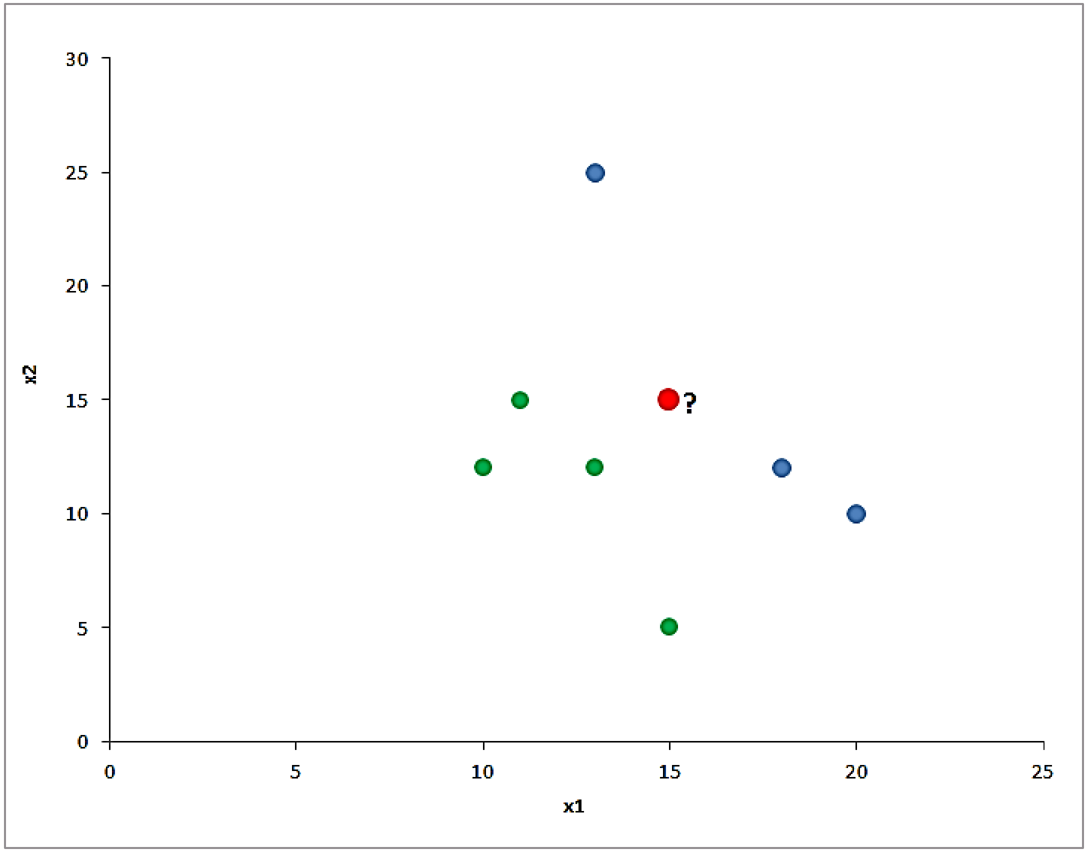


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

Given this scatterplot, what if you are given a new red point whose color is unknown?

K-Nearest Neighbor

Example

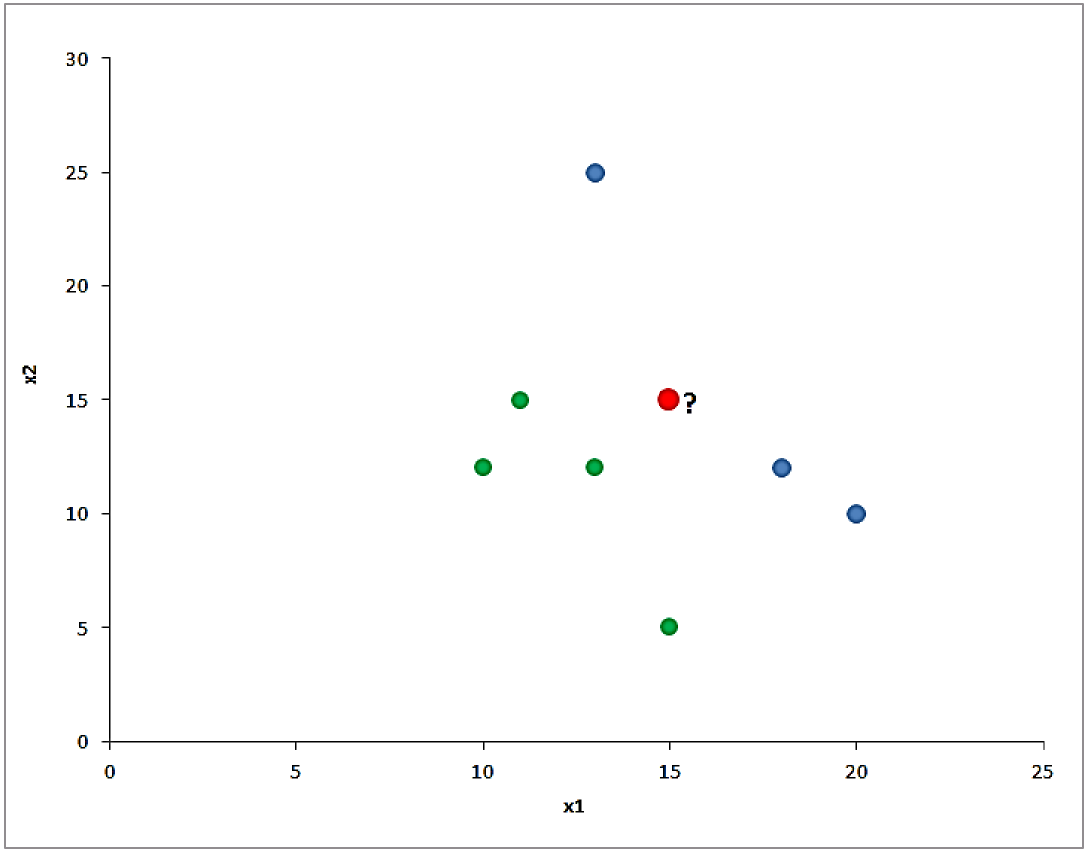


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

Which color would you like to assign to the red point? Green or Blue?

K-Nearest Neighbor

Example

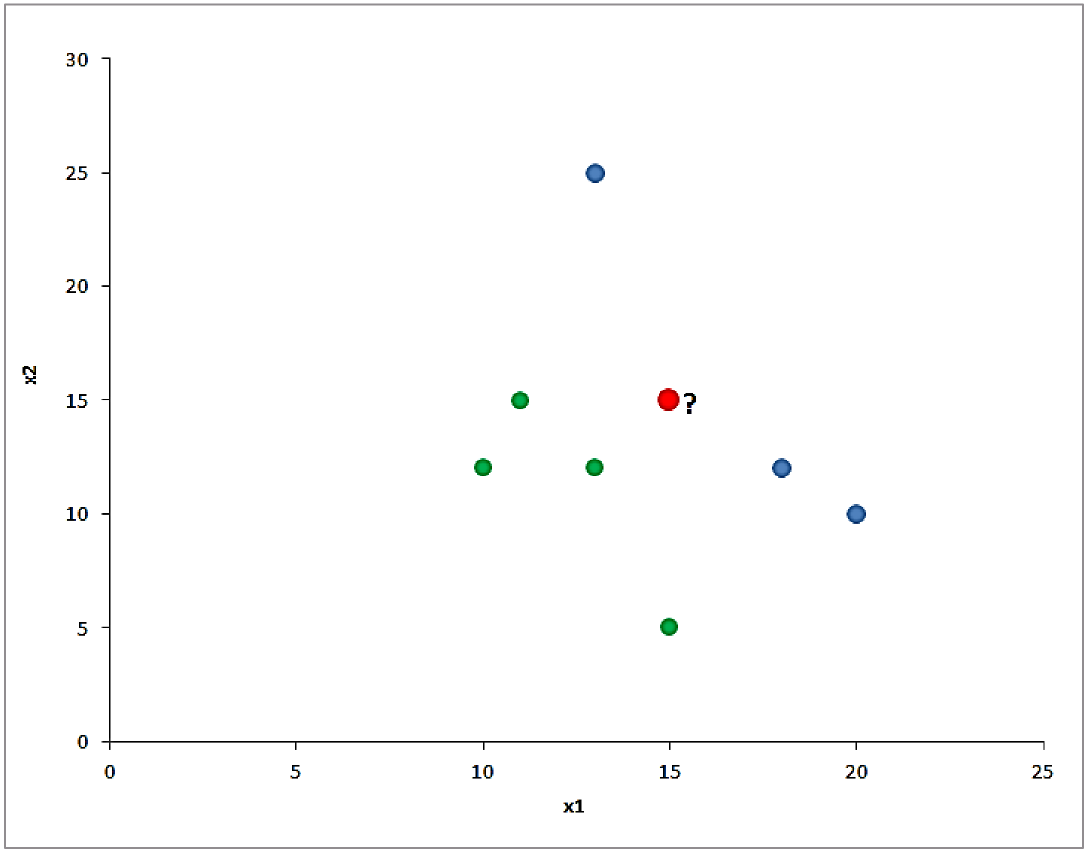


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

With a linear classifier on this data set, the line that best separates the green dots from the blue ones needs to be determined

K-Nearest Neighbor

Example

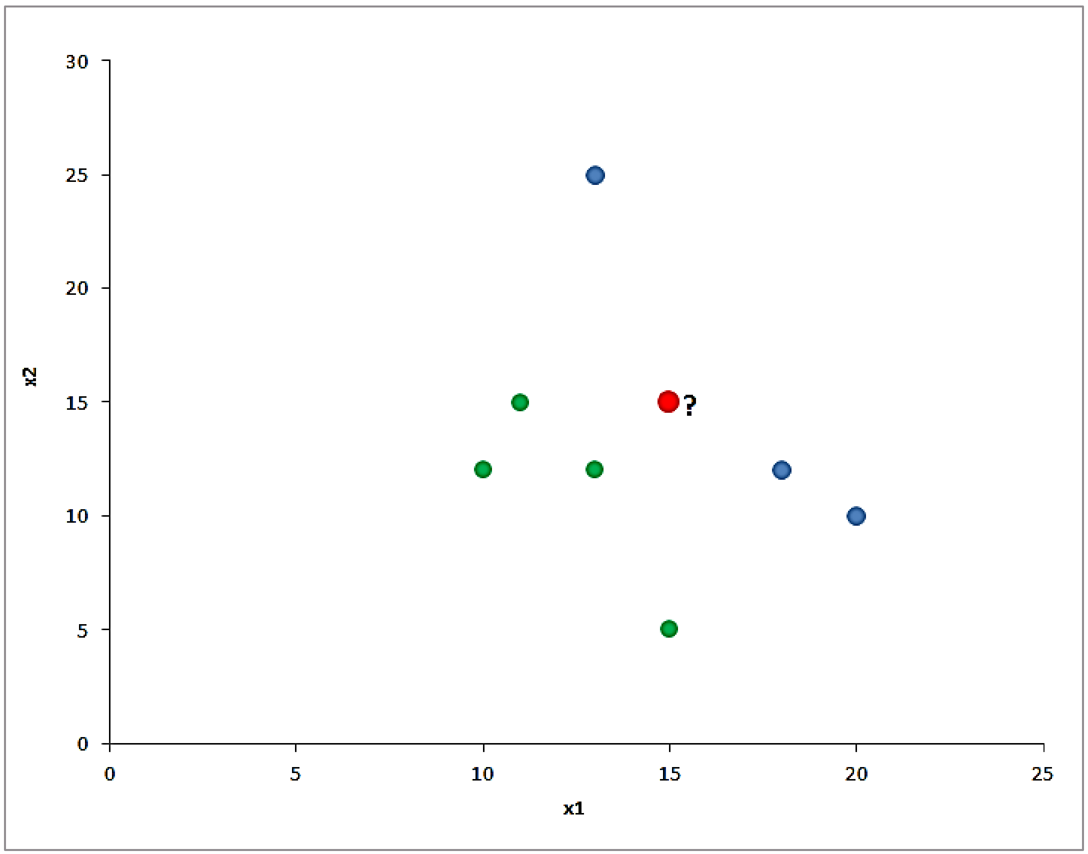


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

If the red point falls on the right side of the line, it is marked **blue** and if it falls on the left side, it is marked as **green**

K-Nearest Neighbor

Example

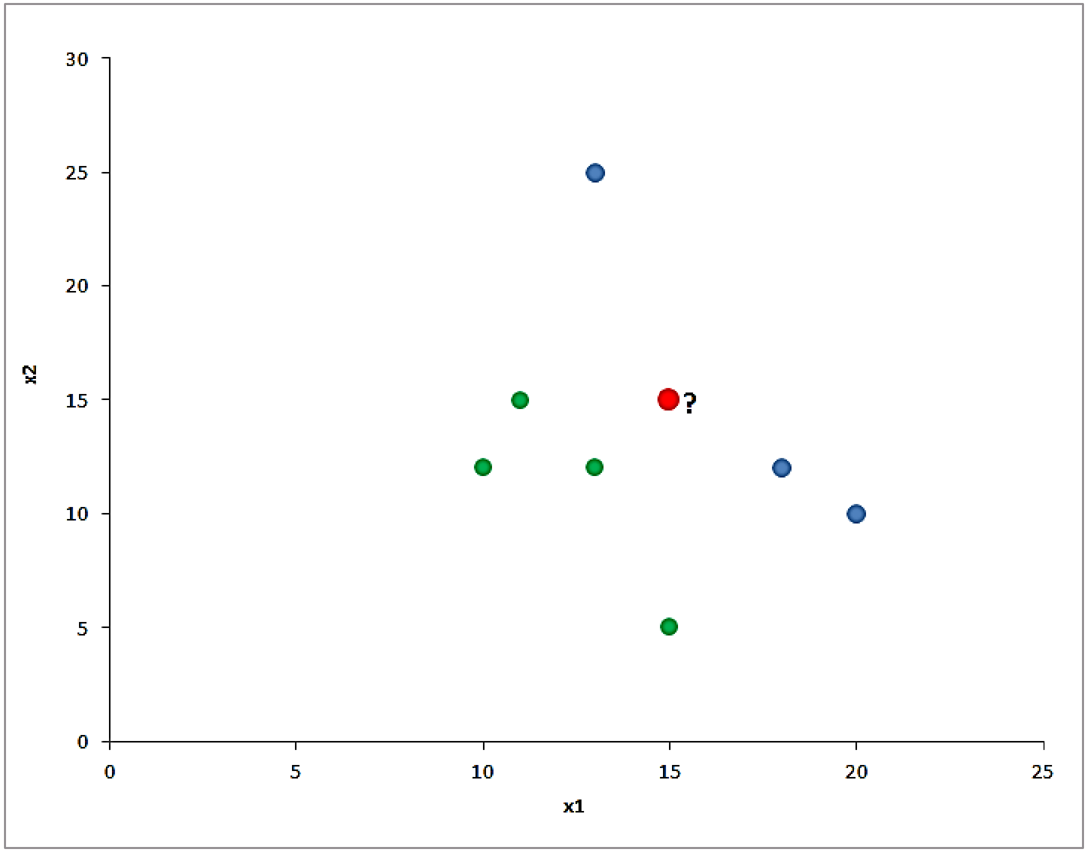


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

There might be cases where perfect separation using a straight line or a **decision plane**, is not possible

K-Nearest Neighbor

Example

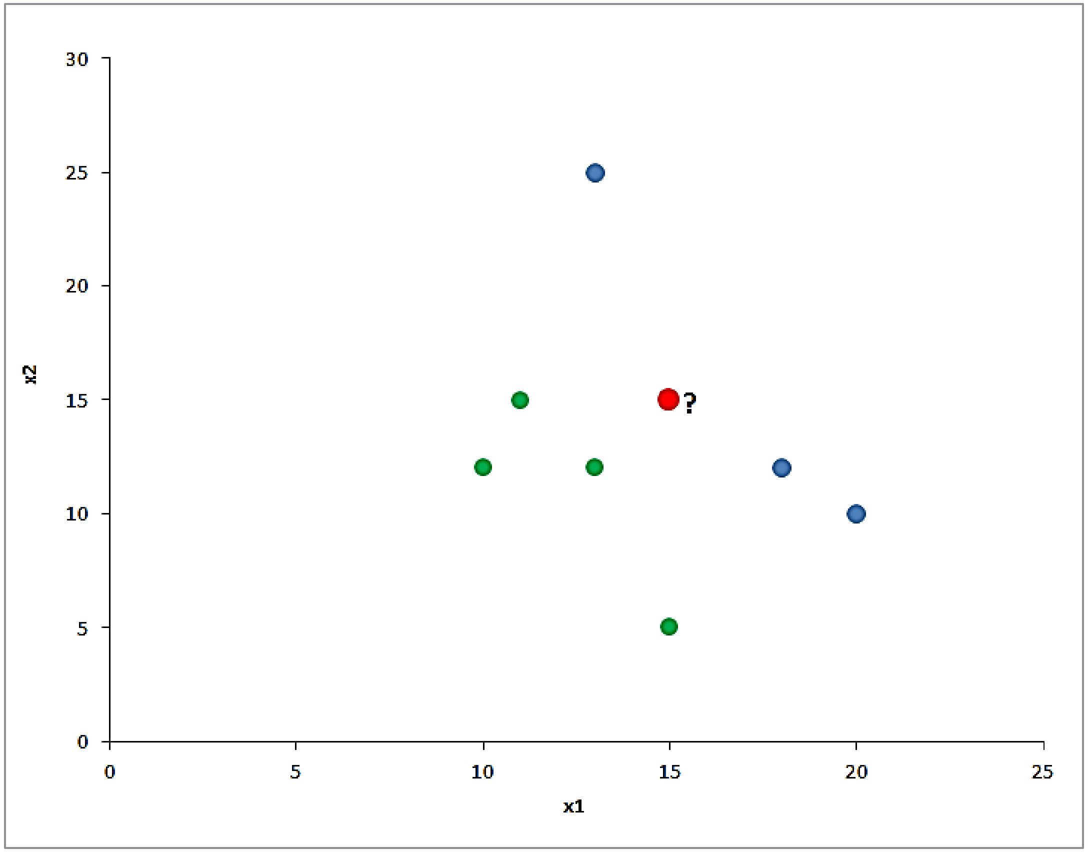


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

All the methods still work, but all possible decision planes make mistakes

K-Nearest Neighbor

Example



id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

The line that makes the minimum mistakes needs to be selected

K-Nearest Neighbor

Distances

They play an important role in unsupervised learning methods

Distances are an important part of many machine learning algorithms

Helps figure out similar observations in the data, identify outlying observations etc.

Helps the computer in figuring out how far apart 2 data points are in a very high dimensional setting

K-Nearest Neighbor

Distances

In the case of 2-dimensional setting, how far apart are the points **0,0** and **1,2** on a scatterplot?

Do you agree that **0,0** and **1,2** are much closer to each other than **0,0** and **100,-100**?

How would you quantify this?

K-Nearest Neighbor

Distances

In mathematical terms, the 2-dimensional space charted out in a scatterplot is also called the **Euclidean** plane

Euclidean distance is a geometric concept of distance between 2 points on the Euclidean plane

In a 2-dimensional plane, there are 2 points – **(x1, y1)** and **(x2, y2)**

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Take the difference of the **x** and **y** values separately, square each of them and add up

K-Nearest Neighbor

Distances

In mathematical terms, the 2-dimensional space charted out in a scatterplot is also called the **Euclidean** plane

Euclidean distance is a geometric concept of distance between 2 points on the Euclidean plane

In a 2-dimensional plane, there are 2 points – **(x1, y1)** and **(x2, y2)**

$$D = \sqrt{\underline{(x_1 - x_2)^2 + (y_1 - y_2)^2}}$$

This gives the squared distance

K-Nearest Neighbor

Distances

In mathematical terms, the 2-dimensional space charted out in a scatterplot is also called the **Euclidean** plane

Euclidean distance is a geometric concept of distance between 2 points on the Euclidean plane

In a 2-dimensional plane, there are 2 points – **(x1, y1)** and **(x2, y2)**

$$D = \boxed{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}$$

Take the square root of the squared distance to obtain the Euclidean distance between the 2 points

K-Nearest Neighbor

Distances

In mathematical terms, the 2-dimensional space charted out in a scatterplot is also called the **Euclidean** plane

Euclidean distance is a geometric concept of distance between 2 points on the Euclidean plane

In a 2-dimensional plane, there are 2 points – **(x1, y1)** and **(x2, y2)**

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Example: Distance = $\sqrt{(1-3)^2 + (2-1)^2} = \sqrt{5} \sim \underline{2.23}$

K-Nearest Neighbor

Distances

In mathematical terms, the 2-dimensional space charted out in a scatterplot is also called the **Euclidean** plane

Euclidean distance is a geometric concept of distance between 2 points on the Euclidean plane

In a 2-dimensional plane, there are 2 points – **(x1, y1)** and **(x2, y2)**

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Euclidean distances are not limited to 2-dimensional settings, they can generalize to cases where the number of dimensions is more than 2

K-Nearest Neighbor

Distances

Distance measures are not limited to Euclidean distances

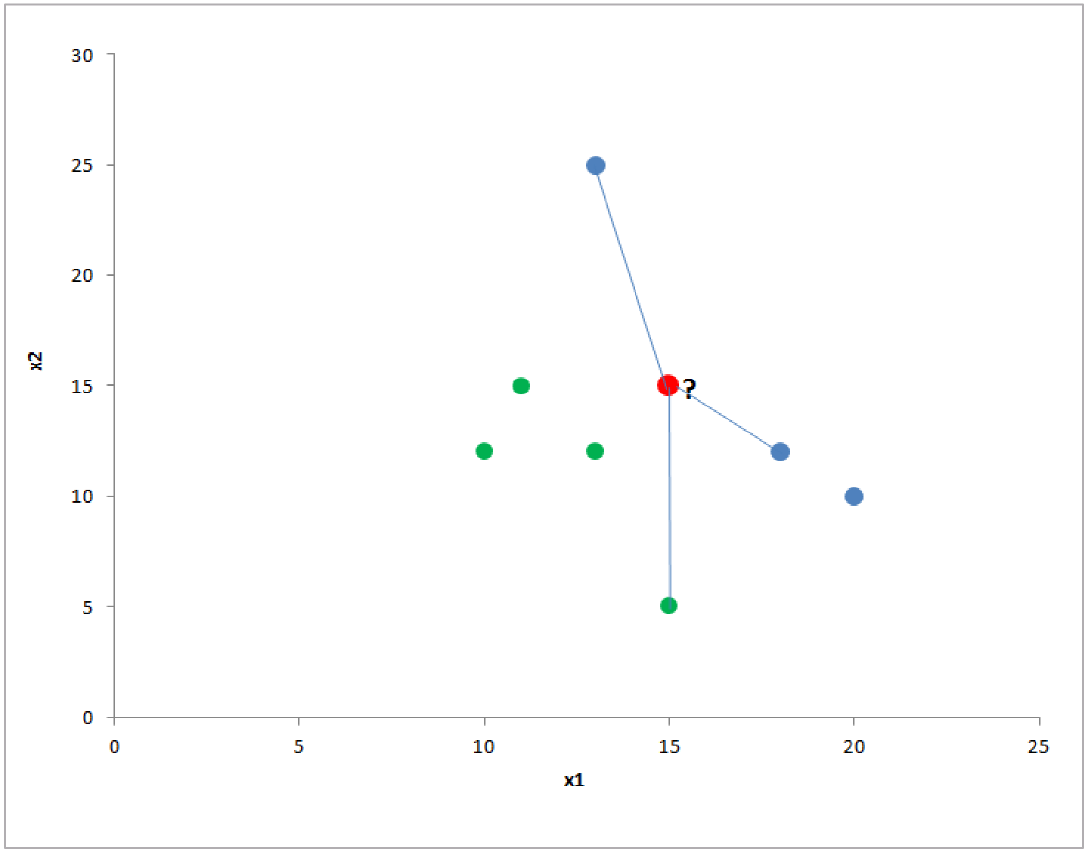
There are many different distance measures defined on the Euclidean plane

Example: Manhattan distance, Minkowski distance, etc.

Euclidean distances work only on the Euclidean plane; more complicated measures are required for other settings

K-Nearest Neighbor

Distances

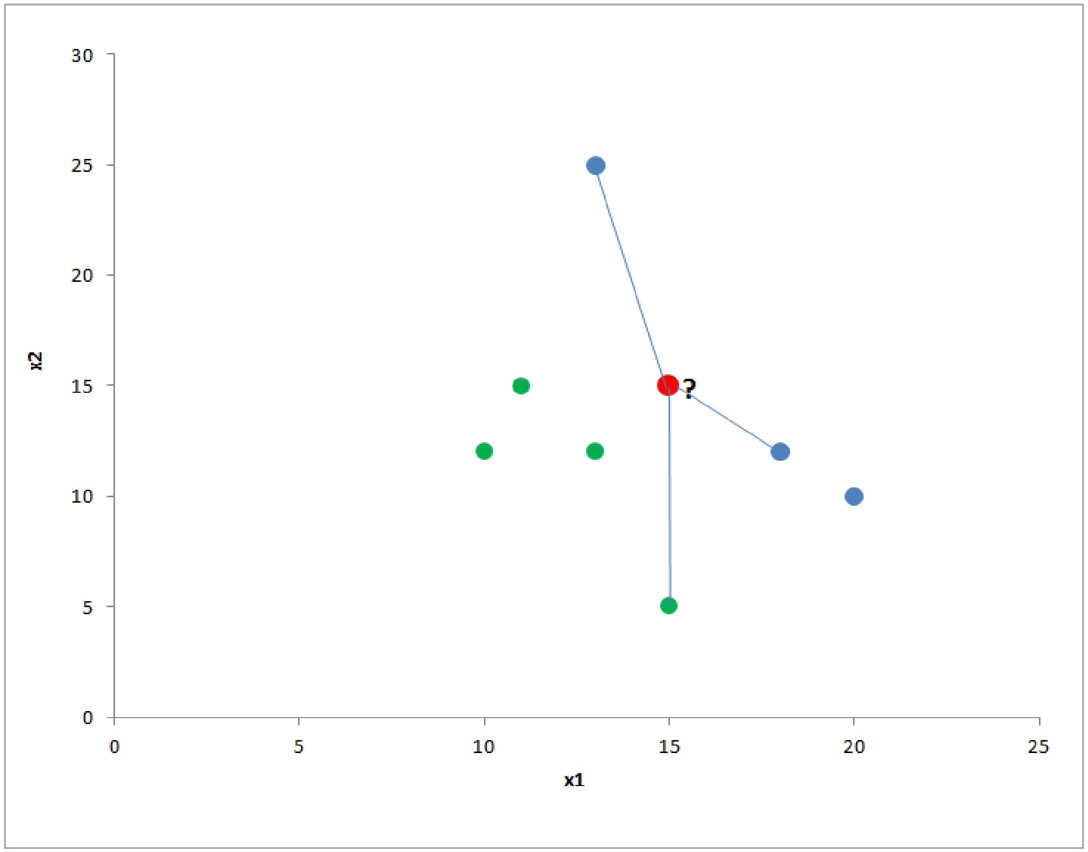


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

For the new red point, look at the nearest neighbors of this red point in the existing dataset

K-Nearest Neighbor

Distances



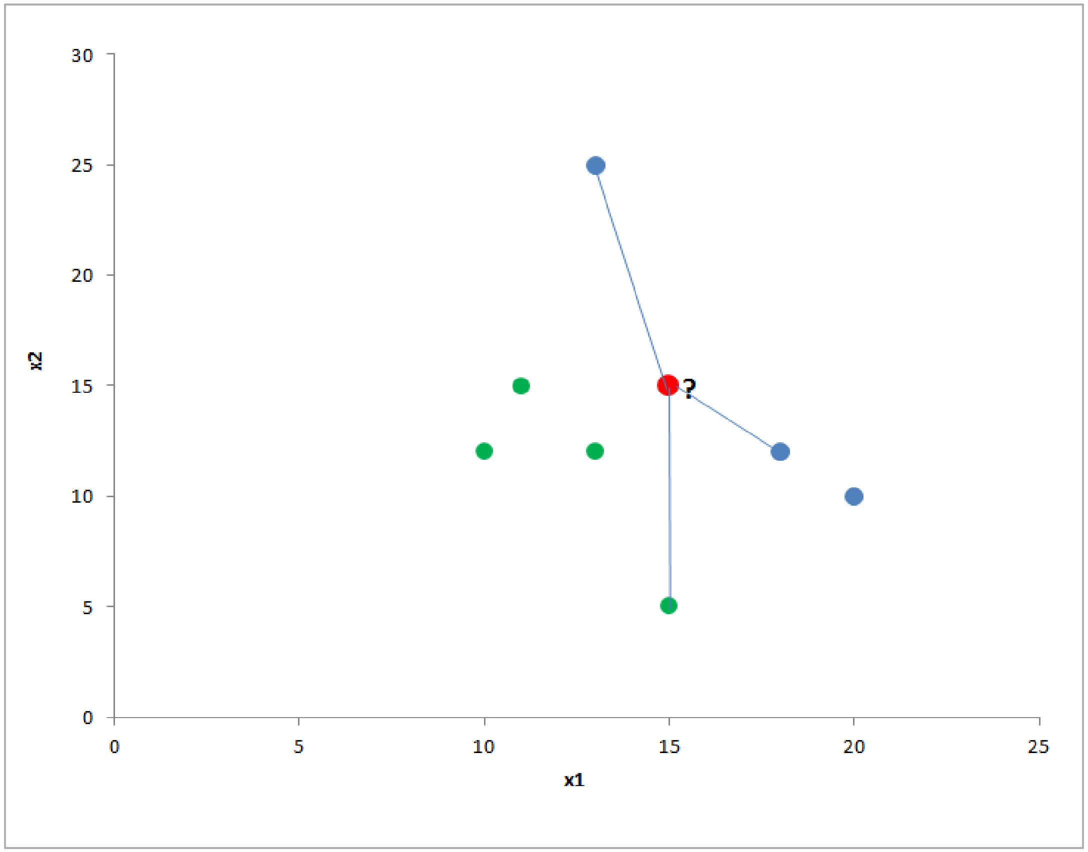
id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

How do you compute whether a point is near a red point or not?

Calculate the distance between the points – Euclidean distance

K-Nearest Neighbor

Distances

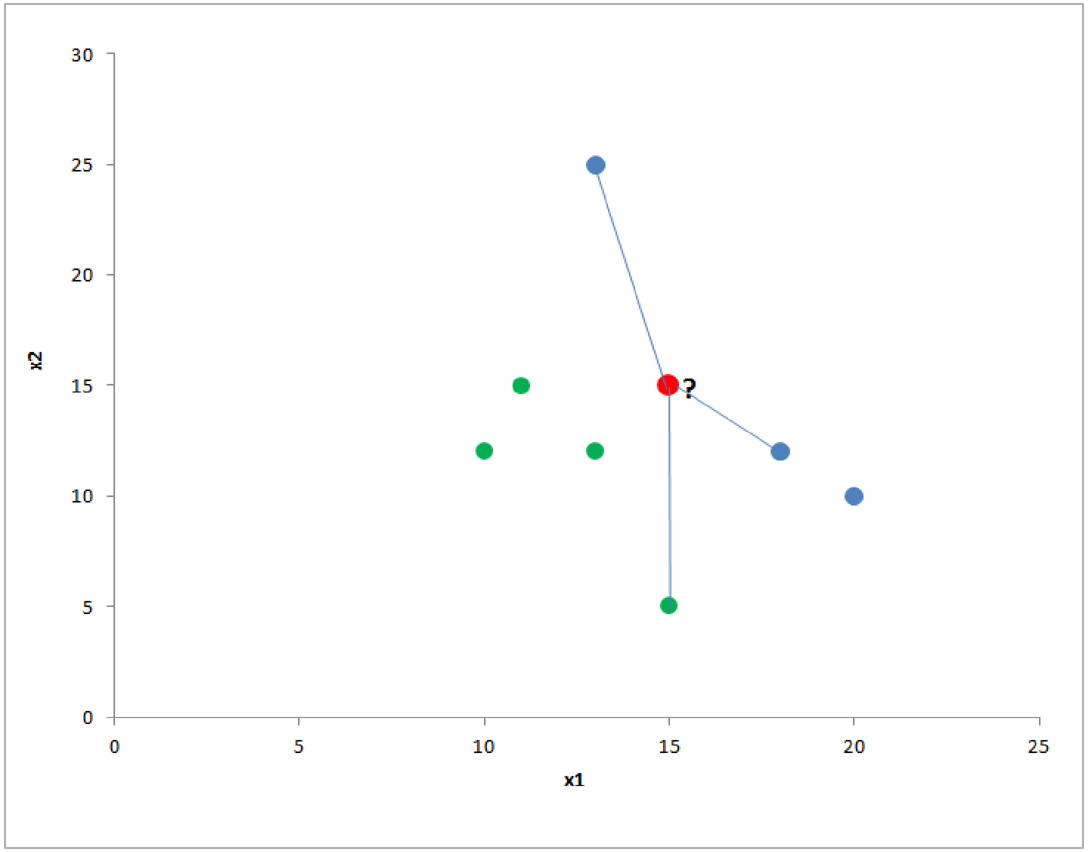


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

From the red point, calculate the Euclidean distances to each of the 7 points in the given data

K-Nearest Neighbor

Distances

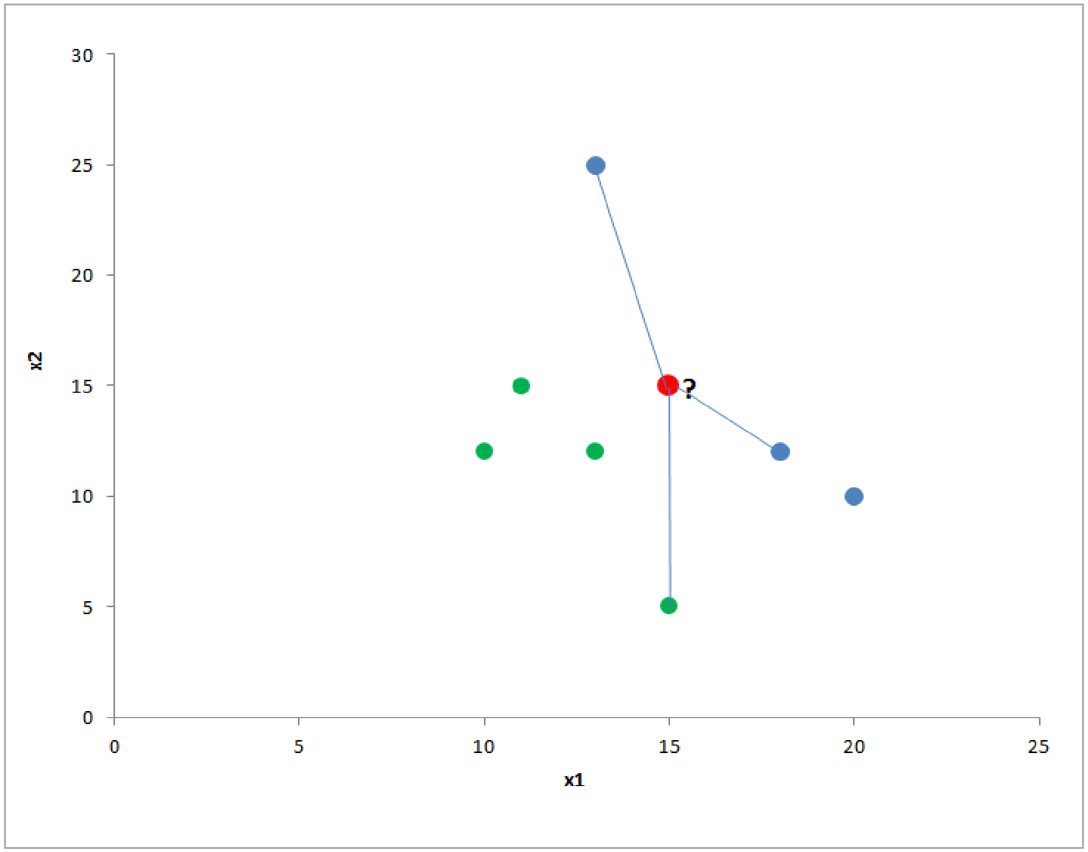


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

Arrange the data points in ascending order of distances, i.e., the lowest distance first

K-Nearest Neighbor

Distances

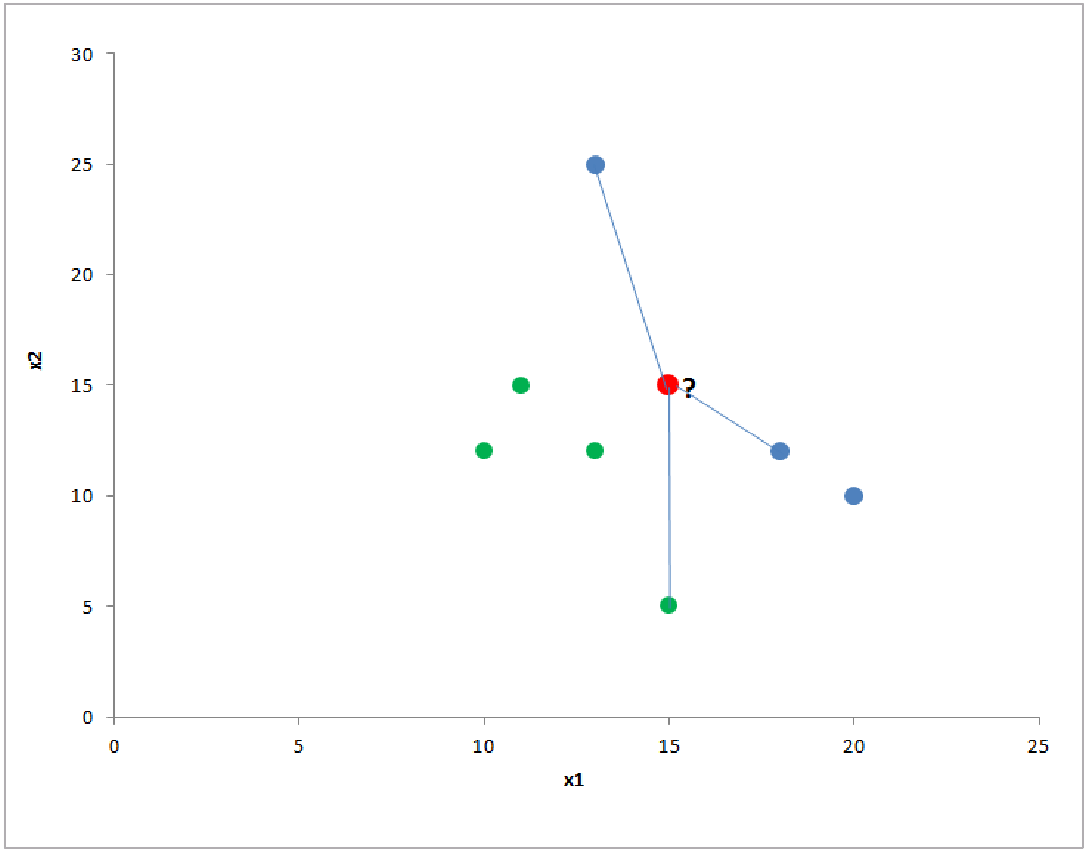


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

Take the first K points in this ordered list

K-Nearest Neighbor

Distances

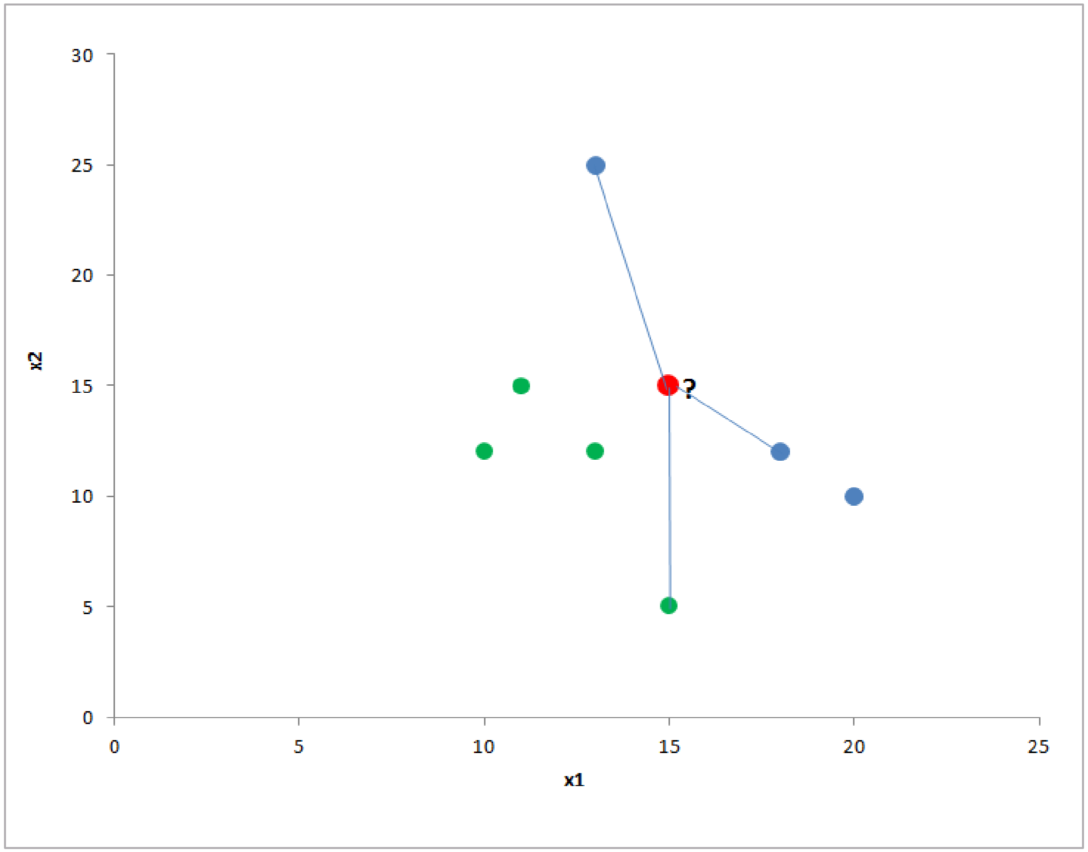


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

Look at the responses of these K points and take a majority vote

K-Nearest Neighbor

Distances

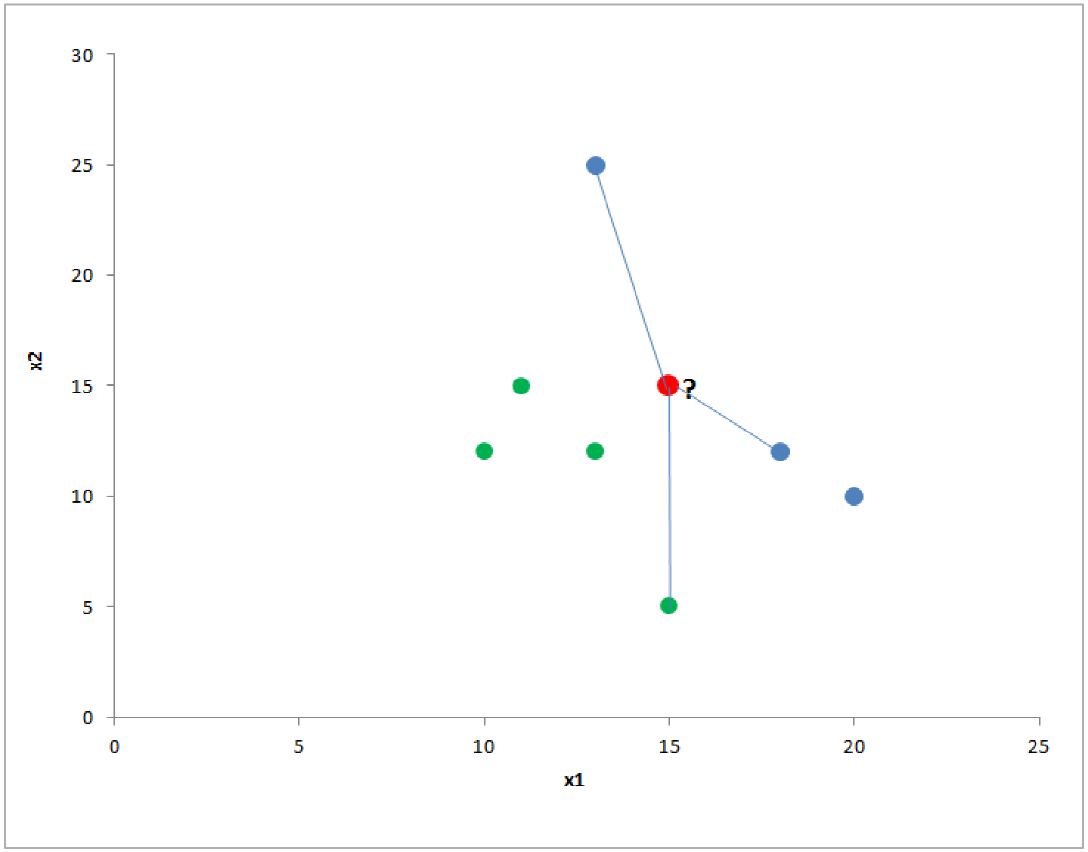


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

Example: K is 3 and after ordering the data in terms of closest first, the first 3 responses are green, green and red

K-Nearest Neighbor

Distances

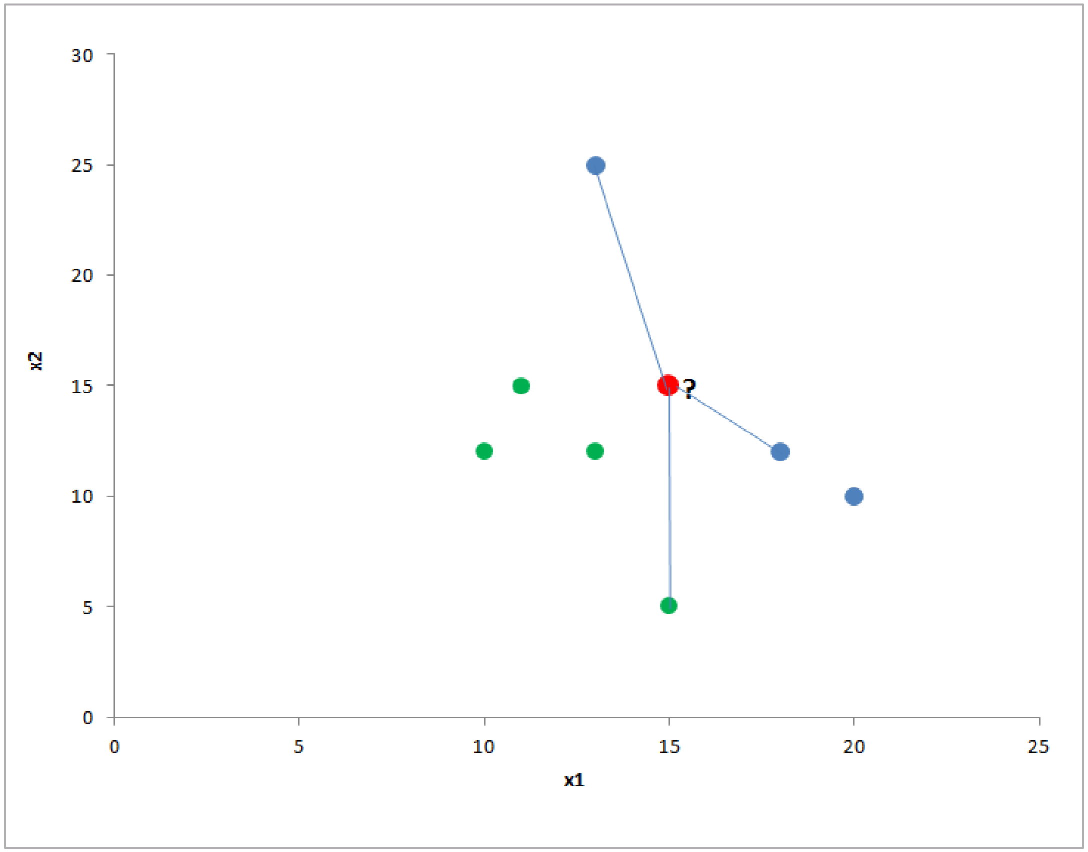


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

The majority vote in this case is **green**

K-Nearest Neighbor

Distances

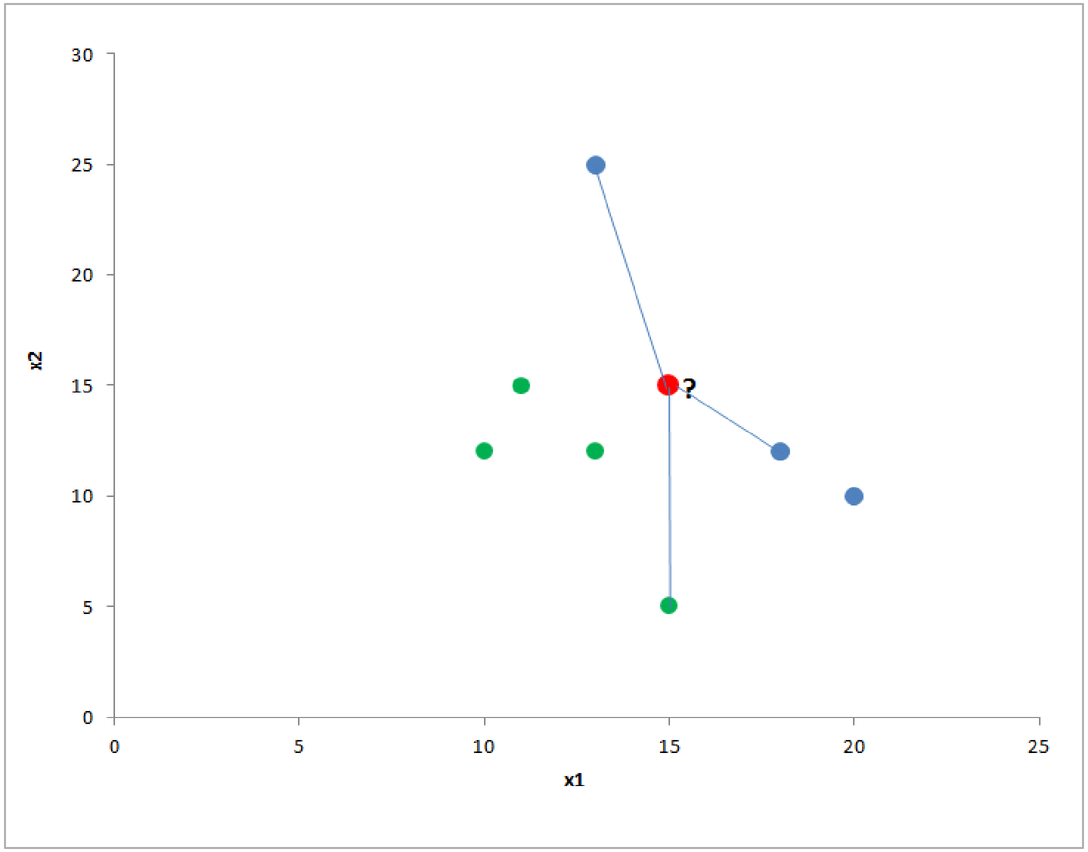


id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

Go ahead and mark the color of the red point as **green**

K-Nearest Neighbor

Exercise



id	x1	x2	y
1	10	12	green
2	11	15	green
3	15	5	green
4	25	5	green
5	15	44	blue
6	13	12	green
7	13	25	blue

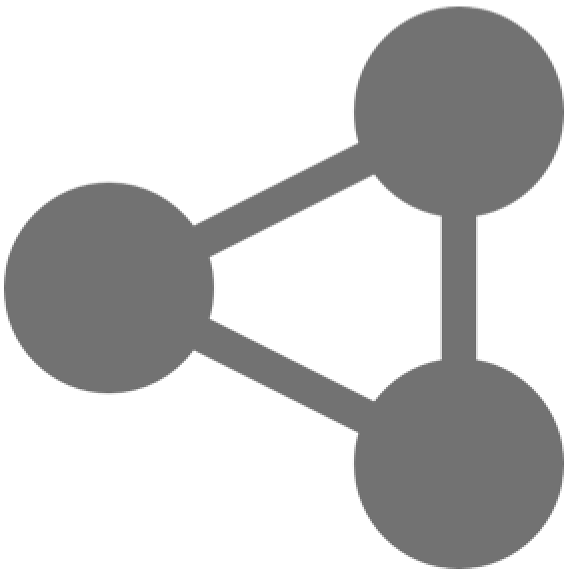
Complete this problem assuming **K=3**

K-Nearest Neighbor

What is K?

How do you do K nearest neighbor if K's value is unknown?

K-Nearest Neighbor



Many machine learning algorithms are dependent on parameters like these which can not be estimated directly from the data

These are typically called **tuning parameters** or **hyperparameters**

All the key results for these algorithms are based on a fixed value of that parameter

Tuning Parameters

To build a good model, it is very important to find out a good value for a tuning parameter

An algorithm can have one or multiple tuning parameters

Hyperparameter optimization is a very big sub-discipline in machine learning

Tuning Parameters

In this course, we will only take a look at one of the many ways of hyperparameter optimization

Tuning Parameters

To select a tuning parameter for the K-Nearest Neighbor, define a candidate list of values – 3, 5 and 7 are some popular choices

This is one of the many cases where splitting the data into **train**, **test** and **validate** come in really handy

For each value of K, run the nearest neighbor, calculate the error on the testing split

Select that K for which the misclassification rate is the lowest

A search through a manually specified sub-set of the hyperparameter space of a learning algorithm

Grid Search

Tuning Parameters

To select a tuning parameter for the K-Nearest Neighbor, define a candidate list of values – 3, 5 and 7 are some popular choices

This is one of the many cases where splitting the data into **train**, **test** and **validate** come in really handy

For each value of K, run the nearest neighbor, calculate the error on the testing split

Select that K for which the misclassification rate is the lowest

Randomized Search

Gradient Based Optimization

Bayesian Optimization

End-to-End Workflow

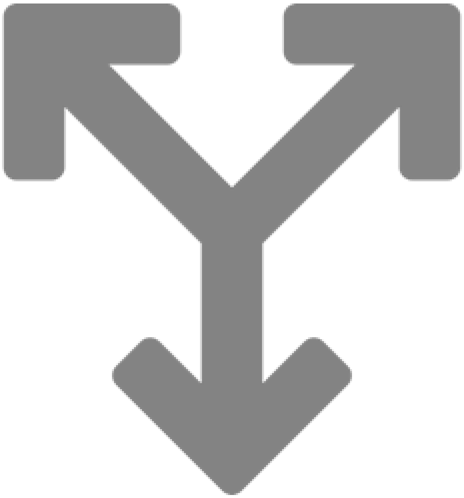
①

Read data into memory

End-to-End Workflow

②

Split the data randomly into **train**, **test**, and **validate** – ensure that the **train** split gets the majority of the data



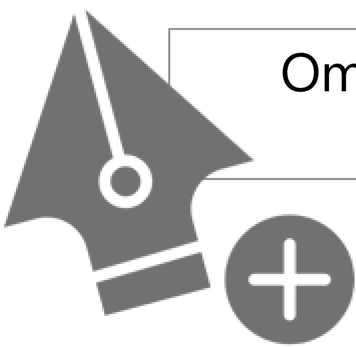
End-to-End Workflow

③

Perform extensive exploration on the training datasets

End-to-End Workflow

④



Omit, add, and create new features in the training set

End-to-End Workflow

⑤

Prepare a list of candidate models

These can be different models/same models with multiple tuning parameters for the purposes of **Grid Search**

Remember that 3-NN is a different model from a 5-NN although they come from the same family of models

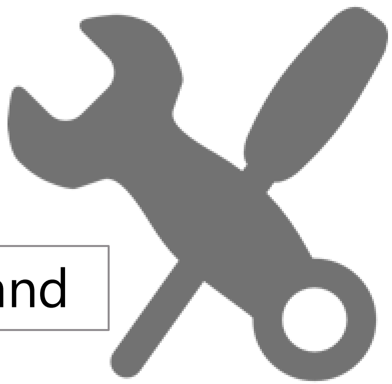
End-to-End Workflow

⑥

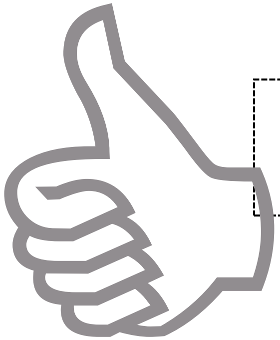
After training, evaluate each of these models on the testing split

End-to-End Workflow

⑦



Fix a performance measure beforehand



It is recommended to not alter the performance measure choice after seeing the results on the testing split

End-to-End Workflow

⑧

Select the model that performs best on the testing split and report it as the final model

End-to-End Workflow

⑨

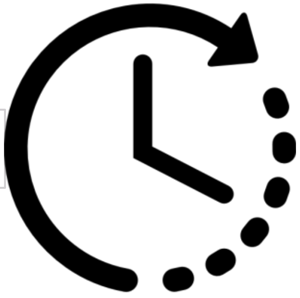
Calculate another round of performance measure for this using the **validate** split – it is the proxy for how well the model will perform when deployed over new and unseen data points

If this performance on **validate** is not up to the mark or decays very fast, all or some of the processes will have to be repeated

End-to-End Workflow

Most professionals in this field rarely get a model right at the first go

This can often be a long and tedious process



Recap

K-Nearest Neighbor and Summing Up the End-to-End Workflow

K-Nearest Neighbor

Tuning Parameters

End-to-End Workflow

Next

Practical Applications of Machine Learning