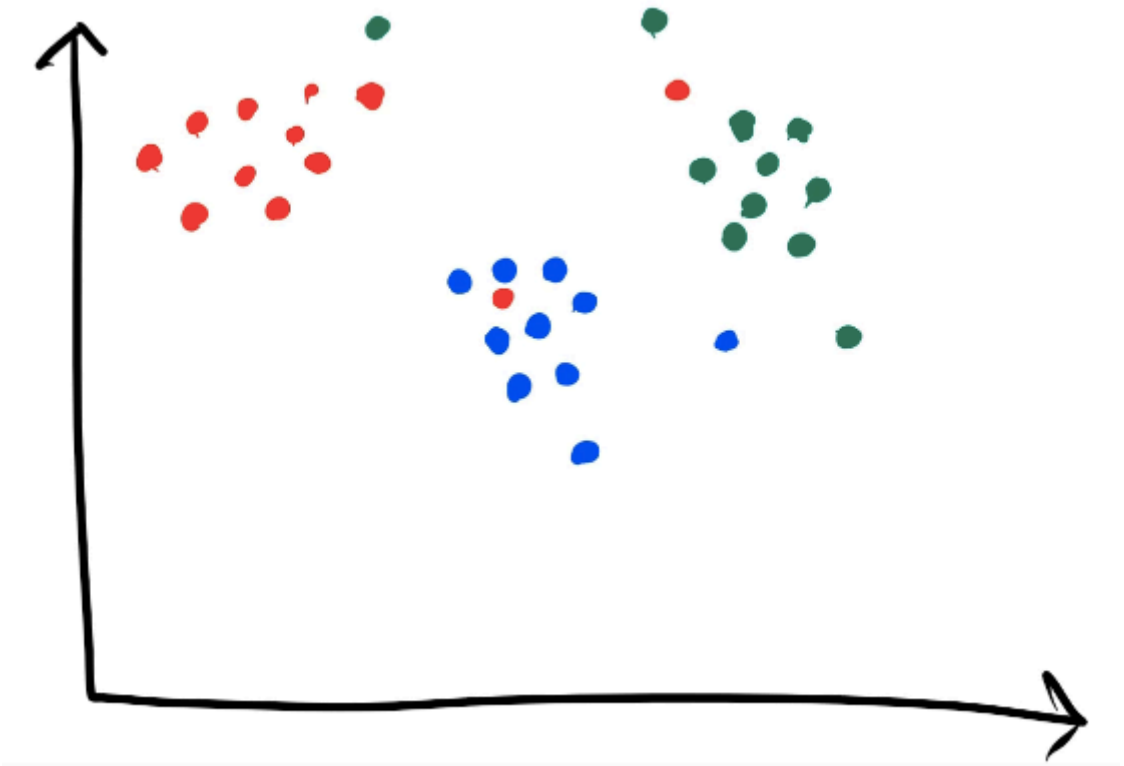
4) KNN

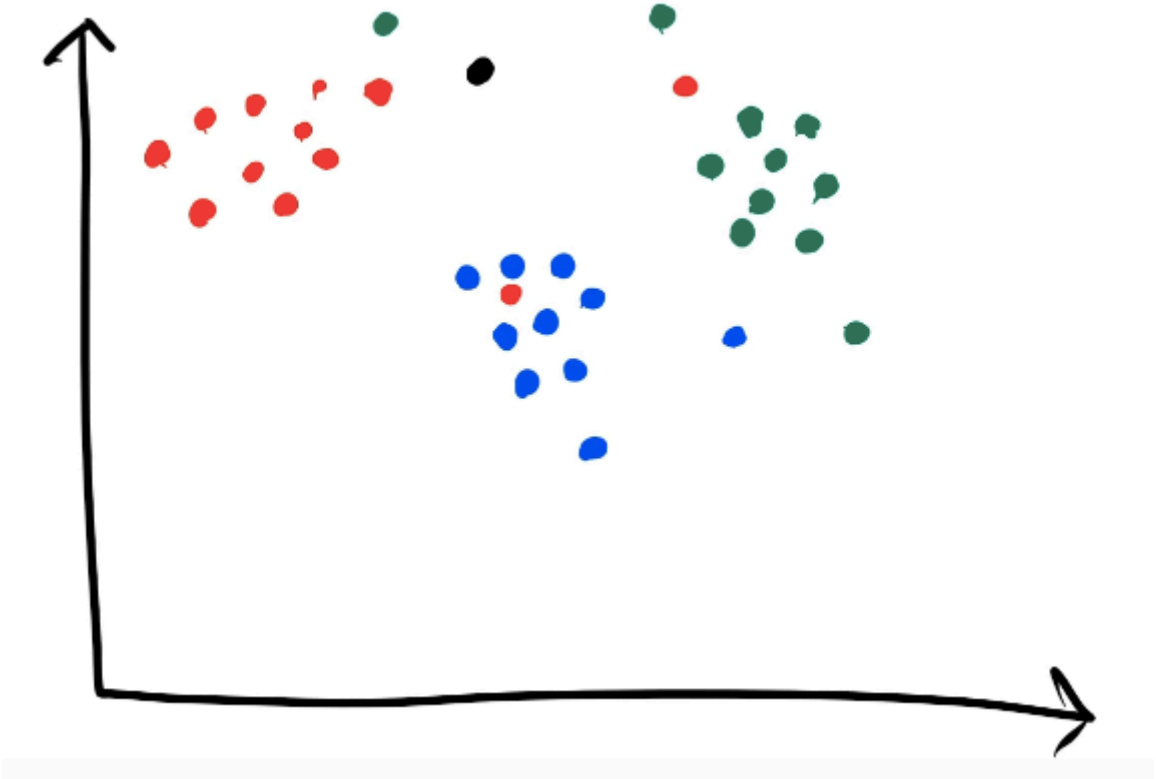
- Used to classify stuff
- It is very sensitive to unrelated features. Unlike regression, we cannot fit unrelated features into the model and then expect it to filter out them itself. We have to perform EDA and or use common sense to see which features are related
- Scaling is extremely important
- https://archive.ics.uci.edu/ml/datasets/Car+Evaluation
- **Pros:**
    - Simple to implement
    - Adapts well as new data is introduced
    - Easy to interpret (meaning it makes intuitive sense to the decision makers, ie it provides results or predictions in a way that is easily understandable)
- **Cons:**
    - Slow to predict, cause a LOT of distances have to be calculated
    - Can require a lot of memory
    - When there are many labels, KNN can start to break down cause of the curse of dimensionality

How it works

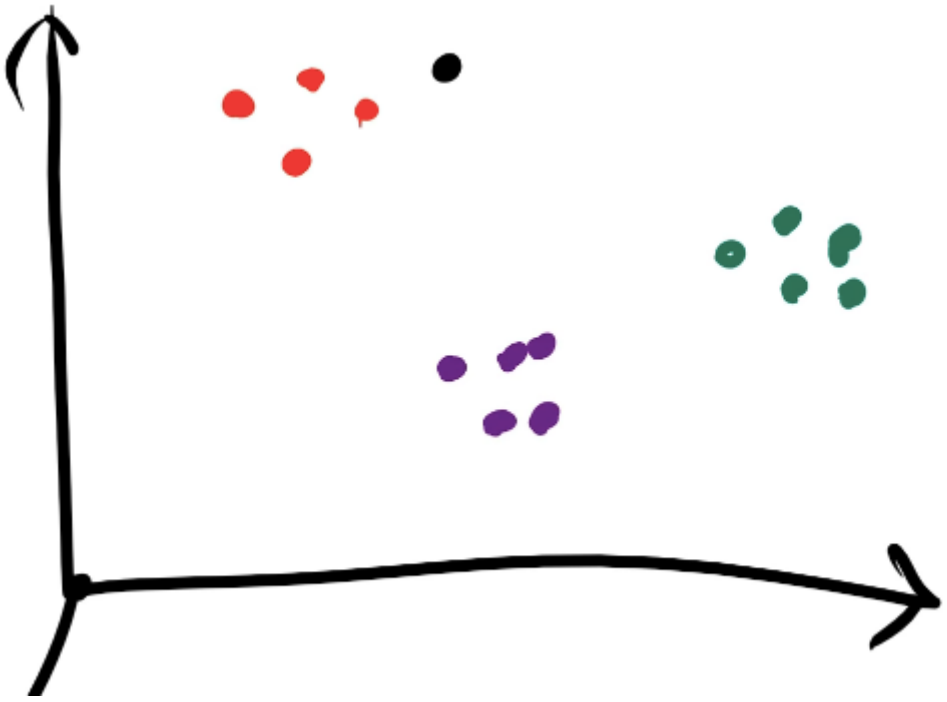
So we plant all the datapoints on a scatter plot, and each point is associated with a "class". Our aim is to determine which class a new point belongs to. We do that by finding the nearest points to that point. Suppose we have 3 classes (red, blue, and green), and various points are plotted like this:



and we need to find the class of this black point:



What we will do is calculate the distances between the black point and all the other points on the graph, select the nearest `k` (hyperparameter) number of points (ie points having the lowest distance), and checking which class most of those points have. We usually calculate the distance between two points using the sqrt($(\Delta x)^2 + (\Delta y)^2$) formula. `k` is usually set to be a small odd number. It is odd to avoid ties, and is small because in situations like these:



If we set `k` to 11, then the black (unclassified) point will not be classified as red, because red will never be the majority no matter how close the unclassified is to the red cluster

Distance/Similarity measurements

"Distance" between two points can be measured in various ways. The ones most popular in practice are:

Manhattan (L1)

The concept here is that "you cannot walk between two points directly, you have to turn at a corner first"

$|\Delta x| + |\Delta y|$

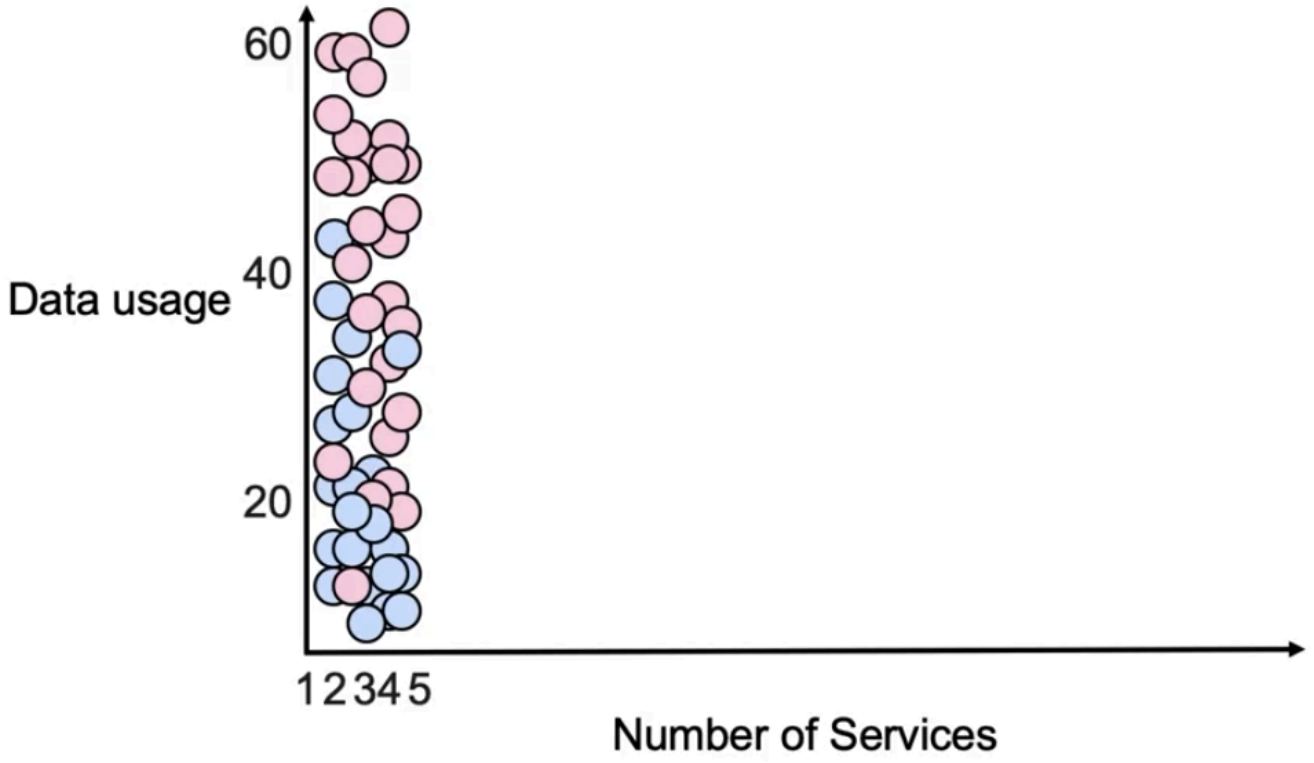
Euclidean (L2)

The most common and easy to understand method, basically how we were taught in school to calculate distance between two points. It works on the pythagorus theorem and calculates the literal distances between the points

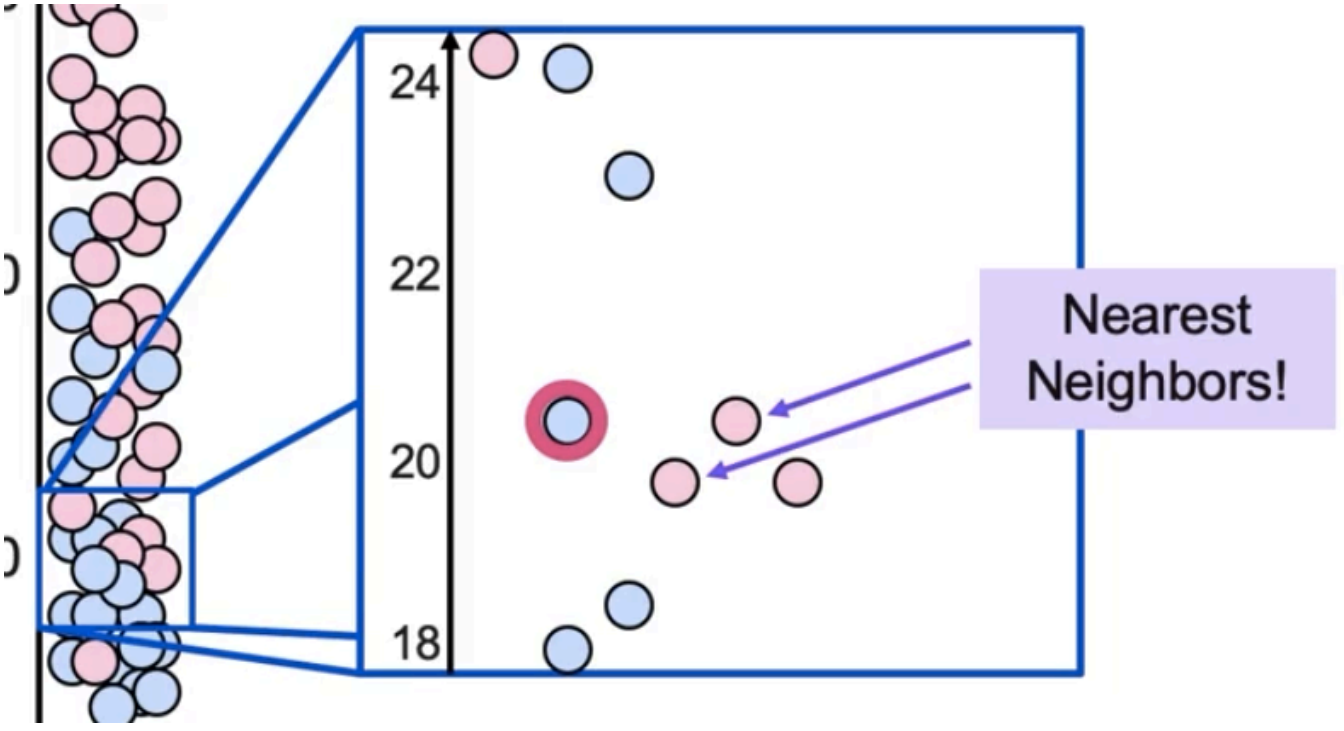
$sqrt(\Delta x^2 + \Delta y^2)$

Why scaling is important

Suppose we have two features (no. of services & data usage in GB), where no. of services ranges from 0-5 and the data usage ranges from 0-60. The graph looks something like this:

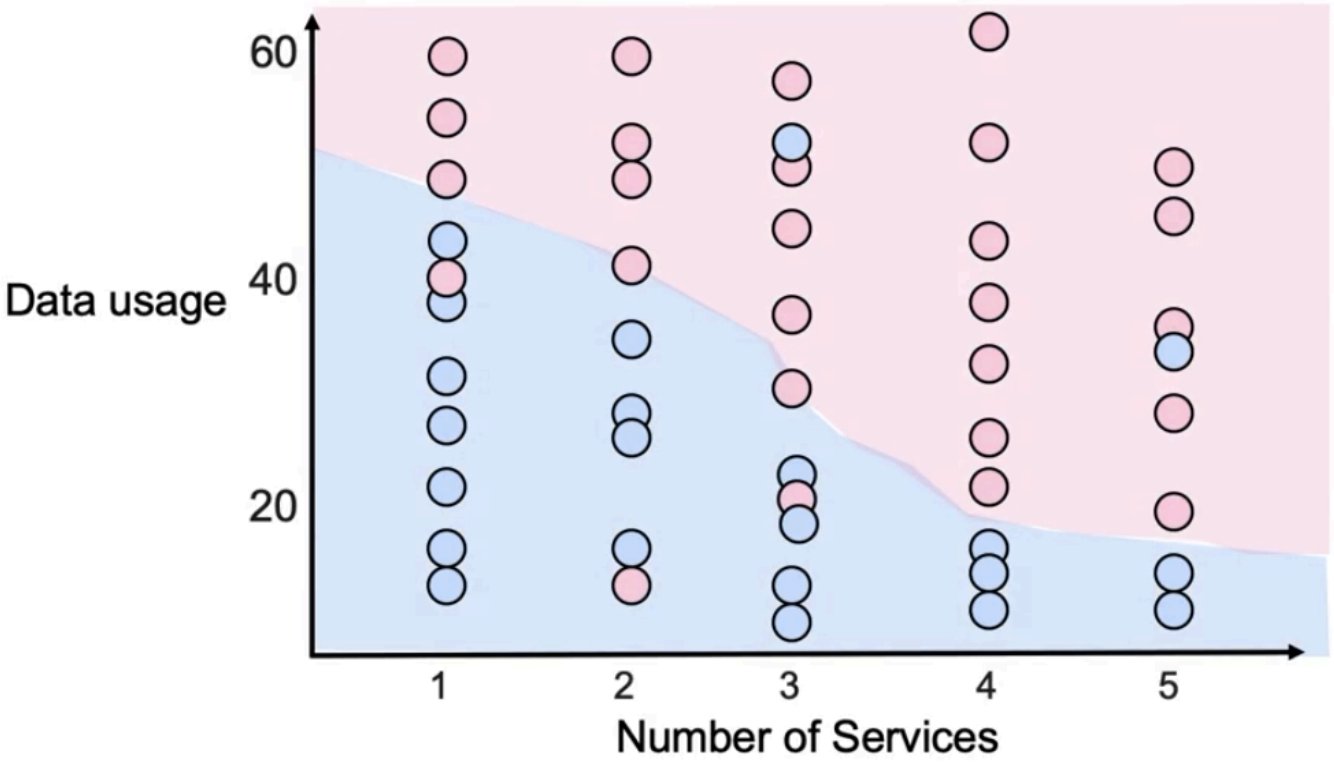


which when zoomed in shows this



So here, although the class matters more on the number of services, this feature will have a minimised impact and little effect on the prediction

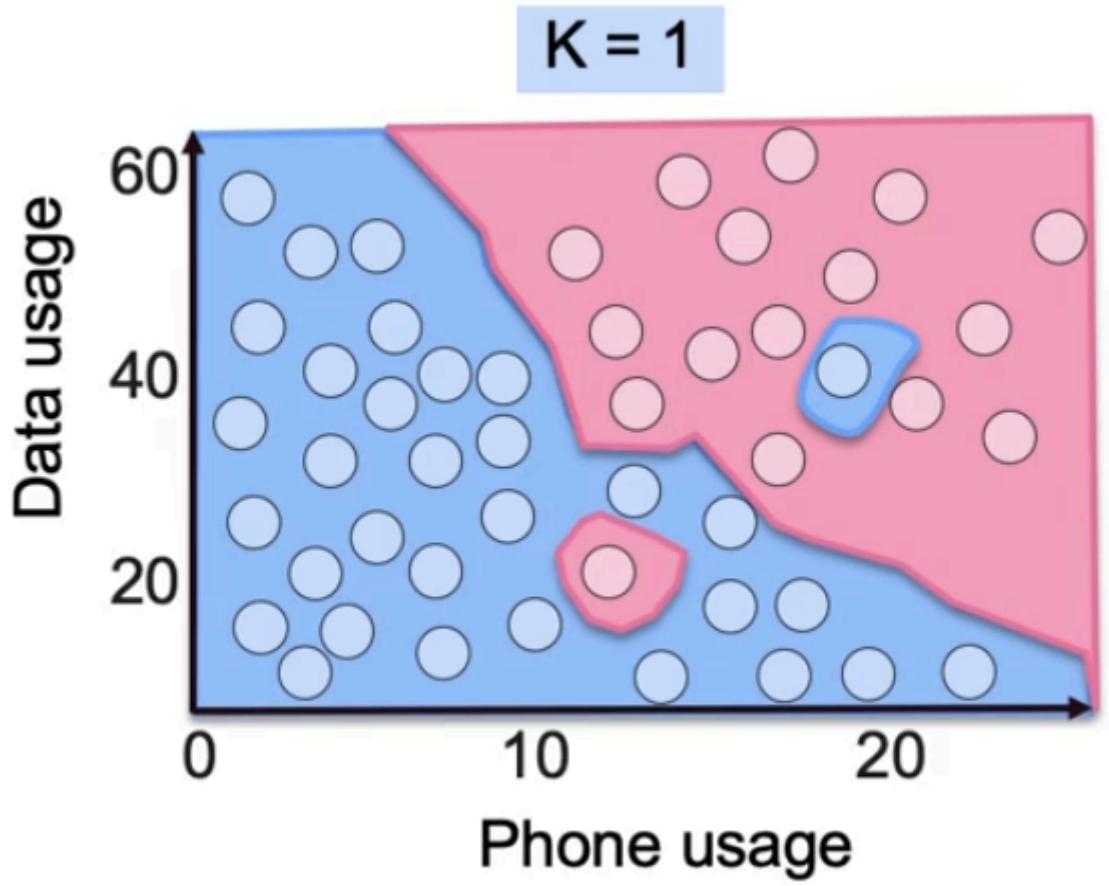
So after min-max scaling (the common scaling approach when using KNN, since it basically makes all dimensions have equal length), the graph will look something like:



Now both the features will have an equal influence on our model

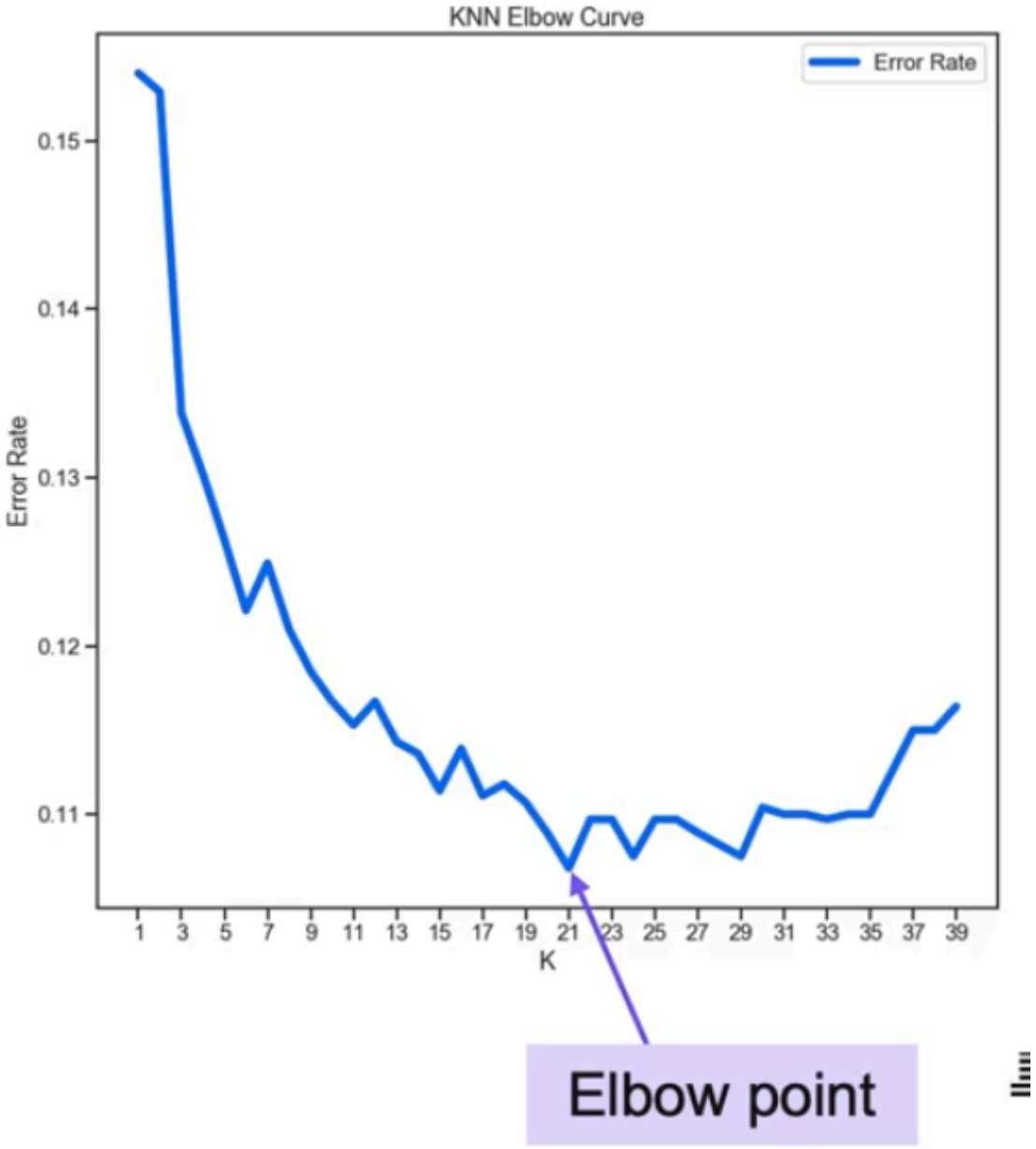
Decision boundary

It is basically a visual representation of like, what class will a portion of the graph be classified as. Basically for every pixel, which class would it belong to (as predicted by our model). For eg:



Choosing the right value of `k`

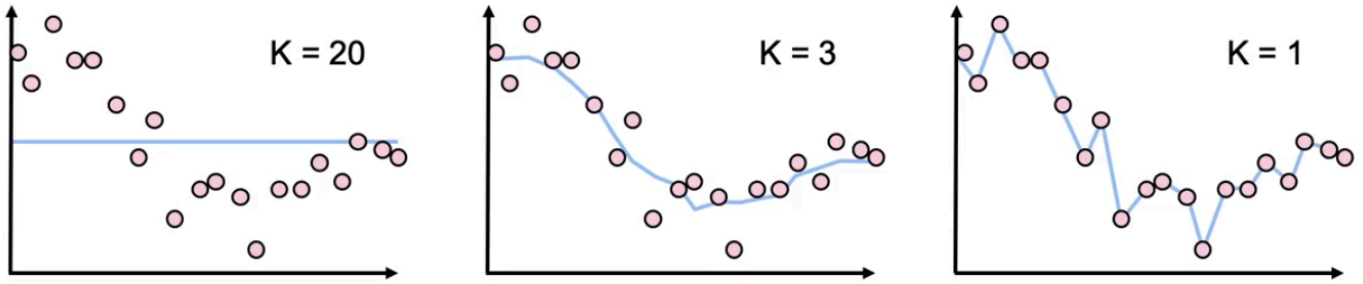
- The right value depends on which error metric is most important
- To find the best value of `k`, a common approach is the "elbow method" approach, where kinks in the curve of the error (as a function of `k`) is emphasised



Using KNN for regression

Regression can in theory be performed with KNN. We basically find the average of the labels of the `k` nearest neighbours

- If `k`=1, the model will act as a line graph
- If `k`>1, the model will basically act as a smoothing function
- If `k`=no. of points, the model will just calculate the mean of all the y values


K = 20
K = 3
K = 1