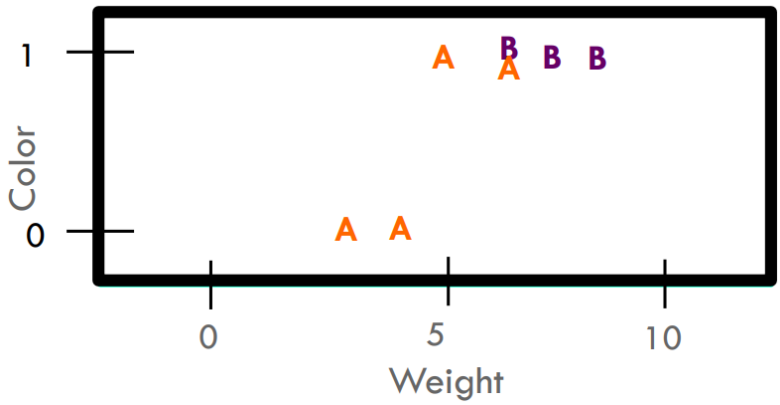
K-Nearest Neighbour

K-nearest neighbour (#K-NN) is a very simple [1] machine learning #algorithm . It takes a lazy learning approach.

Data Visualisation

The inner workings of #K-NN are easiest to understand when looking at how we can visualise the example data we're using to train our model. A simple way of doing this is to turn the #features of our examples into numerical values, such that they can be plotted in n-dimensional space for n #features .

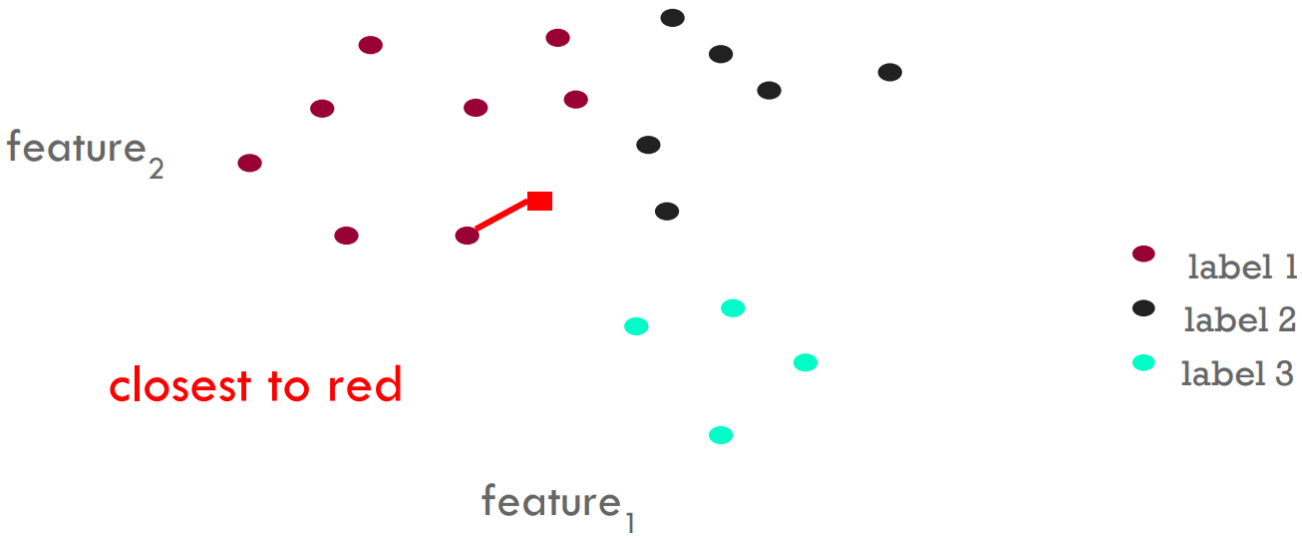
Weight	Color	Label
4	0	Apple
5	1	Apple
6	1	Banana
3	0	Apple
7	1	Banana
8	1	Banana
6	1	Apple



[2]

How it works

Intuitively, when given a new example, #K-NN simply looks at which group/class it lies closest to in the n-dimensional space and classifies it as that.



Unfortunately, #K-NN 's main weaknesses lies in its most attractive trait - its simplicity leaves a lot of things down to interpretation which allows us to make more mistakes when applying the #algorithm .

Problems

Problem

How do we classify an example that lies equidistant from its two closest neighbours, which are from two distinct classes?

Solution

Look at more neighbours! Specifically, look at the k nearest neighbours of our new example. We'll then set the label to the majority class of these k nearest neighbours.

Problem

How do we choose k?

Solution

- Allowing k to be even would put us at risk of encountering 'draws', where half the neighbours are from one class and the other half are from another.
- As a rule of thumb, $k < \sqrt{n}$ where $n = |\text{training set}|$
- Larger values of k will produce smoother boundary effects over the data
- If $k = n$ the predicted class will always be the majority class
- Often, values such as 3, 5, or 7 are chosen, but it should usually be set relatively high
- Use the #validation-set to gauge a good value
- Use #cross-validation [1]

1. Train several models on subsets of the training data and evaluate them all on the same, complementary, testing set. ↩

Problem

How do we define "nearest"?

Solution

We could use the examples with the shortest Euclidean distance

Problem

#Features are very rarely #comparable [1]

Solution

We can #standardise the #features by dividing them by their corresponding standard deviation.

- #Standardisation or #Z-score-normalisation
  - Rescale the data so that the mean is 0 and the standard deviation from the mean is 1
  - $x_{norm} = \frac{x-\mu}{\sigma}$
  - Where μ is the mean and σ is the standard deviation from the mean.
- #Min-max-scaling
  - Scale the data to a fixed range[1]
  - $x_{norm} = \frac{x-x_{min}}{x_{max}-x_{min}}$

---

1. between 0 and 1 ↩

---

1. they're rarely measured with the same units ↩

Definition

Distance is a numerical measure of how different two datapoints are. A lower value means they are more alike, with the minimum value generally set to 0. The upper limit varies depending on the context.

Definition

Similarity is a numerical measure of how alike two datapoints are, with a higher value meaning they are more alike. Values for similarity are usually in the range $[0, 1]$.

Solution

We could use the examples with the shortest #Minkowski distance

Definition

The #Minkowski distance is a generalisation of the Euclidean distance, defined by

$$D(a, b) = (\sum_{k=1}^{p} \mid a_k - b_k \mid^r)^{1/r}$$

where p is the number of dimensions and r is a parameter to be chosen:

- $r = 1 \therefore$ city block/Manhattan/"L_1 norm" distance
- $r = 2 \therefore$ Euclidean distance
- $r \to \infty \therefore$ "supremum"/"L_∞ norm" distance
  - This is the maximum difference between any component of the vectors

Solution

We could apply cosine similarity to the vectors representing pairs of examples.

Given two vectors d_1 and d_2 representing occurrences of words in two documents, we can define

$$cos(d_1, d_2) = \frac{d_1 \cdot d_2}{|| d_1 || \times || d_2 ||}$$

where $\cdot$ indicates the vector dot product and $|| d ||$ is the length of the vector d.

Example

d_1= 3 2 0 5 0 0 0 2 0 0
d_2= 1 0 0 0 0 0 0 1 0 2

$d_1 \cdot d_2$= 3*1 + 2*0 + 0*0 + 5*0 + 0*0 + 0*0 + 0*0 + 2*1 + 0*0 + 0*2 = 5
$||d_1||$ = (3*3+2*2+0*0+5*5+0*0+0*0+0*0+2*2+0*0+0*0)**0.5**= (42)^**0.5**= 6.481
$||d_1||$ = (1*1+0*0+0*0+0*0+0*0+0*0+0*0+1*1+0*0+2*2)**0.5**= (6)^**0.5**= 2.245
cos(d1, d2) = .3150

We can see that if the two documents had no words in common, the final value we'd get would be 0, which in the case of the cosine would imply an angle of 90° between the two vectors[1] and therefore maximum distance. If, on the other hand, the documents had the same number of occurrences of each word, we'd end up with a value of 1 implying an angle of 0° and therefore minimum distance, maximum similarity.

---

1. representable as lines on a 2D plane, for simplicity ↩

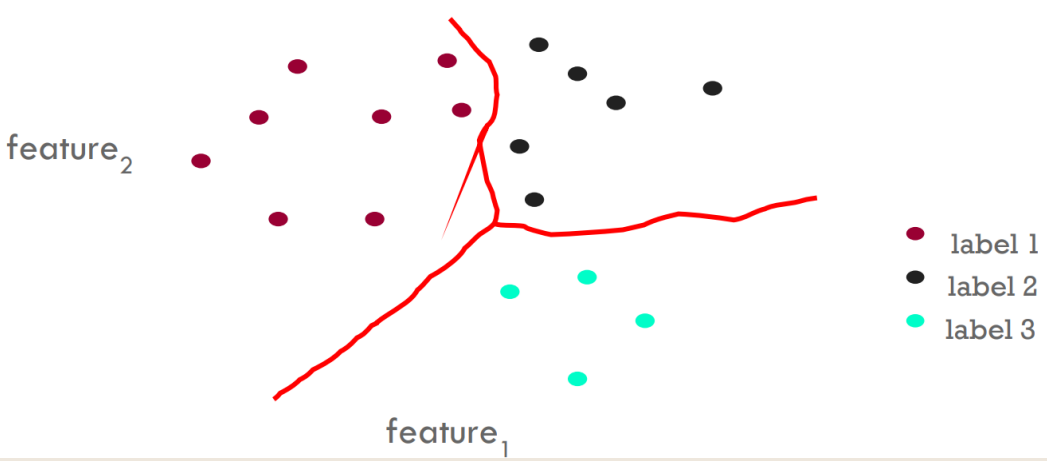
Solution

Calculate #decision-boundaries

Definition

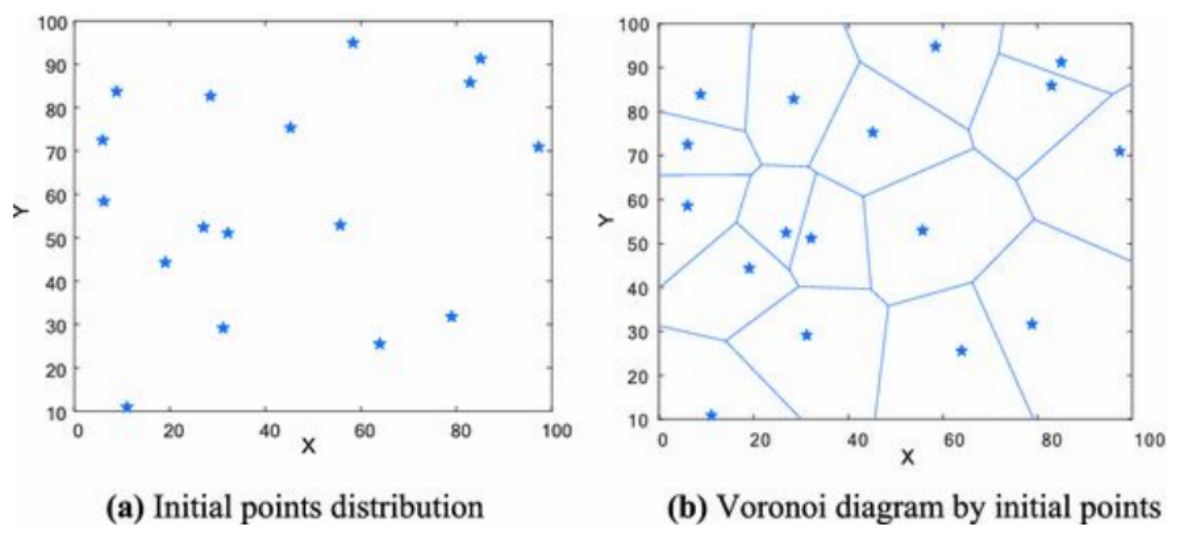
#Decision-boundaries are places in the #feature space where the #classification of a point/example changes, i.e., lines separating two classes.

Example



The red lines denote the #decision-boundaries .

A good way of computing these boundaries is using #Voronoi-diagrams , which describe the areas that are nearest to any point. Each line segment is equidistant from two points.

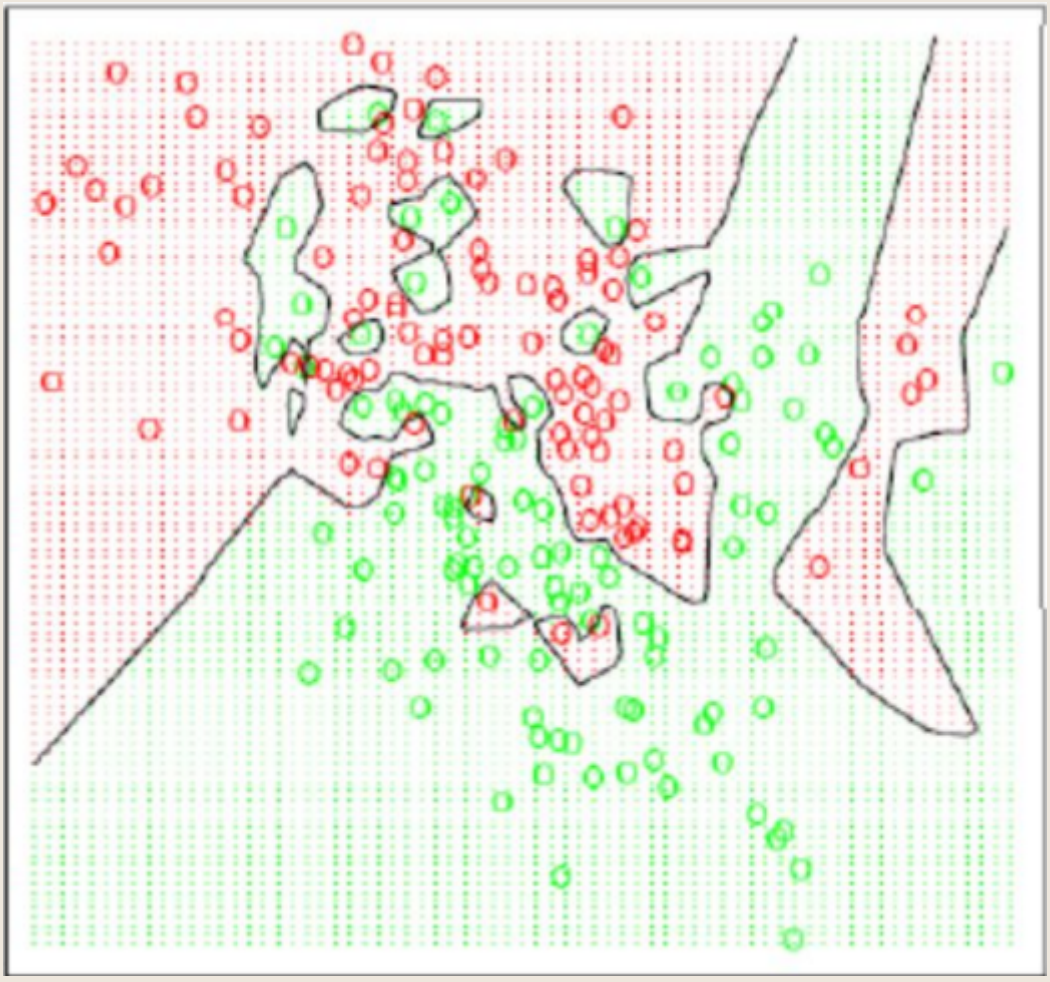


Note

#K-NN doesn't explicity compute decision boundaries; they form a subset of the #Voronoi-diagram for the training data.

Note

The more examples we store, the more complex the #decision-boundaries can become.



Solution

Instead of the k nearest neighbours, we could count the majority of all the examples within a fixed distance, weighting them so that closer examples have a higher impact on the outcome.

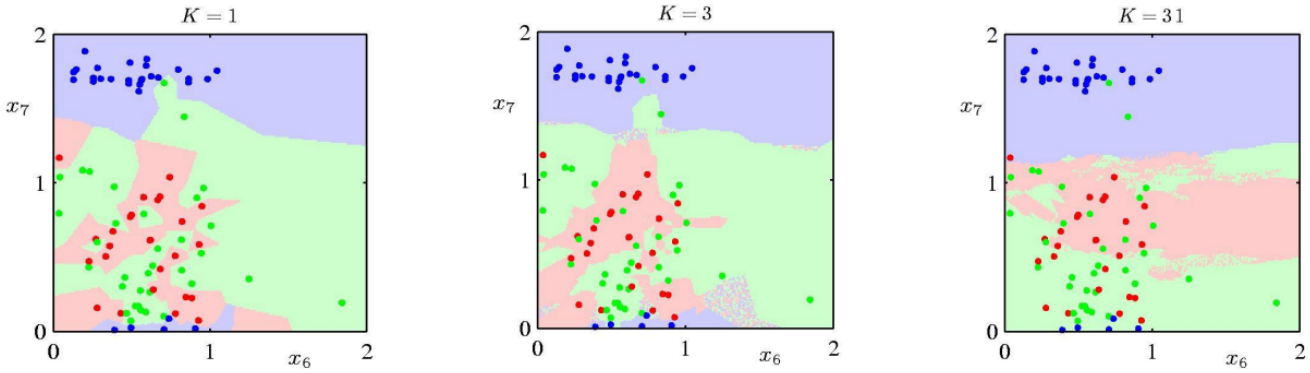
Problem

We often come across the #curse-of-dimensionality. In high dimensions, almost all points are far away from each other. The size of the data space grows exponentially with the number of dimensions, so the size of the data set must also grow exponentially in order to maintain the same density.

All machine learning #algorithms need a large data set for accurate predictions, but #K-NN is especially dependent on this factor as it requires points to be close in every single dimension. This means that without dramatic increases in the size of the data set, #K-NN loses all its predictive power.

The Impact of k

When choosing a value of k, it's important to understand how it relates to the concepts of #underfitting and #overfitting. Specifically, lower values of k will tend towards #overfitting, whereas larger values will tend towards #underfitting, to the point where setting k equal to n where n is the total number of examples will cause the model to always predict new examples as the majority class. Generally a larger value of k will produce a smoother boundary effect.



Computational Cost

#Pre-processing

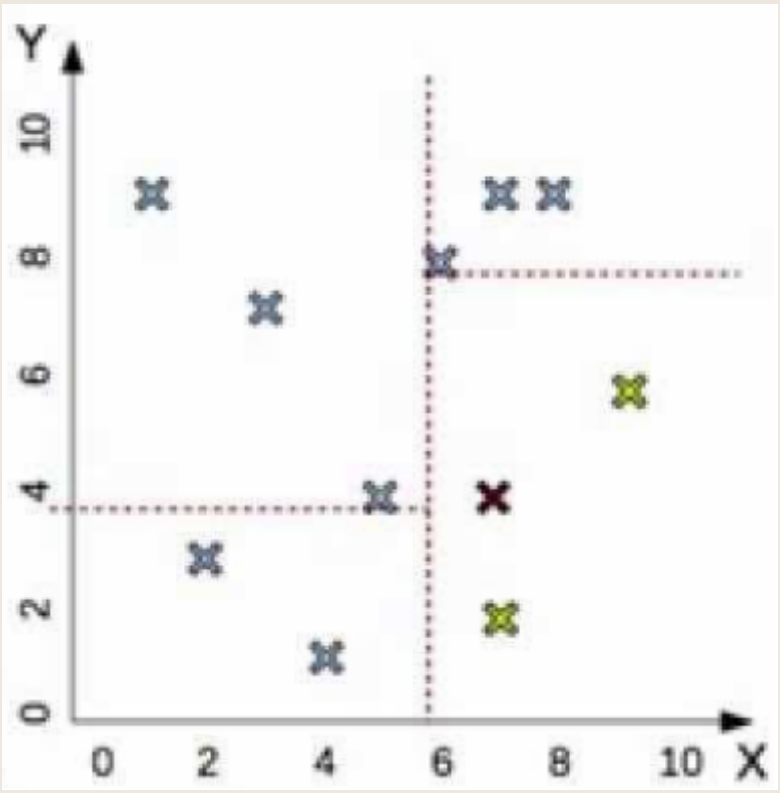
Usually based on a tree data structure, for example a "k-d tree" or "k-dimensional tree".

K-D Trees

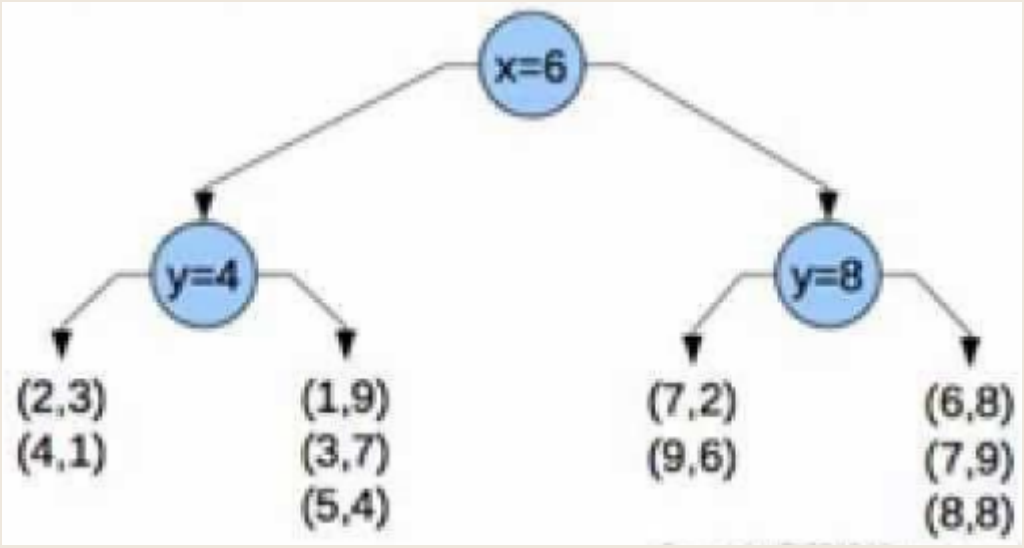
When building a k-d tree, we first plot the data in n-dimensional space. We then pick a random dimension, find the median, and split based on this median. We repeat this process for every remaining dimension.

Example

Given the (x, y) points $\{(1,9),(2,3),(4,1),(3,7),(5,4),(6,8),(7,2),(8,8),(7,9),(9,6)\}$, we could represent the corresponding k-d tree graphically like so:



When we represent this as a tree, it becomes clear how we would choose the class of a given point, for example $(7,4)$[1]:



1. the red point in the graphical representation ↩

No #Pre-processing

If no #pre-processing is applied to the training data, the #algorithm runs in linear time:

- Compute the distance for all n datapoints
- Complexity of distance calculation = $O(kn)$
- No additional space is needed

When to Consider #K-NN

#K-NN is only a viable option if there are fairly few[3] #features per item in the dataset and there is a very large amount of training data.

Advantages

- Very easy to program
- No optimisation or training required
- #Classification accuracy can be very good, and it can even outperform more complex models

Disadvantages

- Choosing distance measure can be difficult
  - Euclidean distance is the most common choice
- Must choose a good value for k
- #Curse-of-dimensionality
- Memory based
  - Must make a pass through the data for each #Classification
  - Can be prohibitive for large data sets

---

1. if not the simplest ↩
2. In this example we've kept one of the #features as a label to aid comprehension ↩
3. less than 20 ↩