KNN - K Nearest Neighbors

03 January 2025 08:58

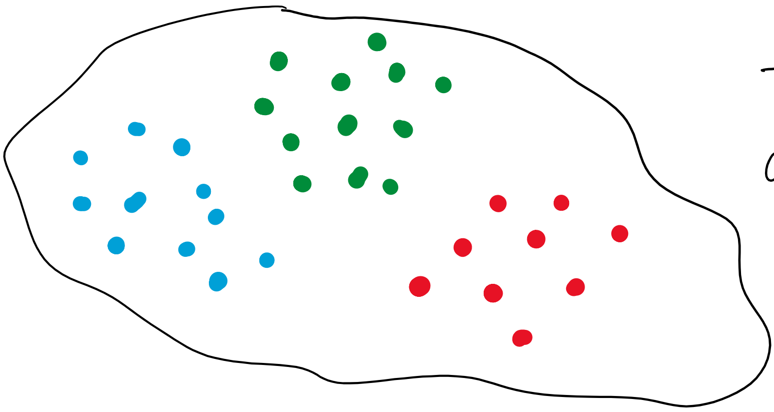
→ Blinkit -

← features → class

1 → High traffic

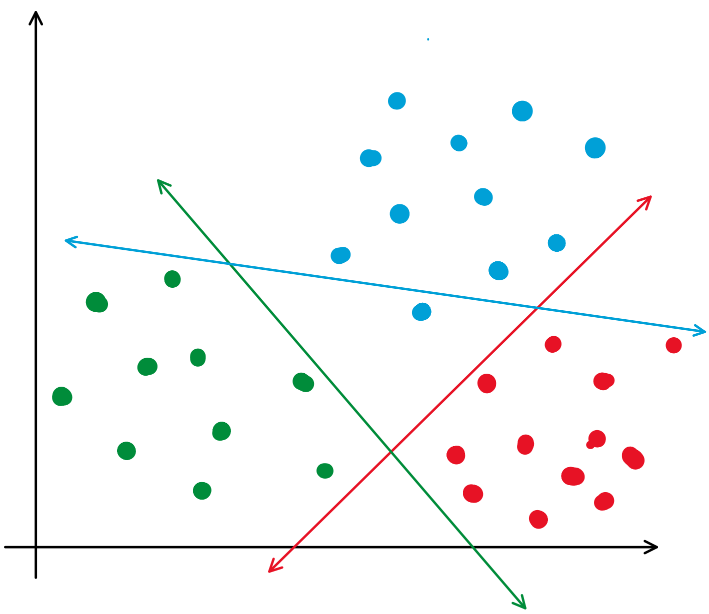
2 → Medium "

3 → Low "

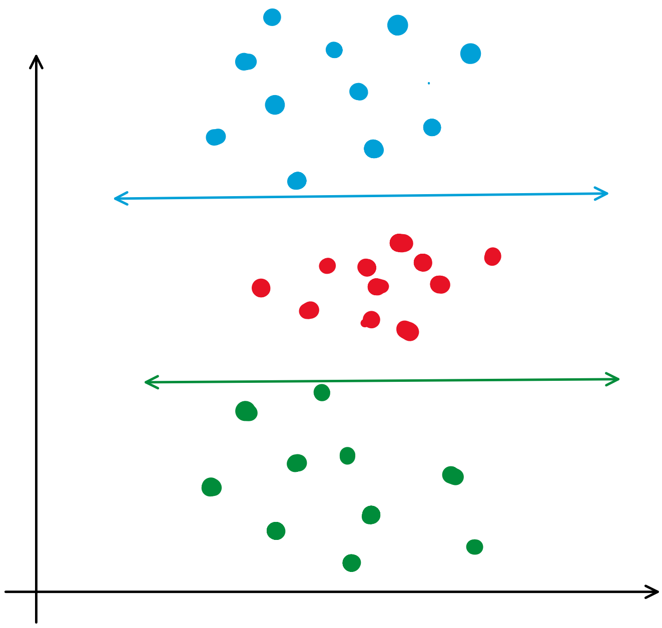


→ Where should I open my next warehouse? It makes sense to open it near to high traffic class.

☆ Why KNN ?

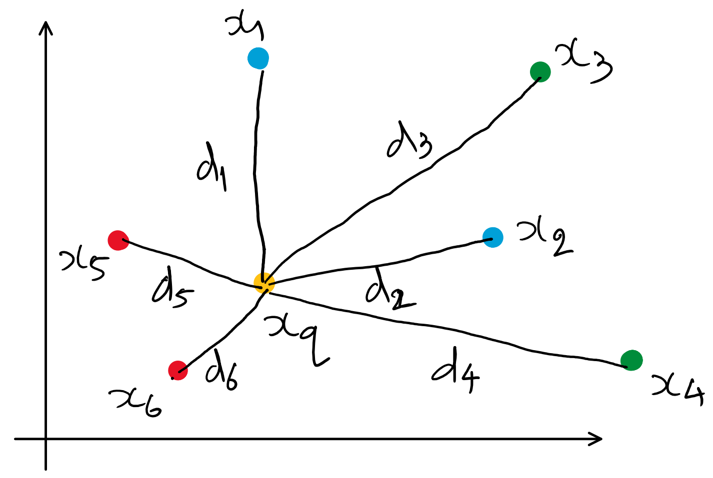


OVR works!



OVR doesn't work with Linear model.

☆ How does KNN work?



① Compute the distances of x_q from each point in the dataset. (Euclidean distance)

→ Here, we assume that we have reduced the dimensions of our data (X) to 2D using PCA.

In 2D space, Euclidean distance between $A(x_1, y_1)$ & $B(x_2, y_2)$ is given by $\sqrt{(x_1-x_2)^2 + (y_1-y_2)^2}$

But in nD space it can be given by:

$$\sqrt{(a_1-b_1)^2+(a_2-b_2)^2+\ldots+(a_n-b_n)^2}$$ for points $A(a_1, a_2, \ldots, a_n)$ & $B(b_1, b_2, \ldots, b_n)$

② Sort the distances in ascending order

$d_6, d_5, d_2, d_1, d_3, d_4$

③ Choose a value of "k"

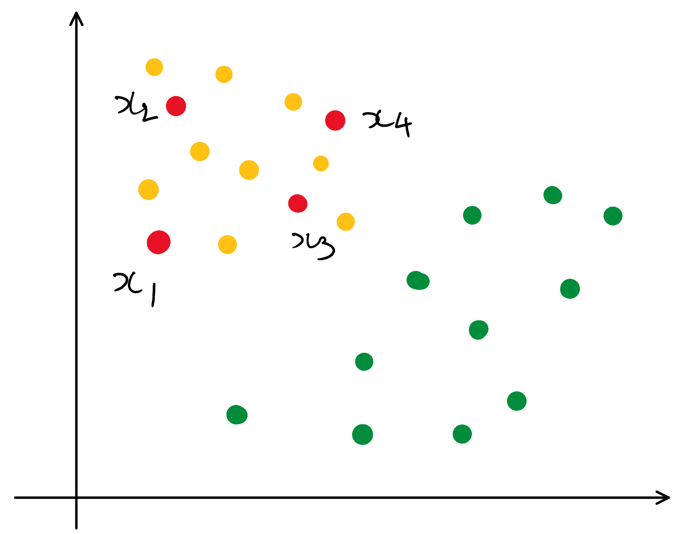
k	neighbors	classes	class of x_q - As per majority/mode
3	x_6, x_5, x_2	Red, Red, Blue	Red

2	x_6, x_5	Red, Red	Red
1	x_6	Red	Red
4	x_6, x_5, x_2, x_1	Red, Red, Blue, Blue	Randomly either Red or Blue
5	x_6, x_5, x_2, x_1, x_3	Red, Red, Blue, Blue, Green	"
6	$x_6, x_5, x_2, x_1, x_3, x_4$	Red, Red, Blue, Blue, Green, Green	Randomly from Red, Blue or Green

→ Usually we choose value of 'k' to be odd to reduce / avoid ties.

☆ Can we use this technique of KNN to deal with imbalanced data?

Ans - SMOTE



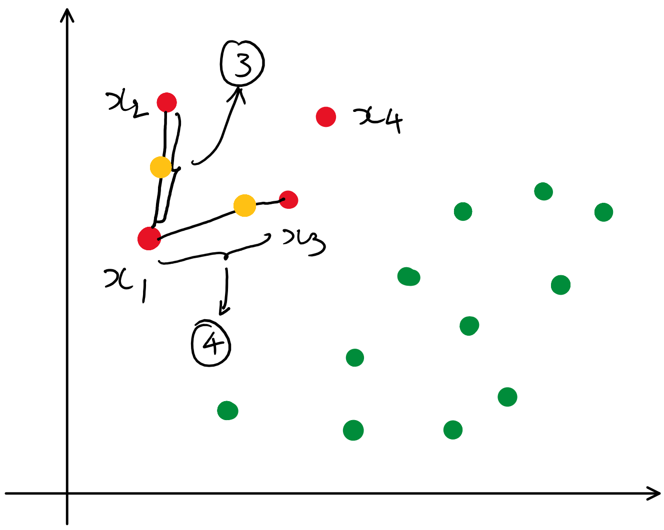
→ How did we make the imbalanced data balanced?

1. class weights
2. Under sampling
3. Over sampling by duplication

Can we use the logic of KNN to do over sampling but avoiding duplication?

Idea: If we generate neighbors of minority class using the logic of KNN until they become as many as the majority class (orange points) then they will nicely simulate as the real, balanced data!

SMOTE: Synthetic Minority Oversampling TEchnique



① Pick a value of k (let's say k = 2)

② Pick a random value $\alpha \in [0, 1]$ (say $\alpha = 0.5 / 0.75$)

③ New point:

$$x_{new} = x_{old} + \alpha * distance$$

kNN Code (in the colab)

★ Evaluating kNN

Good

① Handles multiclass problems better

② Extremely fast in training

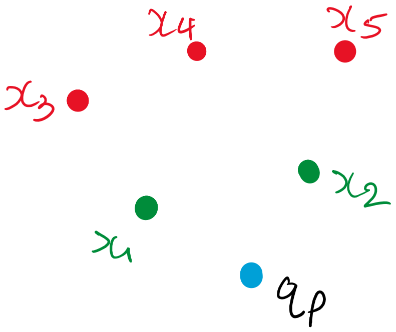
Not Good

① Extremely slow at inference for large no. of datasets as it needs to calculate distance of in training

(3) Simple + intuitive

(4) very good for data that is not linearly seperable

to calculate distance of q_p from each & every datapoints in the dataset.



Intuitively, $q_p \in$ Green class.

But what will be the kNN prediction for q_p if $k=5$?

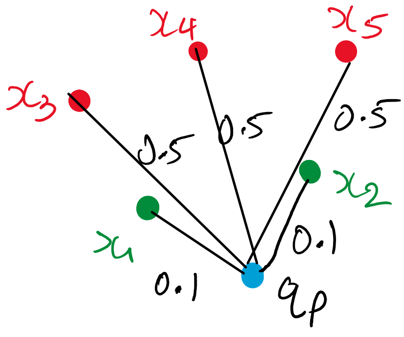
Ans – Red.

Why did this happen?

Because kNN gives equal importance to all datapoints. But ideally it should give higher weightage to x_1 & x_2 because they are near.

weight $\propto$ distance <u>OR</u> weight $\propto \frac{1}{\text{distance}}$?

$w = \frac{1}{d}$



$w_1 = \frac{1}{d_1} = \frac{1}{0.1} = 10$

$w_2 = \frac{1}{d_2} = \frac{1}{0.1} = 10$

20

$w_3 = \frac{1}{0.5} = 2$

$w_4 = \frac{1}{0.5} = 2$

$w_5 = \frac{1}{0.5} = 2$

6

This variant of kNN is known as:

This variant of kNN is known as:

Weighted kNN

Types of Distances

(1) Euclidean $\rightarrow \sqrt{\Sigma (x_{1i} - x_{2i})^2} = \left(\sum_{i=1}^{d} (x_{1i} - x_{2i})^2 \right)^{1/2}$

(2) Manhattan

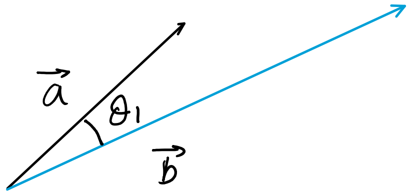
$\rightarrow \Sigma |x_{1i} - x_{2i}| = \left(\sum_{i=1}^{d} |x_{1i} - x_{2i}|^1 \right)^{1/1}$

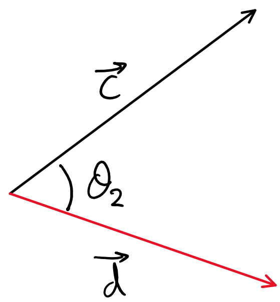
→ We use Manhattan Distance when some sort of 'route' is involved in the problem

Minkowski Distance

p^{th} minkowski distance $= \left(\sum_{i=1}^{d} (|x_{1i} - x_{2i}|)^p \right)^{1/p}$

(3) cosine distance





If $\cos\theta_2 > \cos\theta_1$ then the distance bet^n $\vec{c}$ & $\vec{d}$ is more than that of $\vec{a}$ & $\vec{b}$

$\therefore$ cosine distance = $\cos\theta$ where θ = angle bet^n the vectors

→ We use cosine distance when the dimensionality

→ We use cosine distance when the dimensionality is high as Minkowsky distances fail for higher dimension

④ Hamming Distance

$\vec{v}_1 : [1 \quad 0 \quad 2 \quad 1 \quad 3]$

$\vec{v}_2 : [1 \quad 1 \quad 0 \quad 1 \quad 3]$

NC C C NC NC

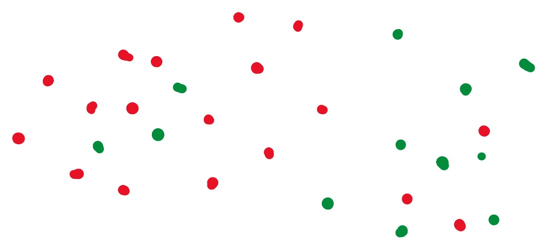
Hamming Distance = 2

☆ Bias - Variance trad-off in kNN:

suppose there are 100 datapoints

100 → 60 Red

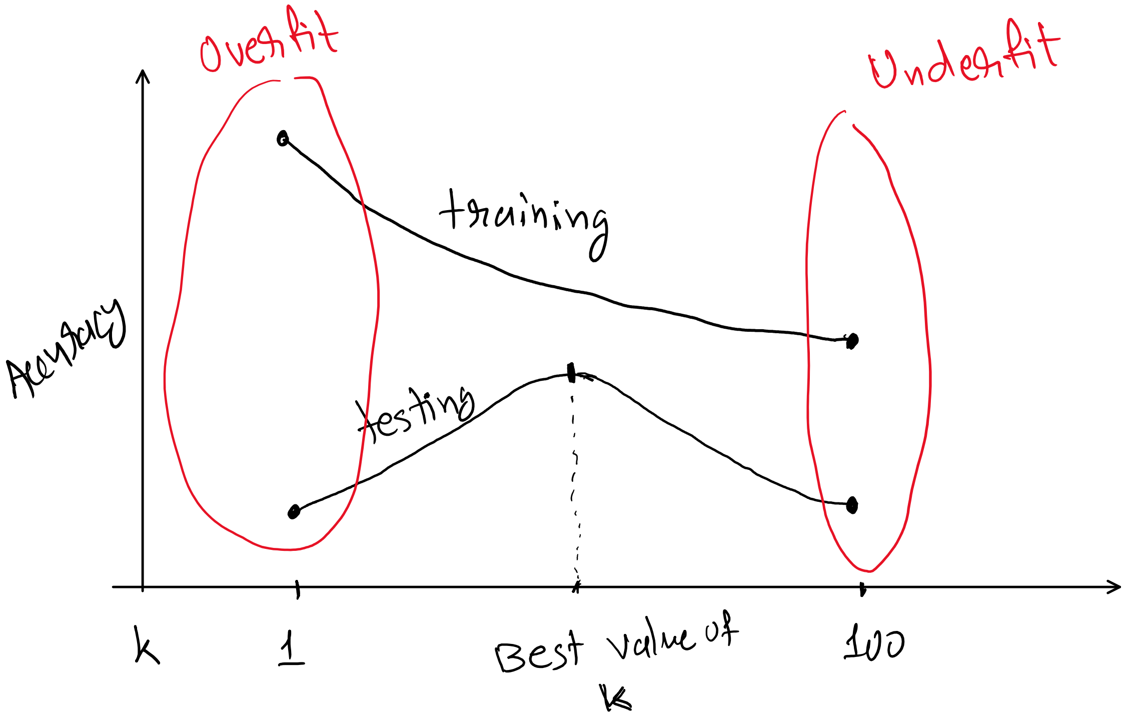
100 → 40 Green



If we take two extreme cases:

k = 1	What we want	k = 100
Overfit	perfect fit	Underfit
Low Bias	Low Bias	Low variance
High Variance	Low Variance	High Bias

That means we are looking for a right value of 'k'


Overfit
Underfit
Accuracy
training
testing
k
1
Best value of k
100