KNN-2

(1) Time complexity

Training : $O(1)$

Testing : we calculate distance of q_p from each points

Let's say Eucledian dist^n = $\sqrt{(a_1-b_1)^2+(a_2-b_2)^2+ \ldots\ldots + (a_d-b_d)^2}$

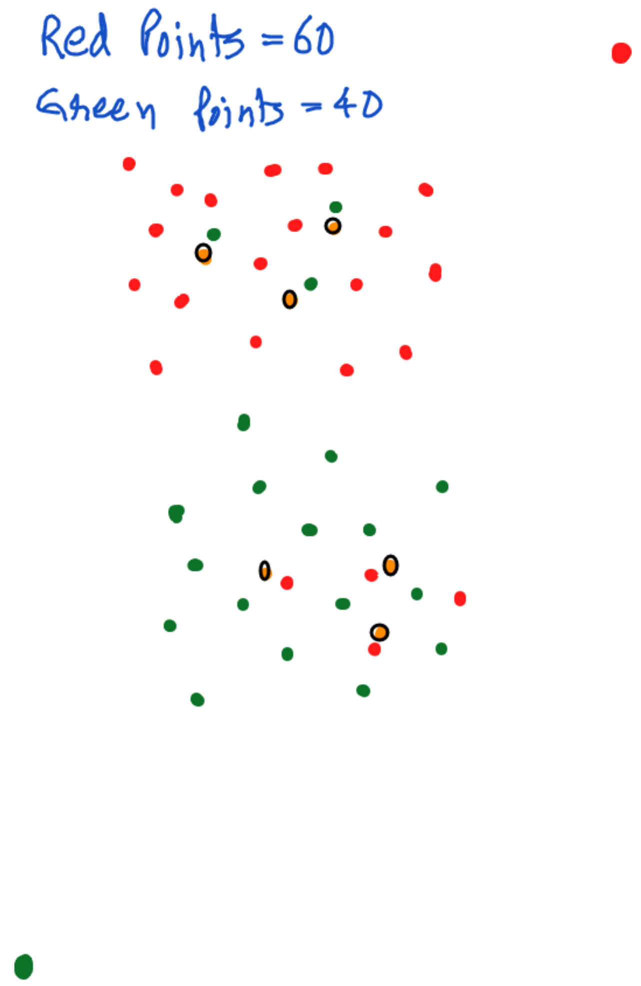
where d = no. of features

$\therefore$ time complexity grows with no. of features & total no. of datapoints.

Testing time complexity = $O(n.d + n\log n)$

(2) Impact of Outliers

Value of k	impact of outliers on the decision
1	Extremely High
5|7|10 - -	Some impact
100 (=n)	Extremely Less



③ Handling Categorical Features

Sol^n: Encoding

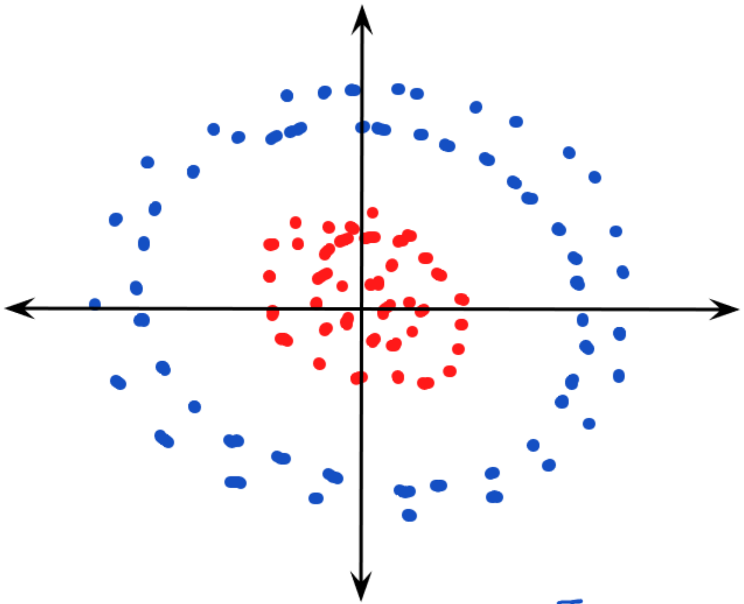
(A) One Hot Encoding – When we have less unique values in the categorical column (feature) as this will create a new dimension for each unique value.

(B) Label Encoding – This makes some values of the categorical feature more important than the others. Usually we use this when we have 2|3 classes (2-3 unique values in the categorical feature)

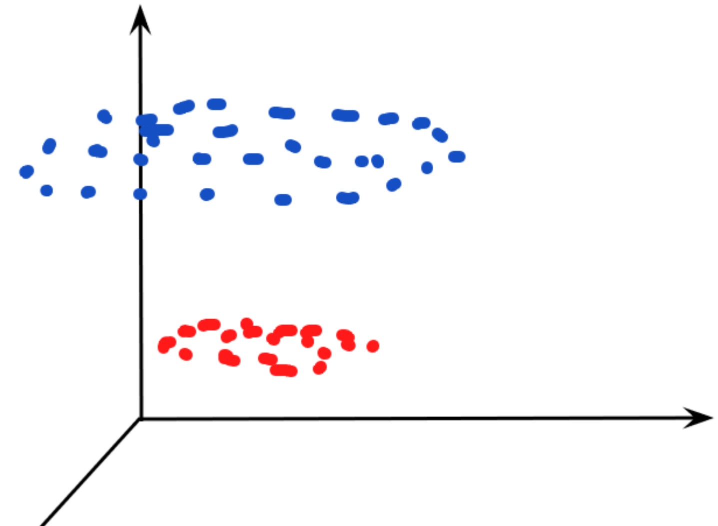
(C) Target Encoding – We take avg.

④ Choosing dimensionality & distance to use

→ Do we always want to reduce the dimensionality?



2D



3D

→ In case of higher dimension, we use cosine distance (cosine similarity) by: $\cos\theta = \dfrac{\vec{x}_1 \cdot \vec{x}_2}{\|\vec{x}_1\| \cdot \|\vec{x}_2\|}$

→ Qutub Minar : Red Fort, Taj Mahal,
India gate

→ Gateway of India : Hotel Taj, Marine drive

<u>Task-1</u> : Converting names of these places into numbers. This is known as vector embedding.

Qutub Minar : [0.6512 0.8585 0.1217 0.3459]

Red Fort : [0.5011 0.8680 0.3117 0.9429]

Marin Drive : [0.0123 0.1111 0.9723 0.0454]

distance bet^n these two vectors should be less

distance bet^n these two vectors should be more

→ Suppose there are 100 mn. such places. If we use kNN, as soon as a user enters a place (q_p), the algorithm will start computing its distance from **all 100 mn places**. This will take longer time to show the results. What is the solution to this?

Ans: Hash Table

Step-1: Create a function that returns same output for the datapoints close to each other. This function is known as **Hashing F^n**

Qutub Minar : [0.65 12 . . . .] $\xrightarrow{\text{Hashing Function}}$ [1 1 0]

Red Fort : [0.5011 . . . .] $\longrightarrow$ [1 1 0]

Marin Drive : [0.0123 . . . .] $\longrightarrow$ [1 0 1]

step-2: We create a table with 100 rows each having 1 mn places in it grouped by their hash value.

[110]	Qutub Minar, Red Fort, ...
[101]	Marine Drive, Hotel Taj, ...

→ A hash table is basically a dictionary with key-value pairs like:

{ [110] : [Q.M, R.F., ...],
[101] : [M.D., H.T. .....],
-- -- -- .. ..
- - .. --
}

Step-3: The search - As the user enters a place, it is converted to vector & feed to hashing function. Suppose hash value of this place comes to be [1 0 1] then we need to find 5 or 7 nearest neighbors to this point not from all 100 mn places but only from 1 mn places lying in the row of hash table corresponding to [1 0 1].