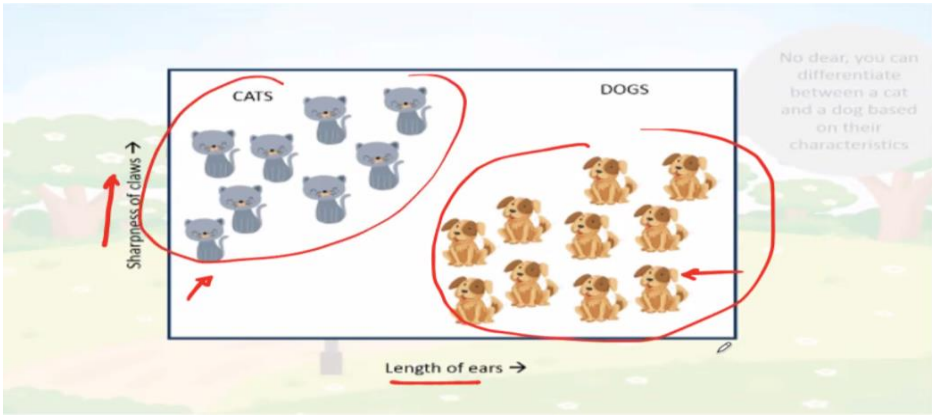
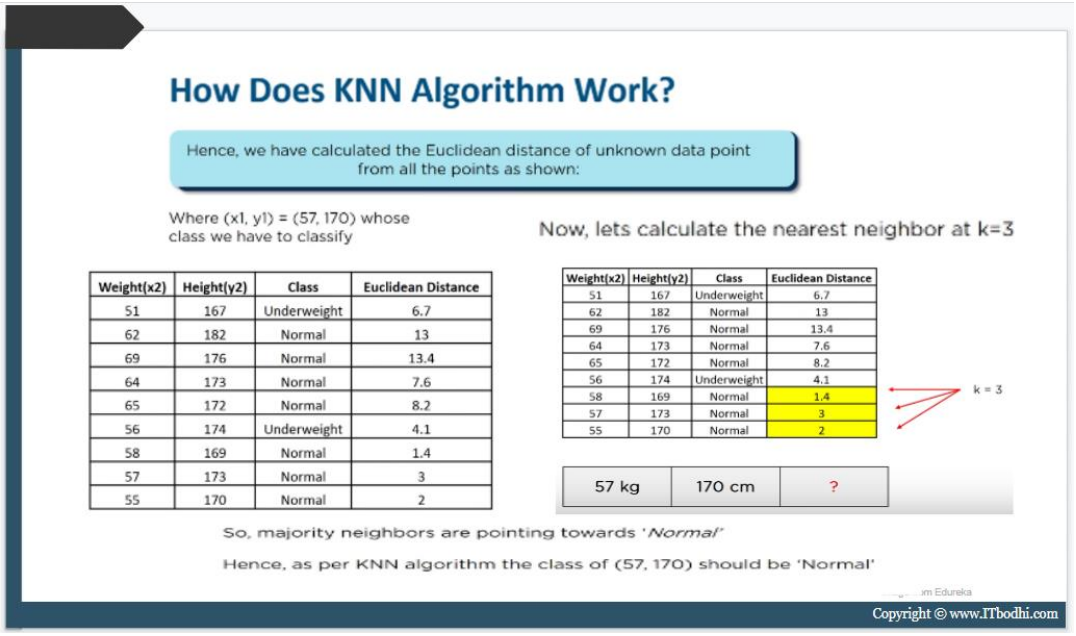
KNN Classifier

1. Supervised Algorithm
2. Classifies a new data point into classes depending on the features of the neighboring data points



3. For any new element k neighbors are referred to for classifying



4. Can also be used for Regression
    a. Groups according to features and calculates the average of 'k' nearest neighbors
    b. Here regression is the process of classifying the input and then finding out the average of the value of variable to be predicted in that group/cluster
5. **No Training Step**
    a. **Everything happens in Runtime.** It is a **<u>Lazy Algorithm</u>**
        - For Logistic Reg: Train-> Optimize -> Deploy, hence after training prediction time becomes extremely low
        - For KNN: Every time all the records will be analyzed and all the work takes place in production every time a new record is to be classified
        - **Extremely High Time Complexity and Extremely Space Time Complexity**
6. Cannot run on large datasets
7. Cannot run on processes that require quick outputs e.g.-> Fraud Transaction Detection

8. In case of small "K" values classification will be very sensitive to outliers i.e., highly unstable decision boundary [**Overfitting]**
9. In case of high "K" data becomes biased to the majority class [**Underfitting**]
    a. **How to choose "K"**?
        - Cross Validation
        - Grid Search CV
10. How to calculate similarity between attributes:
    a. Minkowski Distance ($\sqrt{\sum(\boldsymbol{x}-\boldsymbol{x}\ \ )\wedge\boldsymbol{N}}$
        - Chebyshev (***N=∞***)
        - Euclidian Distance (Numerical Attributes) (***N=2***)
            - If a smaller number of features, choose this
        - Weighted Euclidian Distance (Numerical Attributes)
            - If all features are not equally important
                - Use large value of K and Weighted Euclidean

            weighted Euclidean distance between A and B

            $$d(A,B)=\sqrt{\sum_i w_i(A_i-B_i)^2},$$

        - Manhattan Distance (Numerical Attributes) (***N=1***)
            - If high number of features, choose this
        - Hamming Distance (Categorical Attributes) (***N=0***)
            - Works only in 0 and 1
            - If both values are same, dis = 0 else 1

11. Every attribute is a different direction so if we have 3 attributes for each record,
    dis = Euclid ((x1, y1, z1), (x2, y2, z2))

12. Euclidean vs Manhattan Distance

a. Computing the root of a sum of squares (**RMSE**) corresponds to the **Euclidian norm**: it is the notion of distance you are familiar with. It is also called the **L-2 norm**(...)

b. Computing the sum of absolutes (**MAE**) corresponds to the **L-1 norm**,(...). It is sometimes called the **Manhattan norm** because it measures the distance between two points in a city if you can only travel along orthogonal city blocks.

c. The **higher the norm index, the more it focuses on large values and neglects small ones**. **This is why the RMSE is more sensitive to outliers than the MAE**. **But when outliers are exponentially rare (like in a bell-shaped curve), the RMSE performs very well and is generally preferred.**

13. Best Data Practices for KNN
    a. **Rescale Data**: KNN performs much better if all of the data has the same scale. Normalizing your data to the range [0, 1] is a good idea. **It may also be a good idea to standardize your data if it has a Gaussian distribution**.
    b. **Address Missing Data**: Missing data will mean that the **distance between samples can not be calculated**. These samples could be excluded or the missing values could be imputed.
    c. **Lower Dimensionality**: **KNN is suited for lower dimensional data as everything takes place in Runtime, also has high space and time complexity**. KNN can benefit from feature selection that reduces the dimensionality of the input feature space.

14. Algos for making KNN Fast
    a. Algos that make number of comparisons lesser either column-wise or row-wise can be useful

Making kNN fast

- Training: O(d), but testing: O(nd)
- Reduce d: dimensionality reduction
  - simple feature selection, other methods O(d^3)
- Reduce n: don't compare to **all** training examples
  - idea: quickly identify $m<<n$ potential near neighbors
    - compare only to those, pick k nearest neighbors → O(md) time
  - **K-D trees**: low-dimensional, real-valued data
    - O ($d \log_2 n$), only works when $d << n$, inexact: can miss neighbors
  - **inverted lists**: high-dimensional, discrete (sparse) data
    - O ($n'd'$) where $d'<<d$, $n'<<n$, only for sparse data (e.g. text), exact
  - **locality-sensitive hashing**: high-d, real-valued or discrete
    - O($n'd$), $n'<<n$ … bits in fingerprint, inexact: can miss near neighbors

    b. K-D Tree
        - Instead of comparing with the entire dataset, a random feature is picked up it's median value is found and data is split
        - One of the cons is if a neighbor is outside the boundary but is nearer to the data point we are looking to classify, it will be ignored
        - Other Observation is if k=5 and the shaded region has only 3 points , even then it will group only these 3 points , it will not skip the boundary to achieve the desired k-value
        - K-D Tree can be used if the situation can afford a trade-off between accuracy and computation time or computation complexity

c. Inverted List

-