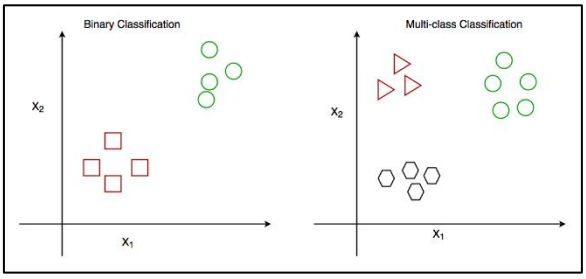
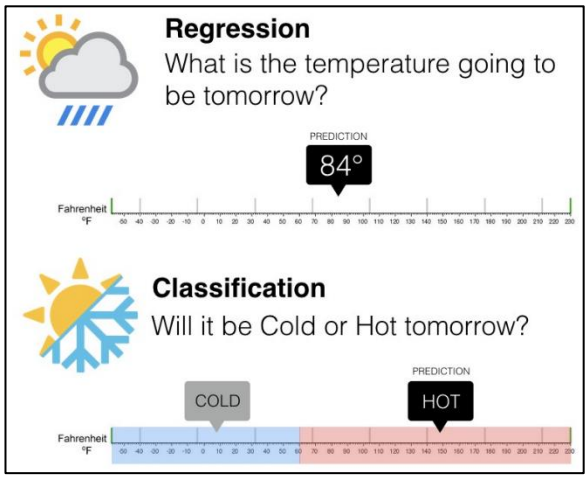
- **Classification & Regression**
  - Classification is the process of finding or discovering a model or function which helps in separating the data into multiple categorical classes i.e. discrete values. Classes are also called as targets, labels or categories.
  - Classification predictive modeling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (Y). e.g. spam detector in emails
  - We will be having a test dataset which contains all input variables & a class. Our task will be to find a function which will help us classify any future / unseen input.
  - If there are only two classes i.e., positive negative, true false. It's considered as binary classification e.g. amazon food reviews. If there are more than two classes .It's considered as multi class classification. E.g. MNIST Data set



  - The data matrix of size $n \times d$ i.e. $n\ data\ points\ and\ d\ features$ will be represented as,

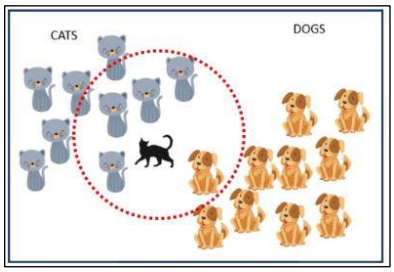
$$D_n = \{(x_i\ y_i)_1^n \nearrow\ x_i \epsilon \mathbb{R}^d, y_i \epsilon \{0,1\}\}$$

  - Given input matrix X; each row X_i represents a review text vector. for each review we have a class label which tells whether text is positive or negative. It is denoted as Y_i .
  - In Machine Learning we will leverage Linear Algebra for computations. We cannot input text into Linear Algebra. thus, even Y value is vectorized. Here we have two classes thus we can have a binary label 1 representing positive review, 0 for negative review
  - Regression is the process of finding a model or function for distinguishing the data into continuous real values.



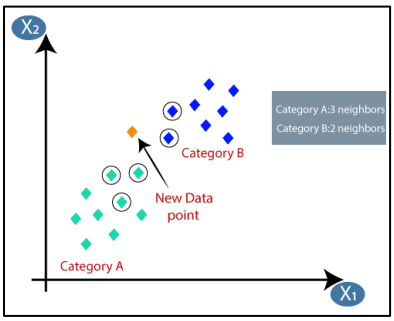
PARAMENTER	CLASSIFICATION	REGRESSION
Basic Output	Predefined classes	Continuous output
Involves	Discrete values	Continuous values
Nature of the	Unordered	Ordered
Example Algorithms	Decision tree, logistic regression, etc.	Regression tree (Random forest), Linear regression, etc.

- **K-Nearest Neighbors**



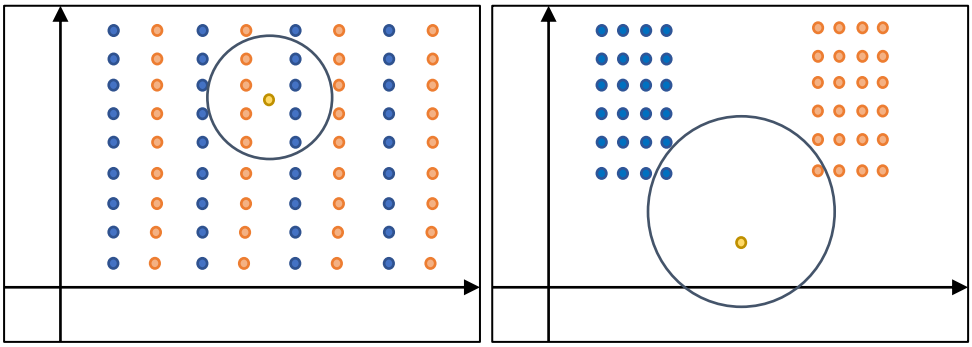
  - One of the simplest ML algorithms based on Supervised Learning technique.
  - It assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
  - It can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
  - It is a non-parametric algorithm, which means it does not make any assumption on underlying data.
  - It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
  - Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

- **KNN Geometric Intuition**



  - 'K' in KNN is a parameter that refers to the number of nearest neighbors to include in the majority of the voting process.
  - Suppose we have binary classification Category A & B.
  - To classify a new point, we will check nearest neighbors. As we can see there are 3 neighbors from category A & 2 from category B. the conclusion will be category A
  - It is always recommended that K shall be equal to an odd number. So that majority rule will work.

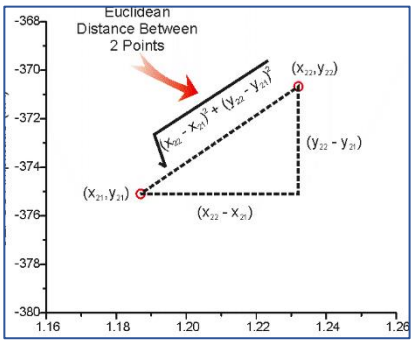
- **KNN Failure cases**



  - In **case-1**, the data is randomly spread and hence no useful information can be obtained from it. Now in such a scenario when we are given a query point (yellow point), the KNN algorithm will try to find the k nearest neighbors but since the data points are jumbled, the accuracy is questionable
  - In **case-2**, the data is grouped in clusters but the query point seems far away from the actual grouping. In such a case, we can use K nearest neighbors to identify the class, however, it doesn't make much sense because the query point (yellow point) is really far from the data points and hence we can't be very sure about its classification.

- **KNN Limitations**
  - **Doesn't work well with a large dataset:** Since KNN is a distance-based algorithm, the cost of calculating distance between a new point and each existing point is very high which in turn degrades the performance of the algorithm.
  - **Doesn't work well with a high number of dimensions:** Again, the same reason as above. In higher dimensional space, the cost to calculate distance becomes expensive and hence impacts the performance.
  - **Sensitive to outliers and missing values:** KNN is sensitive to outliers and missing values and hence we first need to impute the missing values and get rid of the outliers before applying the KNN algorithm.

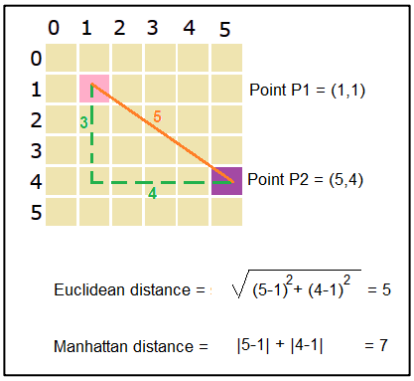
- **Distance measures: Euclidean(L2) , Manhattan(L1), Minkowski, Hamming**
  - **Euclidean($L2$ Norm)**



$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i{}^2 - y_i{}^2)}$$

where x & y are n dimensional vectors

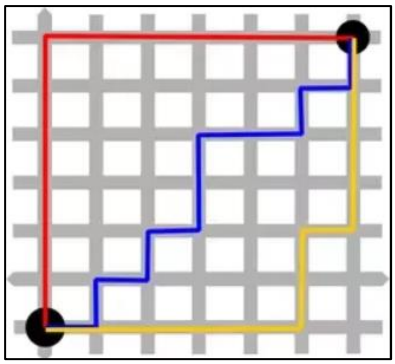
  - **Manhattan($L1$ Norm)**



  - The Manhattan distance between two vectors (or points) x and y is defined as

$$d(x,y) = \sum_{i=1}^{n}|x_i - y_i|$$

  - over the dimensions of the vectors.
  - This is known as Manhattan distance because all paths from the bottom left to top right of Manhattan city have the same distance



  - **$L0$ Norm**
  - Corresponds to the total number of nonzero elements in a vector.
  - For example, the L0 norm of the vectors (0,0) and (0,2) is 1 because there is only one nonzero element.
  - example, when having two vectors (username and password).

L0 Norm	Username	Password	Login
0	Correct	Correct	Success
1	Correct	Incorrect	Fail
1	Incorrect	Correct	Fail
2	Incorrect	Incorrect	Fail

  - **Minkowski(Lp Norm)**

$$d(x,y) = \left(\sum_{i=1}^{n}|x_i - y_i|^{\frac{1}{p}}\right)^{p}$$

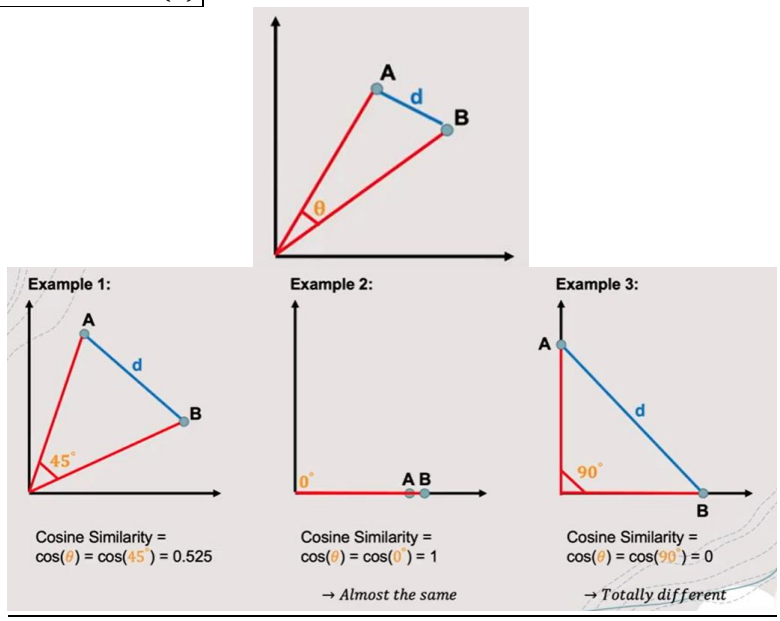
If p =2, it gives Euclidean distance(L2 Norm) & if p =1 , it gives Manhattan distance(L1 Norm)

  - **Hamming Distance**
  - The Hamming distance between two equal-length strings of symbols is the number of positions at which the corresponding symbols are different.
  - The symbols may be letters, bits, or decimal digits, among other possibilities. For example, the Hamming distance between:
    - **karolin***and* **kathrin***is* 3.
    - **karolin***and* **kerstin***is* 3.
    - **kathrin***and* **kerstin***is* 4.
    - **1011101** *and* **1001001** *is* 2.
    - **2173896** *and* **2233796** *is* 3.

- **Cosine similarity**

$$Cosine\ Sim_{A,B} = cos(\theta) = \frac{A\,.\,B}{\|\,A\,\|\,\|\,B\,\|} = \frac{\sum_1^n A_i\,B_i}{\sqrt{\sum_1^n A_i^2}\,\sqrt{\sum_1^n B_i^2}}$$

- Cosine similarity is a metric used to measure how similar the documents are irrespective of their size.
- Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space.
- The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together.
- The smaller the angle, higher the cosine distance.
- $Cosine\ Distance = d = 1 - cos(\theta)$



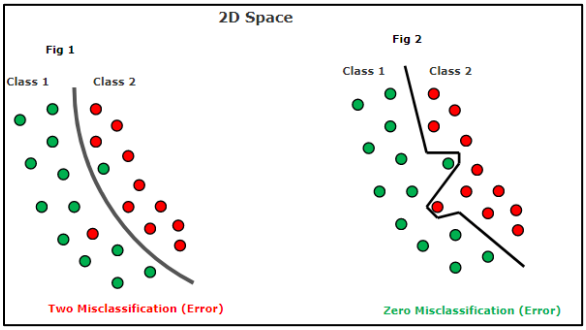
- If X1 & X2 are normalized to unit vectors,

$$[euc\ dist(X1, x2)]^2 = 2(1 - \cos(\theta)) = 2 * cosine\ sim$$

- **EFFECTIVENESS OF K-NN**
  - Given: a text reviews data with class labels;
    - Problem: What is polarity (+ve, -ve) of a new review?
    - Model- Knn nearest neighbours + majority vote
    - Use accuracy to measure the performance of the any ML model
  - Split the dataset into two sets train and test and there is no intersection between train and test data points (each data point either goes to train set or to test set)
  - Splitting can be done randomly into train set = 70% of the total dataset and test set = rest 30%
  - For each query point x_q, Predict y_q. Then accuracy is calculated as

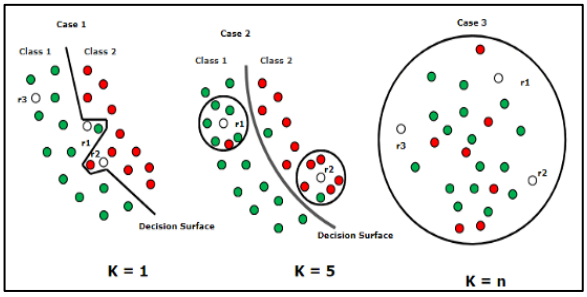
$$accuracy = \frac{n_{correct\ points}}{n_{total\ points}}$$

- **Decision Curve of k-NN**



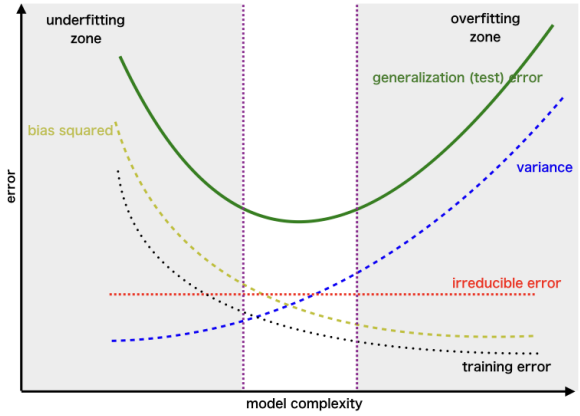
  - Fig 1 shows the dataset is divided by smooth curve into two classes i.e. 'Class 1 and Class 2'. There is misclassification of 2 points.
  - Fig 2 shows the dataset is divided by a non-smooth curve into two classes. i.e. 'Class 1 and Class 2'. There is no misclassification error that can be seen.
  - This curve Is called as decision curve. changing the value of K will defiantly change the structure of these decision surfaces. As 'K' increases, the smoothness of curve also increases.

- when K = n, all points will be classified as majority class irrespective of where point lies.



Case	k	Query Point	Predicted Class
1	1	r1	1
		r2	2
		r3	1
2	5	r1	1
		r2	2
3	n	r1	1
		r2	1
		r3	1

- **Bias Variance Trade off**
  - **Bias –** The bias error is an error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).
  - **Variance –**The variance is an error from sensitivity to small changes in the training set. High variance may result from an algorithm learning the random noise in the training data (overfitting). such models perform very well on training data but has high error rates on test data.
  - we want to choose a model that
    - accurately captures the regularities in its training data,
    - also generalizes well to unseen data.



  - it is typically impossible to do both simultaneously.
  - High-variance learning methods may be able to represent their training set well but are at risk of overfitting to noisy or unrepresentative training data.
  - algorithms with high bias typically produce simpler models that may fail to capture important regularities (i.e. underfit) in the data.
  - The more complex the model $\hat{f}(x)$is, the more data points it will capture, and the lower the bias will be. However, complexity will make the model "move" more to capture the data points, and hence its variance will be larger.

- **Maths behind bias & variance**
  - Let the variable we are trying to predict as Y for a continuous variable X

$$Y = f(X) + e$$

  - e is error term & it's normally distributed with mean = 0
  - We will make a model $\hat{f}(X)$ of $f(X)$ using linear regression or any other modeling technique.
  - So, the expected squared error at a point x is. E is mean

$$Err(x) = E\left[(Y - \hat{f}(x))^2\right]$$

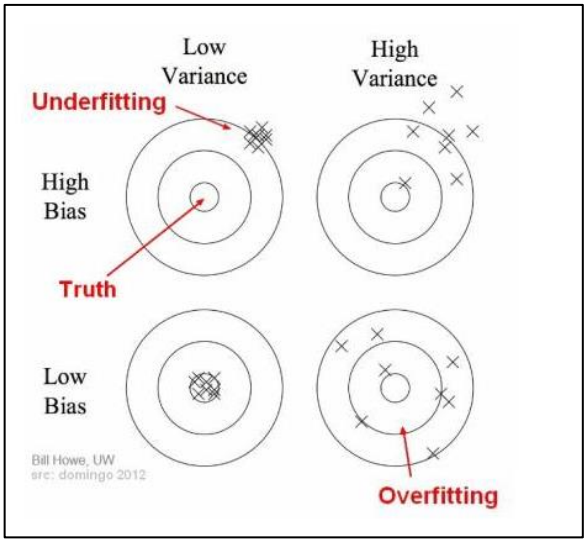
  - The $Err(x)$ can be further decomposed as

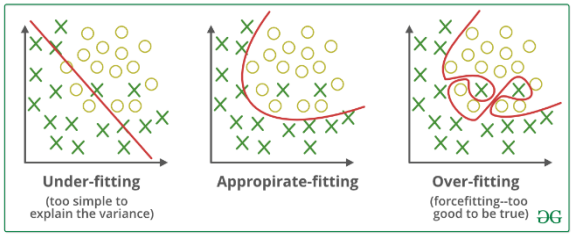
$$Err(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + \sigma_e^2$$

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

  - Irreducible error is the error that can't be reduced by creating good models.
  - It is a measure of the amount of noise in our data.
  - No matter how good we make our model, our data will have certain amount of noise or irreducible error that cannot be removed.
  - Bias = predicted value- actual value
  - Variance = (predicted value – mean predicted value)^2
  - Bias-Variance Decomposition - mlxtend

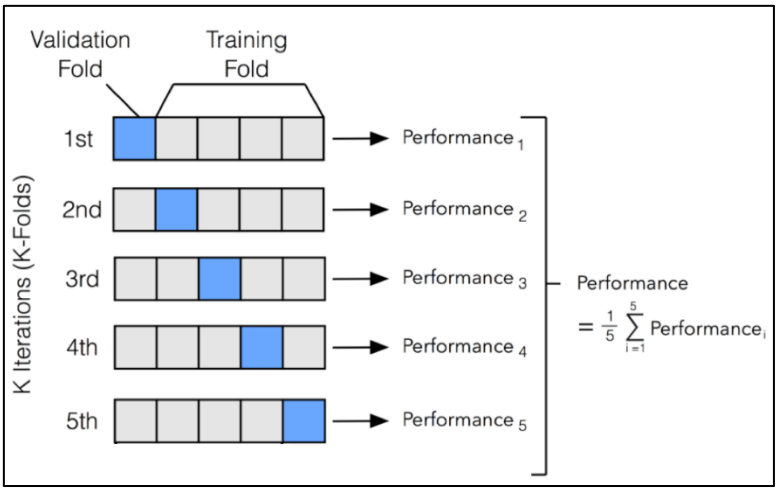
- **Over Fitting and Under Fitting**





  - **Underfitting** – High bias and low variance
  - Occurs when it cannot capture the underlying trend of the data. (It's just like trying to fit undersized pants!)
  - Its occurrence simply means that our model or the algorithm does not fit the data well enough.
  - It usually happens when we have less data to build an accurate model and also when we try to build a linear model with a non-linear data. These models are very simple to capture the complex patterns in data like Linear and logistic regression.
    - Increase model complexity
    - Increase number of features, performing feature engineering
    - Remove noise from the data.
    - Increase the number of epochs or increase the duration of training to get better results.
  - **Overfitting** – High variance and low bias
  - Occurs when we train it with a lot of data (just like fitting ourselves in oversized pants!).
  - When a model gets trained with so much of data, it starts learning from the noise and inaccurate data entries in our data set.
    - The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore, they can really build unrealistic models.
    - These models are very complex like Decision trees which are prone to overfitting.
    - To Reduce Overfitting
      - Increase training data.
      - Reduce model complexity.
      - Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
      - Ridge Regularization and Lasso Regularization
      - Use dropout for neural networks to tackle overfitting.
    - When K=1, Underfit
    - When K=n, Overfit
- **Cross Validation**
    - In any ML model, if our model performs bad on test data, we either change the hyperparameters or change model. If we keep repeating this process there is a chance that we will achieve good accuracy on test data without overfitting.
    - But despite this It perform terribly on unseen data, the reason is you measured the generalization error multiple times on the test set, and you adapted the model and hyperparameters to produce the best model for that particular set. This means that the model is unlikely to perform as well on new data.
    - Solution is to holdout some data from train set. This holdout data is called as validation set'
    - Now, you train multiple models with various hyperparameters on the reduced train set and you select the model that performs best on the validation set.
    - After this validation process, you train the best model on the full training set (including the validation set), and this gives you the final model.
    - Lastly, you evaluate this final model on the test set to get an estimate of the generalization error

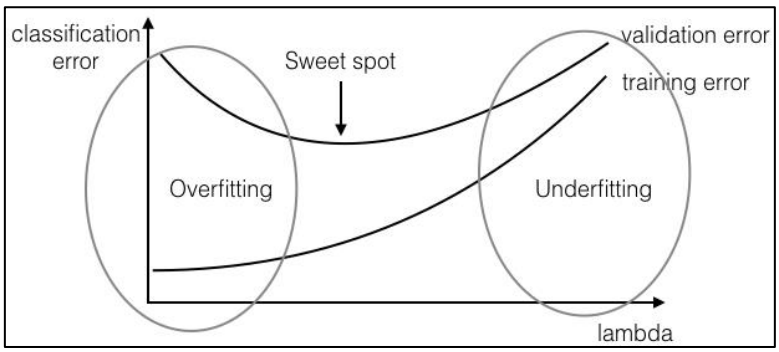
- **K-fold cross validation**
  - By reducing the training data, we risk losing important patterns/ trends in data set, which in turn increases error induced by bias. With k fold CV we will have enough data for train & cv.
  - After splitting the total data set (D_n) into training (D_{Train}) and test (D_{Test}) data set, in the ratio of 80:20, we further randomly split the training data into k equal parts. E.g. 5



  - We have to randomly split data set into k equal parts then we would have to compute k different accuracies for each value of hyperparameter and take their mean.
  - Time complexity for k folds will be k*nd making it O(knd) where k is the number of folds.
  - Generally, k =5,10 is preferred
- **How To determine underfitting or overfitting**

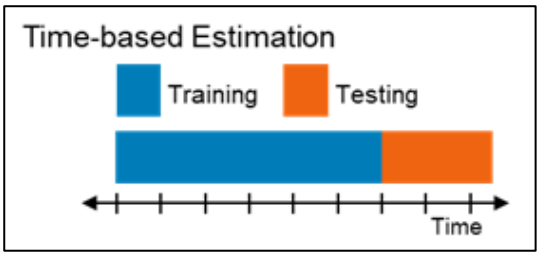
$$Accuracy = \frac{No\ of\ correctly\ classified\ points}{total\ no\ of\ points}$$

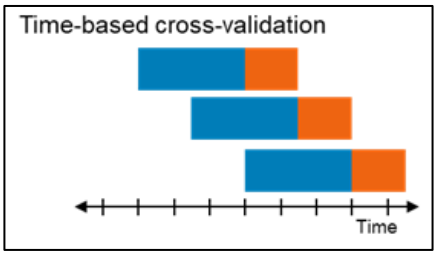
$$Error = 1 - Accuracy$$



  - **Overfitting: Training Error is Low & CV error is High**
  - The classifier learned on the training set is too specific, and cannot be used to accurately infer anything about unseen data.
  - Although training error continues to decrease over time, test error will begin to increase again as the classifier begins to make decisions based on patterns which exist only in the training set and not in the broader distribution.
  - **Underfitting: Training Error is High & CV error is also High**
  - The classifier learned on the training set is not expressive enough to even account for the data provided.
  - In this case, both the training error and the test error will be high, as the classifier does not account for relevant information present in the training set.

- **Time Based Splitting**

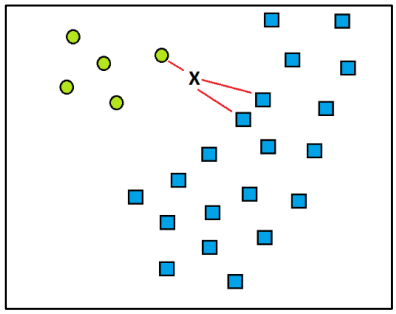




  - In most cases, train and test splitting is done randomly
  - When dealing with time-related and dynamically changing environments, where the characteristics of the environment change throughout time, it is best to use time-based splitting to provide statistically robust model evaluation and best simulate real-life scenarios.
  - For this we should use time-based cross validation, a method taken from the time-series field, which forms a type of "sliding window" training approach.
  - We will sort the data in the ascending order of timestamp
  - In time-based splitting, we will get reviews in train, cross validation and test set based on time stamps; the reviews that were written first occur in train set, then in cross validation and then in test set.
  - The argument here is we should avoid predicting past data based on models trained on future datasets.
  - Reviews or products change over time, new products get added or some products get be terminated.
- **k-NN for regression**
  - The regression is basically,

$$D_n = \{(x_i\ y_i)_1^n \text{ ⍰ } x_i \epsilon \mathbb{R}^d, y_i \epsilon \mathbb{R}\}$$

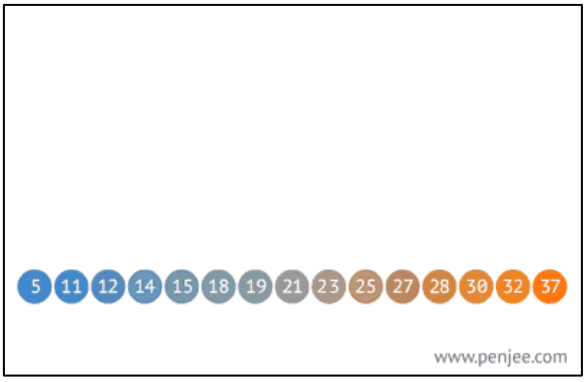
  - So, in k-NN regression if there is a query point surrounded by different neighbor points, we take mean or median of all values of neighbors and this mean or median will then become the value for query point.
  - For example, if there is a query point and if we take value of k=10,then we will take mean or median of ten nearest points to the query point and the value of this average will be assign to query point.
- **Weighted k-NN**



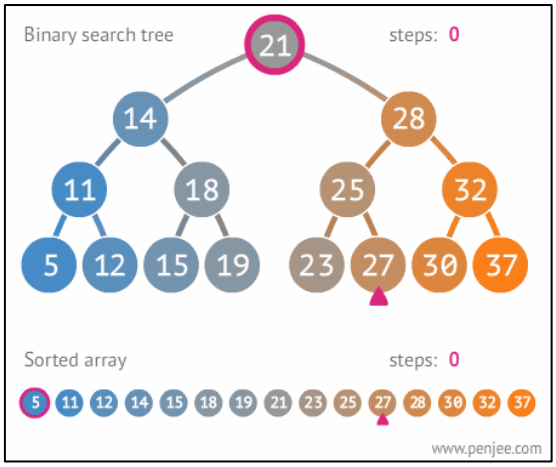
  - By standard KNN (k=3) , X should be a Blue Square as there are 2 Blue Squares vs 1 Green Circle. But in weighted KNN we calculate the weight (Likelihood) for each instance
  - In simple terms, we can compute the distance between query point and Nearest neighbor points. The weight would be inverse of dimension. I.e., more distance – less weight & vice versa
  - By considering weights, though query point has only one green circle in surrounding of query point. The weight is more as compared to blue points. Then X is Green Circle
- **Voronoi Diagram**

  - They can be used to visualize 1-NN
  - The closest pair of points corresponds to two adjacent cells in the Voronoi diagram.

- **Binary search tree**
  - Knn: Time complexity: O(n) if d is small and k is small and Space complexity: O(n)
  - We can reduce time complexity to O(log n) using binary tree = o(tree depth)
  - For BST, the left subtree of a node contains only nodes with values lesser than the node's values. The right subtree of a node contains only nodes with values greater than the node's value. The left and right subtree each must also be a binary search tree.
  - First the array is split at 21(middle value) then further left array is split at 14 and further at 11 unless it remains with only 2 elements which can't split further



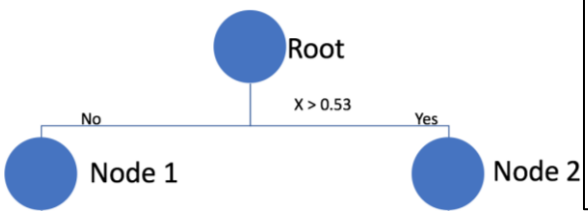
  - The time complicity for search in binary trees o(logn) or o(tree depth). E.g. to search 27, it requires 3 steps in BST while as in sorted array it requires 9 steps

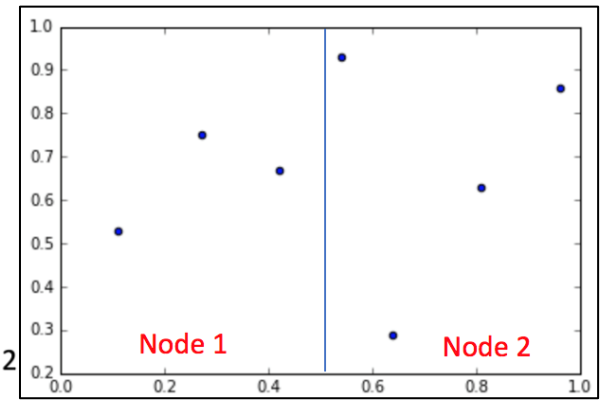


  - BinaryTreeVisualiser - Binary Search Tree
- **k-d Tree**
  - In the 2-Dimensional case each point has two values it's x and y coordinates. So, we will first split on the x-coordinate, next on y-coordinate and then again on x-coordinate and so on
  - Data-

	X	Y
0	0.54	0.93
1	0.96	0.86
2	0.42	0.67
3	0.11	0.53
4	0.64	0.29
5	0.27	0.75
6	0.81	0.63

  - Let's split data into two groups. We do it by comparing x with mean of max and min value.(median can also be used)
  - Value = (Max + Min)/2
    = (0.96 + 0.11)/2
    = 0.53

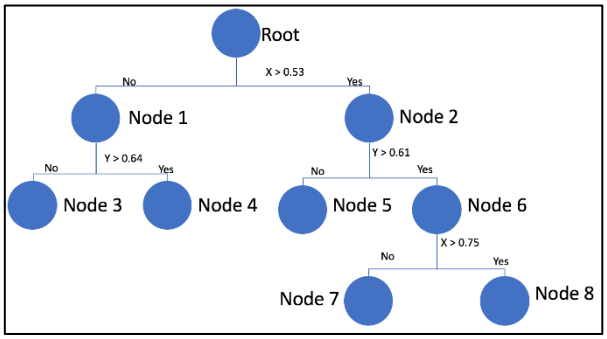


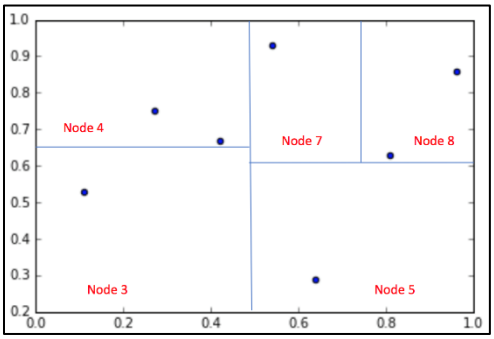


- At each node we will save 3 things.
  - Dimension we split on
  - Value we split on
  - Tightest bounding box which contains all the points within that node.

Tight bounds	X	Y
Node 1	0.11 <= X <= 0.42	0.53 <= Y <= 0.75
Node 2	0.54 <= X <= 0.96	0.29 <= Y <= 0.93

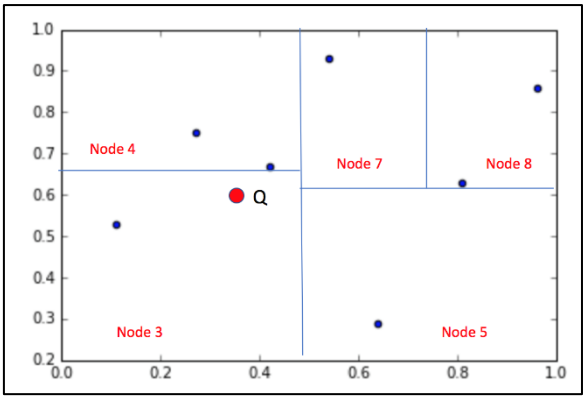
- Similarly dividing the structure into more parts on the basis of min & max, until we get maximum 2 data points in a Node.



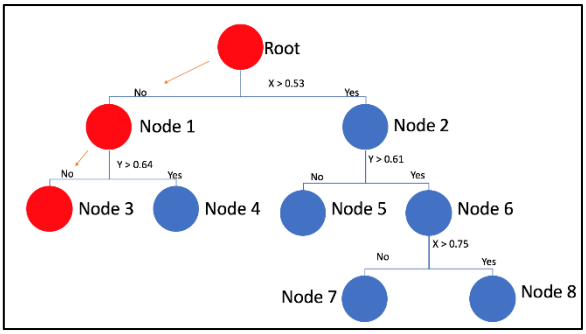


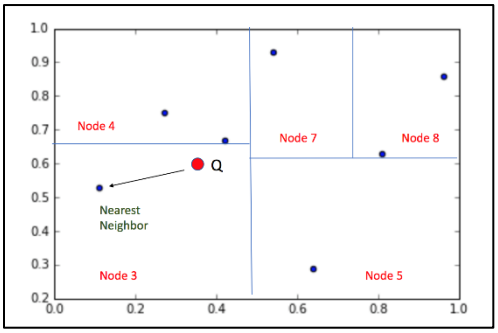
- A kd-tree for a set of n points uses O(n) storage and can be constructed in O(n logn) time
- What dimension to split? Choose the Widest dimension first or the alternating dimensions
- What value to split? Choose the median value or the Centre of all the values
- When do we stop splitting? When there are fewer data points are left than a specific value says m

- **How Nearest Neighbor Search Query works in a K-D Tree?**
  - Let's say now we have a query point 'Q' to which we have to find the nearest neighbor in above kd tree

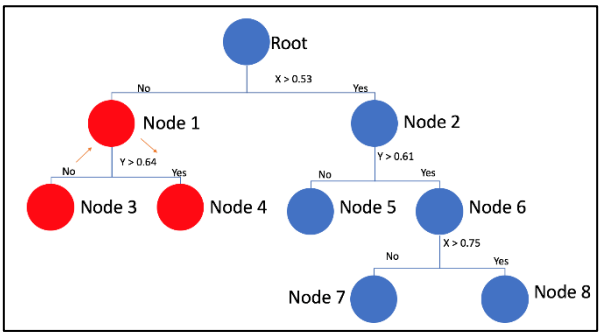


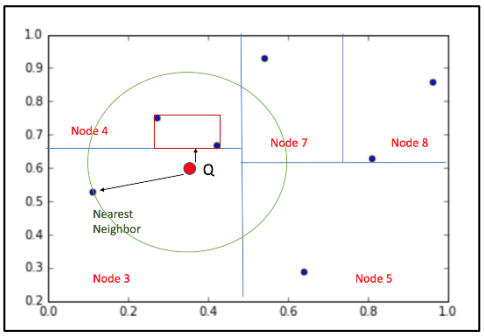
- Using the tree, we made earlier, we traverse through it to find the correct Node and using Node 3 to find the Nearest Neighbor.



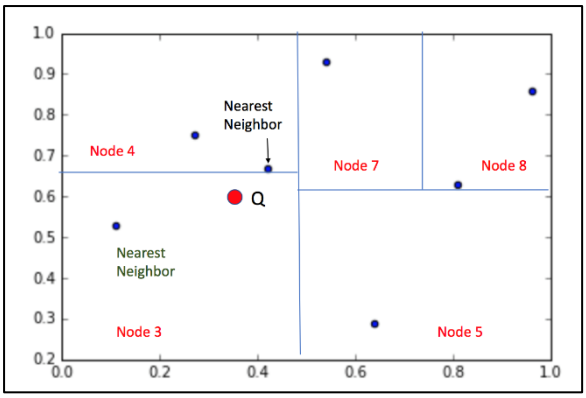


- But it is not the Nearest neighbor to the Query point.
- now traverse one level up, to Node 1.
- The nearest neighbor may not necessarily fall into the same node as the query point. We can check if we should inspect all points in node4 by checking if the tightest box containing all the points of Node 4 is closer than the current near point or not.

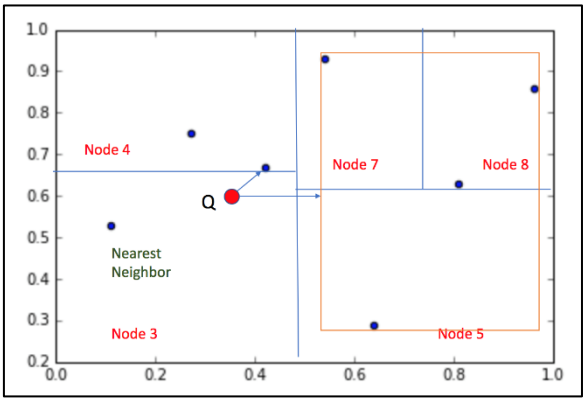




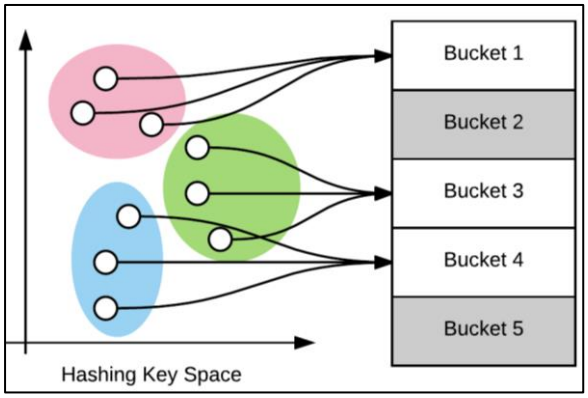
- The bounding box for Node 4 lies within the circle, So Node 4 may contain a point that's closer to the query. From the distance of the points within the Node 4 to query point ,the point lying above the query point is actually the nearest neighbor within the given points.



- When traversed up, in the tightest box in node2 is far from the current Nearest point. So, it can be pruned

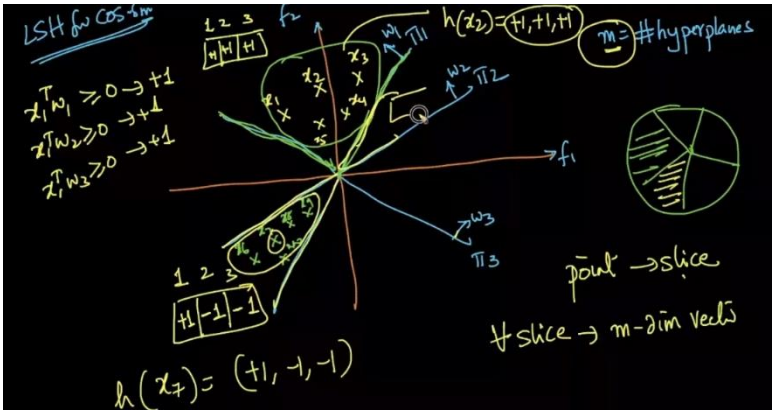


- **Locality-sensitive hashing**

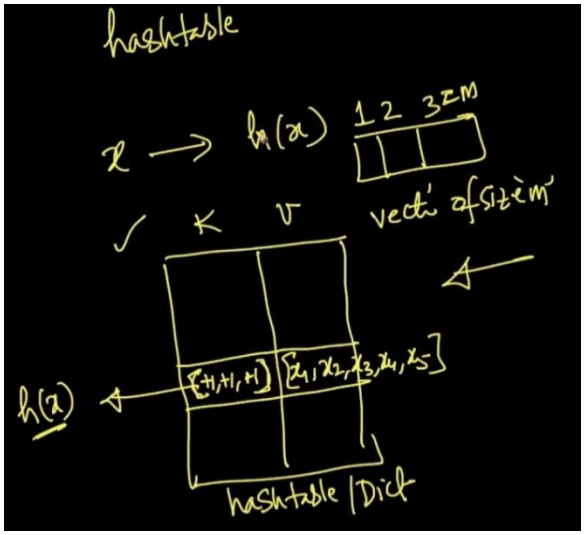


  - locality-sensitive hashing (LSH) is an algorithmic technique that hashes similar input items into the same "buckets" with high probability.
  - The number of buckets is much smaller than possible input items.
  - It will put all the points in a neighborhood inside one bucket/region/area in a hash table(this bucket will have a corresponding value attached to it) so that when we call a particular value the function directly points to the neighborhood (which has all the points in it) thereby drastically reducing the distance comparisons we will have to do for K-NN.

- **LSH for cosine similarity**



1. First make 'm' hyperplanes to split into regions and create slices such that cluster of points lie in a particular slice and be called their neighborhood. typically, m = log(n)
2. Next for each point create a vector(also called hash function) by W1(transpose).point. if it is greater than 0 it lies on the same side of that hyperplane else other side. Based on that create a vector of m size.
    a. For e.g. the vector can be [1,-1,-1] denoting point x lies on same side of normal to hyperplane 1, opposite side to normal of hyperplane 2 and 3. Now this vector serves as key to the hash table and all the points with the same key or vector representation will go in the same bucket as they have similar vector representation denoting they lie in the neighborhood of each other.
3. Now it may happen that two points which are very close fall on different slice due to placing of hyperplane and hence not considered as nearest neighbor. To resolve this problem, create l hash tables. (l is typically small). In other words, repeat step 2 l times thus creating l hash tables and m random planes l times. So, when a query point comes compute the hash function each of the 'l' times and get its neighbors from each of bucket. Union them and find the nearest neighbors from the list.

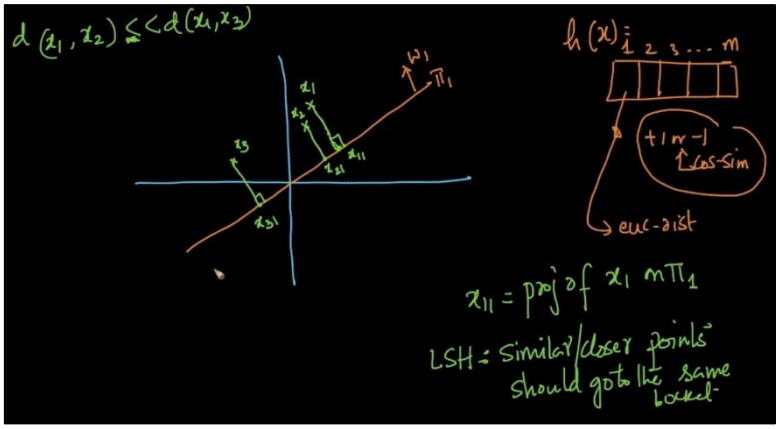


4. So basically, in each 'l' iterations create m hyperplanes and hence region splitting will be different thus vector representation or hash function of the same query point will be different in each of the representations. Thus, the hash table will be different as points which lied on the same region in previous iteration might lie in a different region in this iteration and vice versa due to different placement of hyperplanes. Time complexity is O(m*d*l) for each

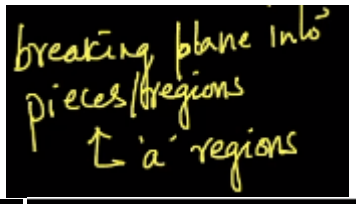
query point. And for creating the hash table O(m*d*l*n) which is one time only. Space complexity is O(n)

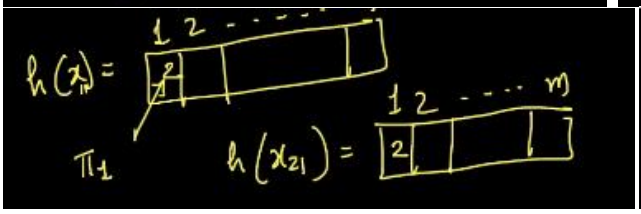
- **LSH for Euclidean distance**
  - Divide the plane into small parts.
  - Project each data-point on the planes.
  - For each datapoint take the distance along each plane and use it to calculate the hash value.

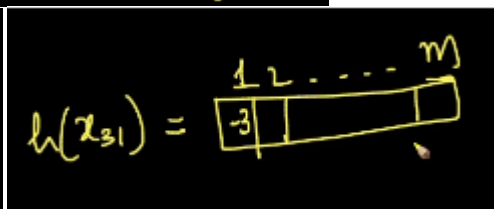
  Rest of the procedure is similar to cosine similarity process. Like finding the nearest neighbor.
  - X11 is projection of x1 on plane μ1

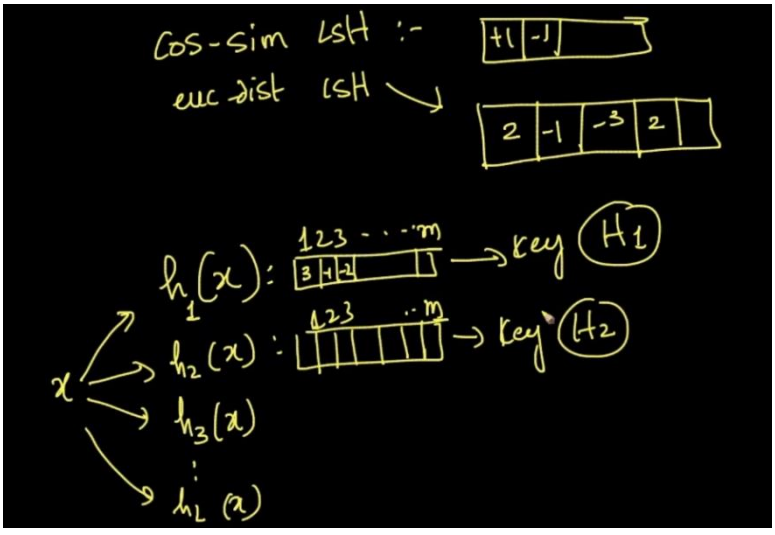


  - Now we will check in which bucket the projection will lie and will vectorize the result such that h(x11) will be the bucket number in which the point lies

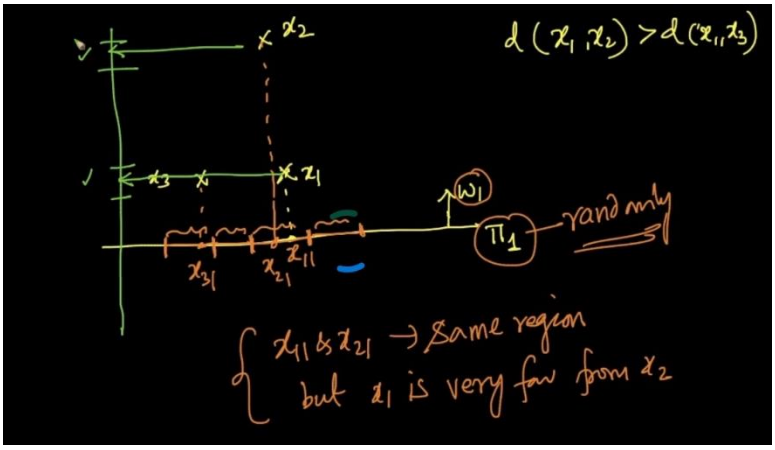








  - Since LSH is a probabilistic or randomized approach it's not perfect



- **Probabilistic class label**
  - In probabilistic approach instead of giving deterministic value, we want to establish certainty by giving probabilities that is helpful sometimes like in medical applications

ID	Actual Class Label	Probability of YES	ID	Actual Class Label	Probability of YES
1	YES	0.954	11	YES	0.521
2	YES	0.941	12	NO	0.495
3	YES	0.9	13	NO	0.456
4	YES	0.839	14	NO	0.394
5	NO	0.803	15	NO	0.313
6	YES	0.769	16	NO	0.29
7	YES	0.739	17	YES	0.223
8	NO	0.692	18	NO	0.173
9	NO	0.603	19	NO	0.13
10	YES	0.586	20	NO	0.053

  - We can combine probabilistic approach with weighted KNN as follows:
    1.query point x_q has $d1, d2, d3$ distances to the 3 NNs.
    d1 belongs to positive class & d2,d3 belongs to negative class

$$w1 = \frac{1}{d1}, w2 = \frac{1}{d2}, w3 = \frac{1}{d3}$$

$$P(x_q = \text{positive}) = \frac{w1}{w1 + w2 + w3}$$

$$P(x_q = \text{negative}) = \frac{w2 + w3}{w1 + w2 + w3}$$

    2. Now use these ratios to obtain the probability values to decide the class label.
  - This probabilistic output helps in the interpretability of the model.
  - we give x_q to the model we get y_q along with confidence
  - y_q=1 with 90% confidence ,we can trust model and take yq=1
  - if y_q=1 with 55% confidence then it is just kind of random so we cannot trust model