31.2 Imbalanced vs Balanced Dataset

Let us assume we have a binary classification problem. Let us consider a dataset 'D_n' (with 'n_1' positive points and 'n_2' negative points).

$n_1+n_2 = n$

If $n_1 \approx n_2$ (say $n_1 = 580$, $n_2 = 420$), then we can say that the given dataset is a **Balanced Dataset**.

If $n_1 << n_2$ (or) $n_2 >> n_1$ (say $n_1=100$, $n_2=900$ (or) $n_1=150, n_2=850$), then the given dataset is an **Imbalanced Dataset**.

Q) What happens when we apply K-NN on severely imbalanced data?

Ans) Let us assume we are working on a 2-class classification problem.

$n_1 \rightarrow$ Number of positive points

$n_2 \rightarrow$ Number of negative points

If $n_1 >>> n_2$, then we have a lot of positive points and a very few negative points. In case, if the negative points overlap with the positive points, then it becomes difficult to visualize, and the predictions are going to be the majority class biased for any new query point.

As K-NN is based on majority voting, almost all of the new query points are classified into the majority class.

Q) How to work around the imbalance dataset issue? (irrespective of the algorithm)

Ans)

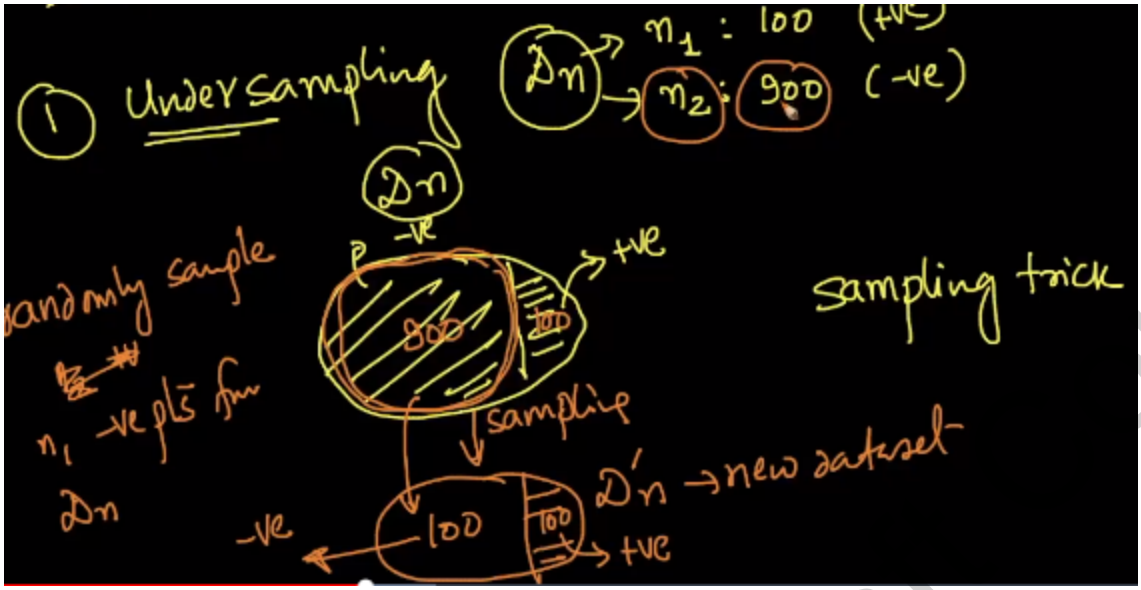
1) **<u>Undersampling</u>**

Let us consider the dataset 'D_n' where

Number of positive points (n_1) = 100

Number of negative points (n_2) = 900

Here we have the negative class as the majority class. So as per undersampling, we have to create a new dataset D_n'. All the minority class data points should be copied from D_n to D_n'. From the majority class, data points have to be randomly sampled and those randomly sampled data points should be added to D_n'. The count of the data points in the random sample should be equal to the total number of minority class data points from 'D_n'.



Now we have our dataset D_n' in balanced form, and we should use it in data modelling from here onwards instead of using 'D_n'.

Problem with Undersampling

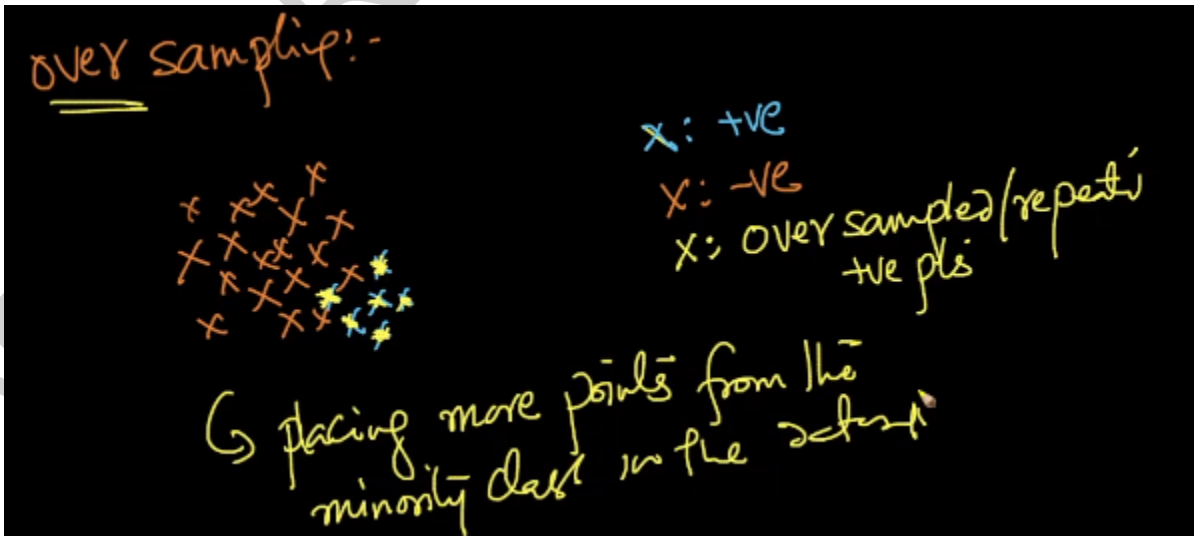
We so far had D_n=1000 and D_n'=200. Here we are removing nearly 800 data points. So data modelling is performed on a small amount of the data. As we all know that lge amount of data leads to better results, small amount of data could not give good results as we are losing lot of information.

2) **Oversampling**

Our dataset 'Dn' has 1000 data points.

Let Number of positive points (n_1) = 100

Number of negative points (n_2) = 900



In oversampling, we create another dataset D_n' with all the majority class data points, and the same number of minority class data points. The minority class data points are increased by repeating the same data points multiple times. This way, we get our dataset balanced. We now have to use D_n' for data modelling from her onwards.

There are also certain complex ways of oversampling. Repeating the same data points is a simple idea. In the complex way, instead of repeating, we create artificial/synthetic points.

One of the ways of creating the artificial/synthetic points is by taking the region of the minority class points and creating artificial points in that region. This way of creating artificial points is known as **Extrapolation**.

3) **Using the Class Weights**

Let our dataset 'D_n' have 1000 data points.

Number of positive points (n_1) = 100

Number of negative points (n_2) = 900

Now we shall take the class weights into consideration. The ratio of the class weights is inversely proportional to the counts.

$class\_weight_{+ve}$:$class\_weight_{-ve}$ = (1/1):(1/9) = 9:1

So here we consider every positive data point as 9 positive data points, and then go for data modelling. Here we are not creating new points, but are assuring each point of minority class as more points. (proportional to the majority class)

Drawback of having Imbalanced Dataset

Let us assume we have 100 positive(n_1) and 900 negative(n_2) points in our dataset 'D_n'. Here it leads to a dumb model. A dumb model is the one which blindly gives the same result. It goes blindly with the majority class. So here we can get high accuracy with imbalance data on a dumb model.

So the metric 'accuracy' shouldn't be taken into consideration, when we are working with Imbalanced data. Instead of throwing away the data (in undersampling), we should better go for oversampling, as oversampling can fix much more fundamental problems.

Q) Why do we prefer oversampling over undersampling?

Ans) The whole point of oversampling is to ensure that our model doesn't classify all the points into the majority class. Oversampling ensures that we have roughly equal number of training points from each class thereby avoiding the model from choosing the majority class as the class label for all the test data points.

Q) When is it generally better to go for undersampling?

Ans) Undersampling is typically performed when we have billions of data points and we don't have sufficient memory(RAM) resources to process the data. Undersampling may lead to worse performance as compared to training the data on full data (or) the oversampled data, in some cases. In other cases, we may not have a significant loss in performance due to undersampling.

Undersampling is mainly performed to make the training of models more manageable and feasible when working within limited memory/storage resources.

31.3 Multi-class Classification

So far we have seen binary classifiers where the output variable can contain only 2 classes. In a multi-class classification problem, we can see the output variable can contain more than 2 classes.

Example:

Let us assume we are working on a 7-NN problem, and the distribution of the nearest neighbors of a given query point 'x_q' are as follows.

0	6	1	0	.	.	.	0	**number of nearest neighbors**
1	2	3	4	.	.	.	C	**classes**

So if we have to classify a given query point 'x_q',

a) By majority vote, we get y_q' = class 2
b) By probabilistic vote, we get
   $P(y_q'=2) = 6/7$
   $P(y_q'=3) = 1/7$
   $P(y_q'=1) = P(y_q'=1) = \ldots\ldots = P(y_q'=1) = 0$

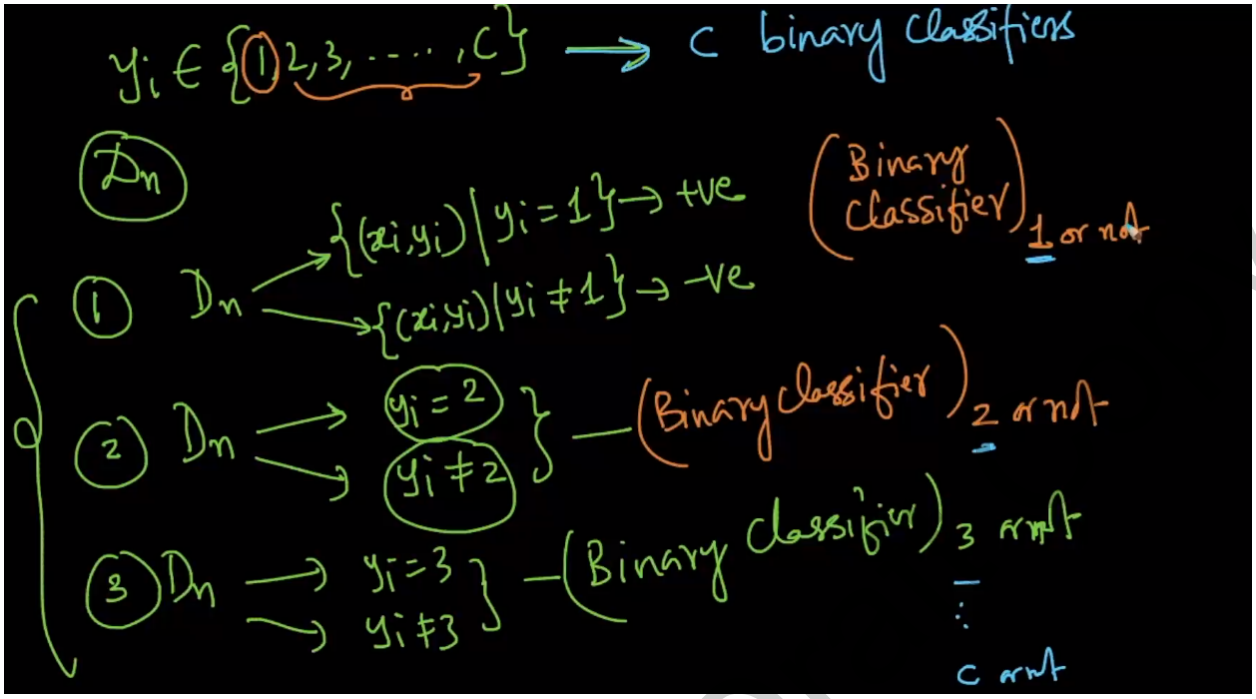
K-NN can easily be extended to multi-class classification. But there are certain classification algorithms like Logistic Regression, SVM, etc which couldn't be extended to multi-class classification as K-NN. So in case, if we are working on a multi-class classification problem, we have to convert the 'C' class classification(C>2) into 'C' binary classification problems.

Conversion of a 'C' (C>2) class classification into 'C' binary classification problems

The given dataset is represented mathematically in the form of a set as

Dn = {(xi, yi)|yi∈{1,2,3,...., C}}

So now we are splitting it into 'C' binary classification problems as shown below



So a 'C' class classification problem can create 'C' binary classifiers. This solution is called one (vs) rest approach. KNN and Naive Bayes are excluded from this approach.

Here as 'C' binary classifiers are getting created, we get 'C' number of functions the models learn to map the input to the output values.

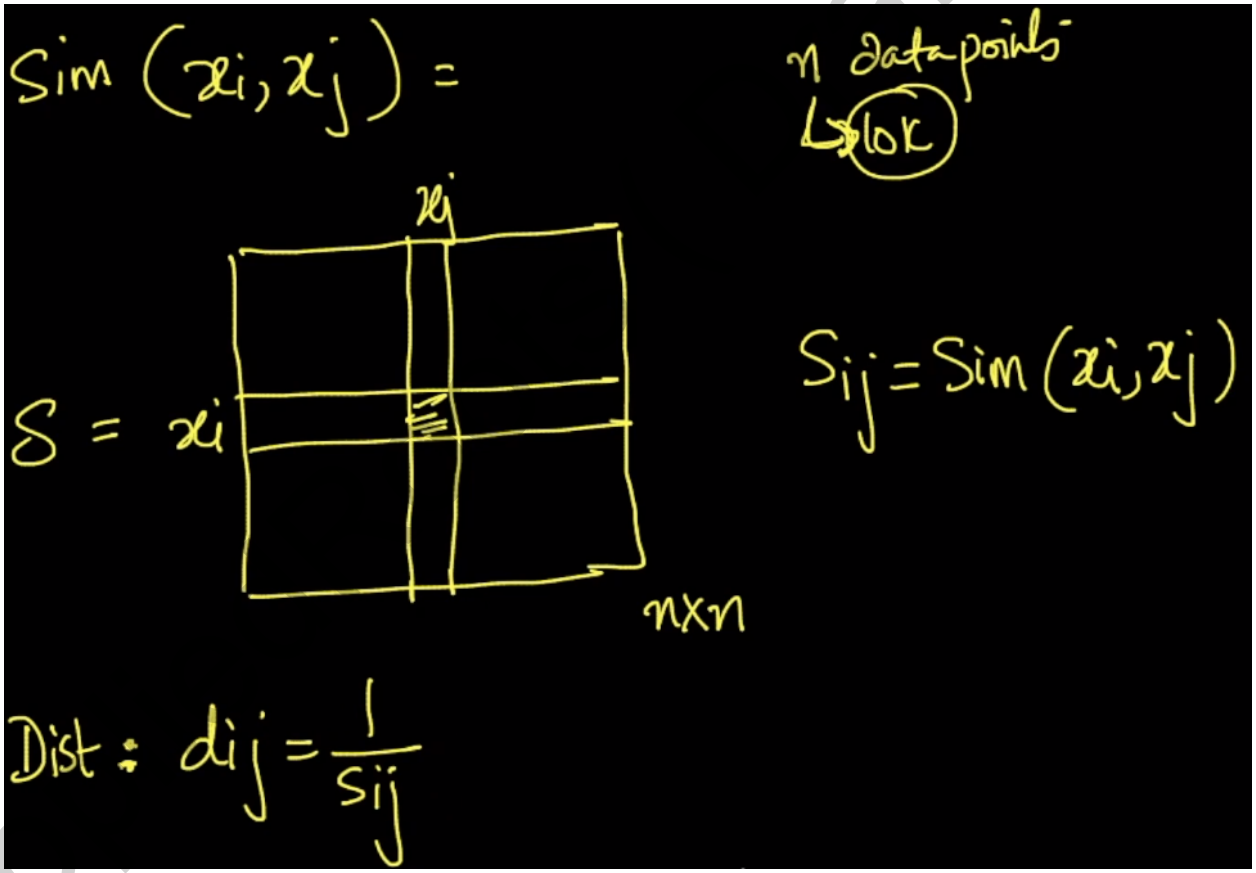
31.4 K-NN, given a Distance or Similarity Matrix

Sometimes we come across a situation where we could not represent the input 'x_i' in a vector format (or) conversion of text to vector is not possible.

Let us assume we are working in the Pharmacy domain. Let 'x_i' be a chemical compound. Conversion of the input of this compound into a vector is more difficult.

A pharmacist/domain expert couldn't convert a chemical compound into a vector format, but can give us the similarity scores between the pairs of compounds. So sometimes we do not get the input data in a vector form, but can get a similarity matrix as an input.

If we have 'n' data points in the training dataset, then the similarity matrix consists of 'n' rows and 'n' columns. The element at position (i,j) in the similarity matrix denotes the similarity score between the i^{th} and j^{th} compounds. The Similarity matrix is represented as shown below



The Similarity matrix is denoted by 'S' and has a shape nXn. An element 'S_{ij}' in the similarity matrix 'S' denotes the similarity score between the i^{th} and j^{th} compounds.

$S_{ij} = Sim(x_i, x_j)$

Once we have the similarity matrix 'S', we can convert it into a distance matrix.

$D \rightarrow d_{ij} = 1/S_{ij}$

Given a similarity matrix or a distance matrix, instead of $x_i \in R^d$ explicitly, we can easily apply K-NN, as K-NN cares only about the distances, but not about representation of input.

Techniques like Logistic Regression do not work as easy as K-NN, when the Similarity Matrix (or) Distance matrix is given, instead of vector representation of the input.

Note: For a new query point(in test dataset), the domain expert could generate the similarity scores between a query chemical compound and the existing compounds in the training dataset, using the formulae/rules/processes of Chemistry.

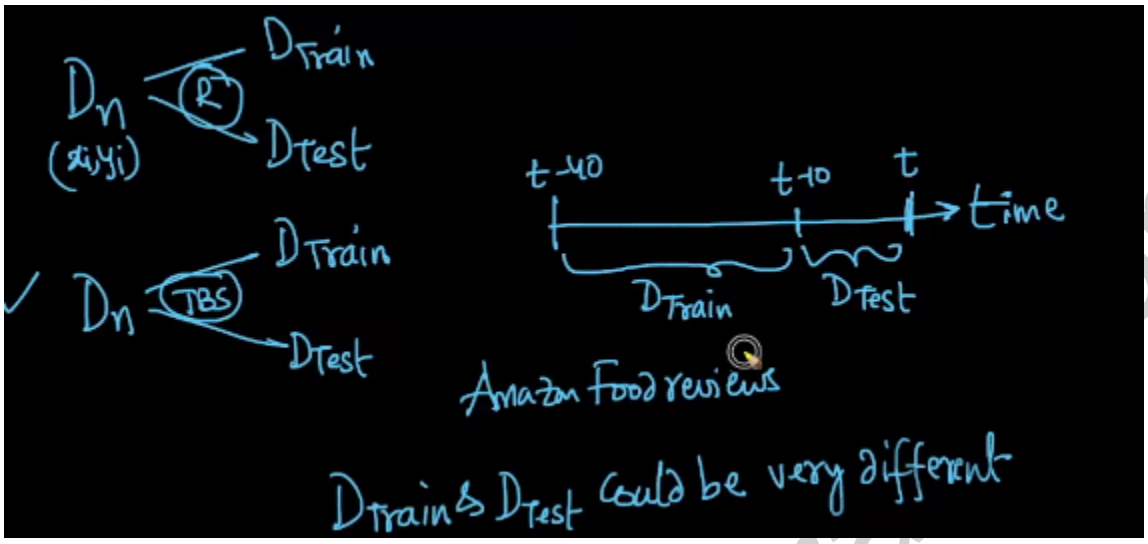
Once the domain expert has the similarity scores, K-NN is one simple strategy that he may use and is very common. Often times, computing the similarity matrix is a hard part.

Q) Is the similarity matrix the same as the correlation matrix?

Ans) Let us assume we have a dataset with 'n' data points and 'd' dimensions. Then the correlation matrix is of the order dXd. Correlation matrix tells us how much correlated each feature is with the other feature.

Whereas in the similarity matrix, each cell 'S_{ij}' tells us about the similarity between the i^{th} and the j^{th} compounds. The similarity matrix is of the order nXn.

31.5 Train and Test Set Differences



So far we have seen a dataset (D) being split into D_{Train} and $_{DTest}$ using random sampling where each data point has an equal chance of being selected into the sample.

But here a problem arises when we have time based splitting of the data. This problem occurs rarely in random splitting, but frequently in time based splitting.

The problem here is we get the data of new categories added in the test data over time. These categories might not be present in the training data. Also old category products that are not making good business might be removed from the training data. The underlying data changes over time and we have to be very careful, if we are applying time based splitting on the changing data.

Due to addition of the new category data and removal of some old categories data, the K-NN decision surface and the shape of the distribution also changes. The decision surface created using 'D_{Train}' will work good for 'D_{Train}', but not for 'D_{Test}'. The cross-validation error will be low, but the test error will be high. The distribution of test data has changed entirely to a new different region. In this case, we have to perform a check (ie., checking whether 'D_{Train}' and 'D_{Test}' have the same distribution or not)

How to check if D_{Train} and D_{Test} have the same distribution?

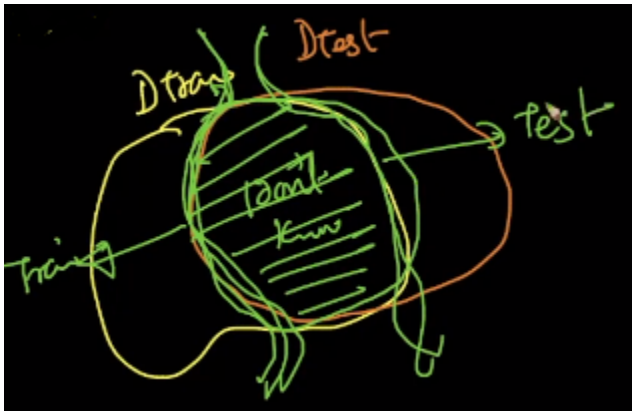
Let us assume we have a dataset 'D_n'. We split it into 'D_{Train}' and 'D_{Test}' using Time Based Splitting. Both 'D_{Train}' and 'D_{Test}' consist of both positive and negative points.

We shall now create a new dataset D_n'. For every data point (x_i, y_i) in both 'D_{Train}' and 'D_{Test}', we should create (x_i', y_i').

y_i' = 1 ; x_i' = concat(x_i, y_i) in 'D_{Train}'

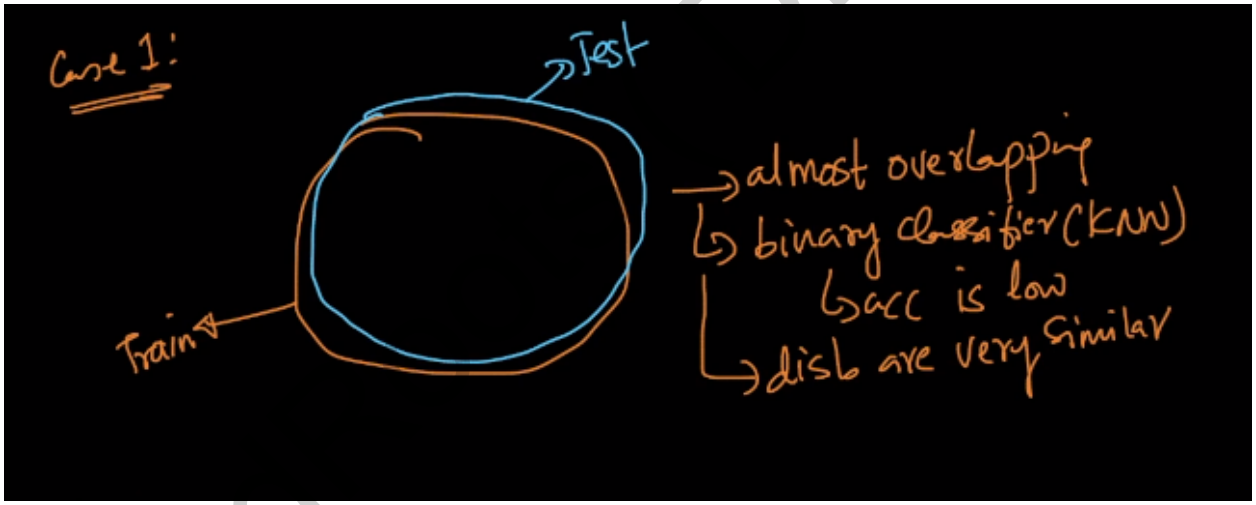
y_i' = 0 ; x_i' = concat(x_i, y_i) in 'D_{Test}'

We shall now build a binary classifier on D_n'. Let f(x) be the function that predicts the label as 0 or 1. Let the distribution of the points might be like this.



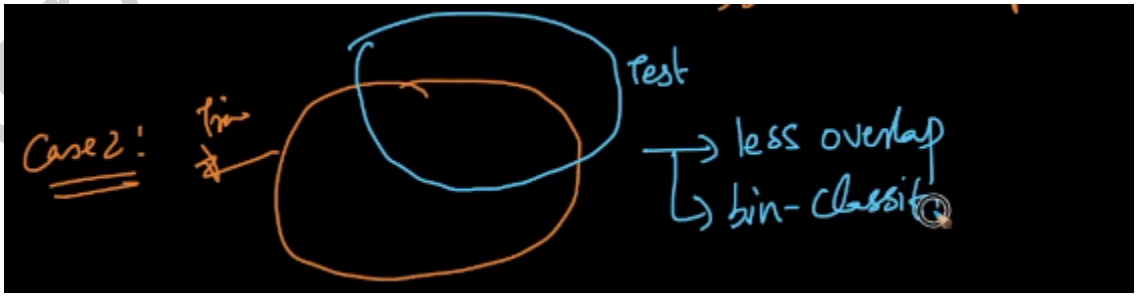
The binary classifier will be able to separate 'D_{Train}' and 'D_{Test}' to some extent. If this binary classifier gives an accuracy of 70%(say), it means in 70% of the cases, we are able to separate 'D_{Train}' and 'D_{Test}'.

Case 1: (Best Case)



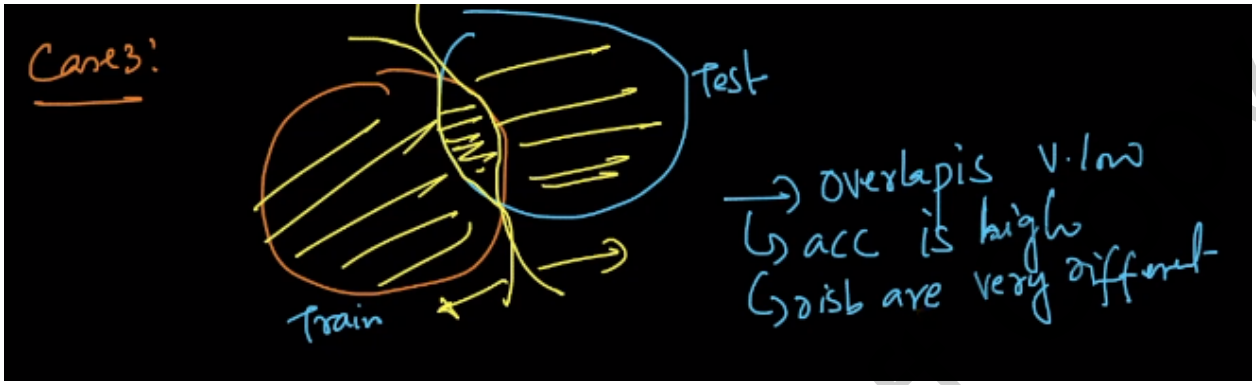
Here we have the distributions almost overlapping. Such cases lead to low accuracy. It means both the 'D_{Train}' and 'D_{Test}' distributions are similar.

Case 2: (Average Case)



Here we have less overlap. This case leads to medium accuracy. It means the distributions of 'D_{Train}' and 'D_{Test}' are similar.

Case 3: (Worst Case)



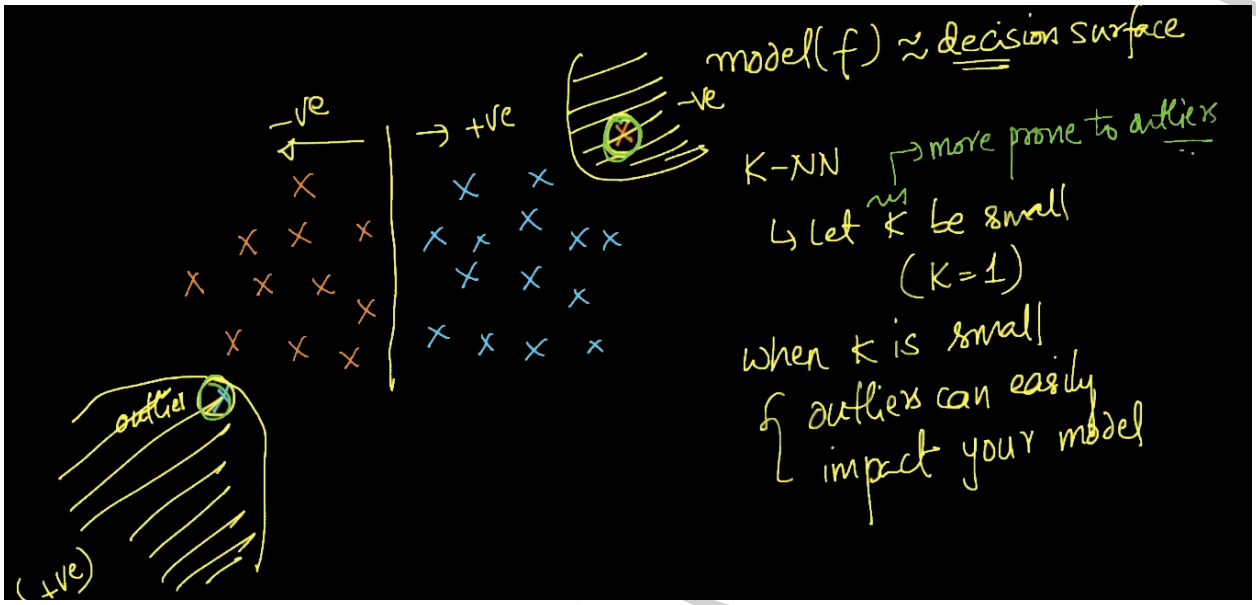
Here we have least overlap. This case leads to high accuracy, and the distributions of the 'D_{Train}' and 'D_{Test}' are different, and K-NN can perform the classification task very well.

Note: If 'D_{Train}' and 'D_{Test}' are from the same distribution, then the ML algorithms work perfectly well.

In case, if 'D_{Train}' and 'D_{Test}' are from different distributions, then no ML algorithm can perform well. So in such a case, it means the features chosen are changing over time (or) features are not temporally stable. In this case, we need to change/design/build new features.

31.6 Impact of Outliers

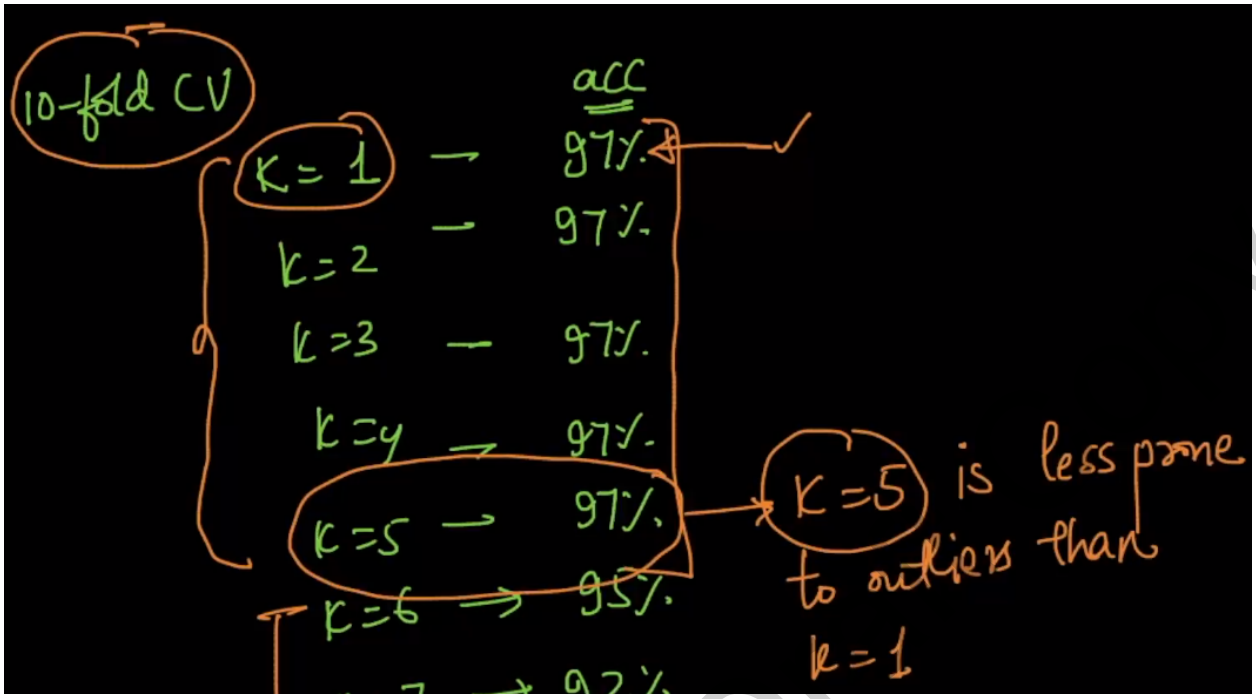
For every classification problem, we need to understand the impact of outliers on the model. A model can easily be understood by its decision surface, because the decision surface is the geometrical surface that separates the data points belonging to different classes.



Let us assume we are applying the K-NN model with K=1. In the left side region, we have a +ve point present in the region with -ve points. This point is an outlier. So if any query point falls nearer to this outlier, then by considering this 1 nearest neighbor(ie., the outlier), the query point will be classified as +ve, even though it is present in the region with the -ve class points in majority.

Similarly on the top right, we could see a -ve point falling in the region with the +ve class points in majority. Even here a new query point falls nearer to this negative point, by taking the 1-NN into consideration, that query point will get classified as -ve, even though it is present in the region with the +ve class points in majority.

So when the ‘K’ value is small, the outliers can easily impact our model. When the ‘K’ value is small, the model tends to be more prone(easily affected) by the outliers.



Let us assume we are performing 10-fold cross-validation for different values of 'K' (here it is 'K' in K-NN), then let us assume that we got the best accuracy score of 97% for K=1,2,3,4,5,. In such a case, we have to choose K=5, as the optimal hyperparameter value because the model with K=1 is more prone to the outliers, whereas the K-NN model with K=5, is less prone to the outliers. It means the model with K=5 is less affected by the outliers.

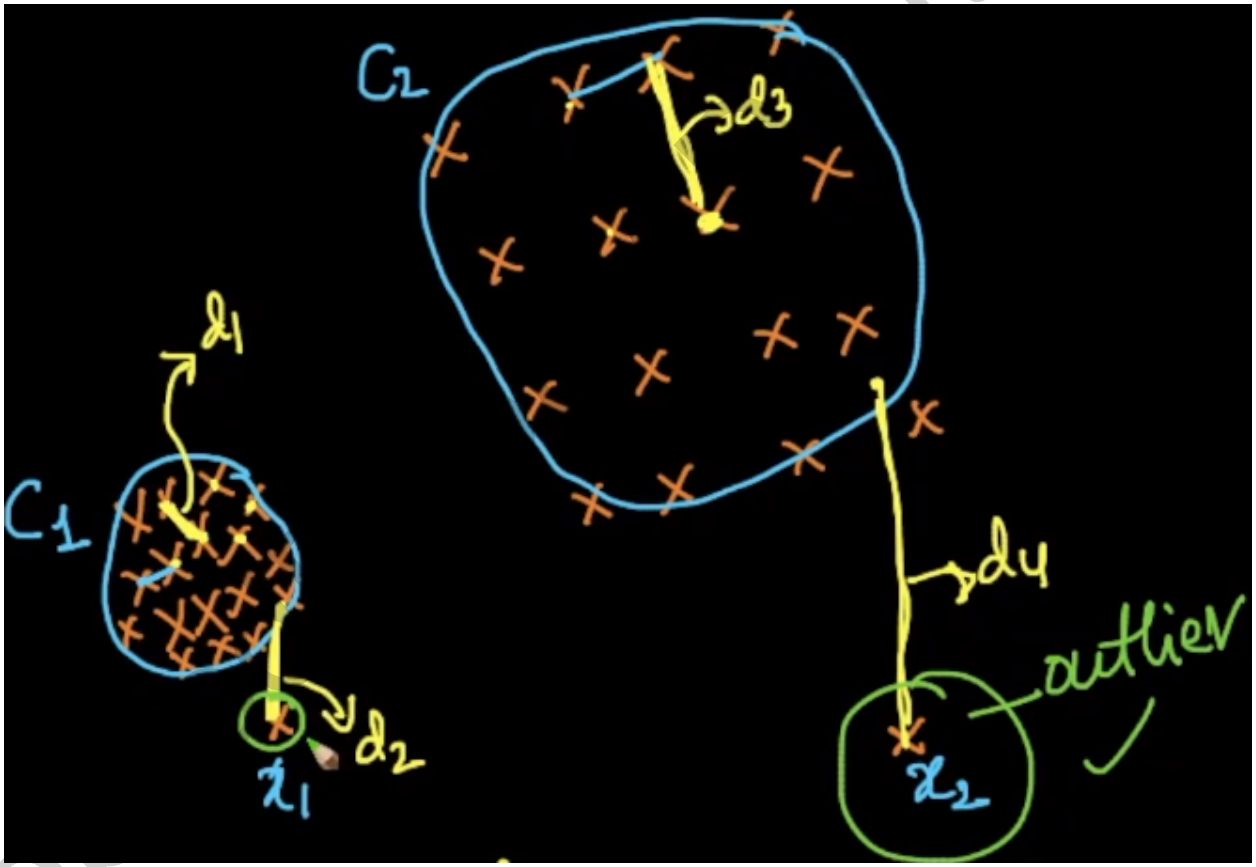
In the next set of lectures, we are going to discuss a topic called 'Local Outlier Factor' which is used to remove the outliers from our dataset.

31.7 Local Outlier Factor (Simple Solution: Mean distance to KNN)

Local Outlier Factor(LOF) is a technique used to remove the outliers from the dataset. It is inspired by K-NN.

Example

Let us consider a binary classification setting with +ve and -ve classes. We shall focus only on the -ve class points. These points are divided into two clusters 'C_1' and 'C_2' ('C_1' being a dense cluster and 'C_2' being a sparse cluster). 'C_1' and 'C_2' clusters contain only -ve points as of now. We have to first process all the -ve points and find out the outliers among them, and remove them from both the classes. Let us assume K=5, in K-NN.



Simple Solution

1) For every point 'x_i', compute its 'K'(here K=5) nearest neighbors.
2) Compute the average of the distances of 'x_i' to all its 5 nearest neighbors. (ie., avg(d_1, d_2, d_3, d_4, d_5) where 'd_p' is the distance of the point 'x_p' to 'x_i').

3) Sort all the points x_i's by their average distances in ascending order.
If the average distance of a point 'x_1' is very high when compared to the other average distances, then we declare the point 'x_1' as an outlier.

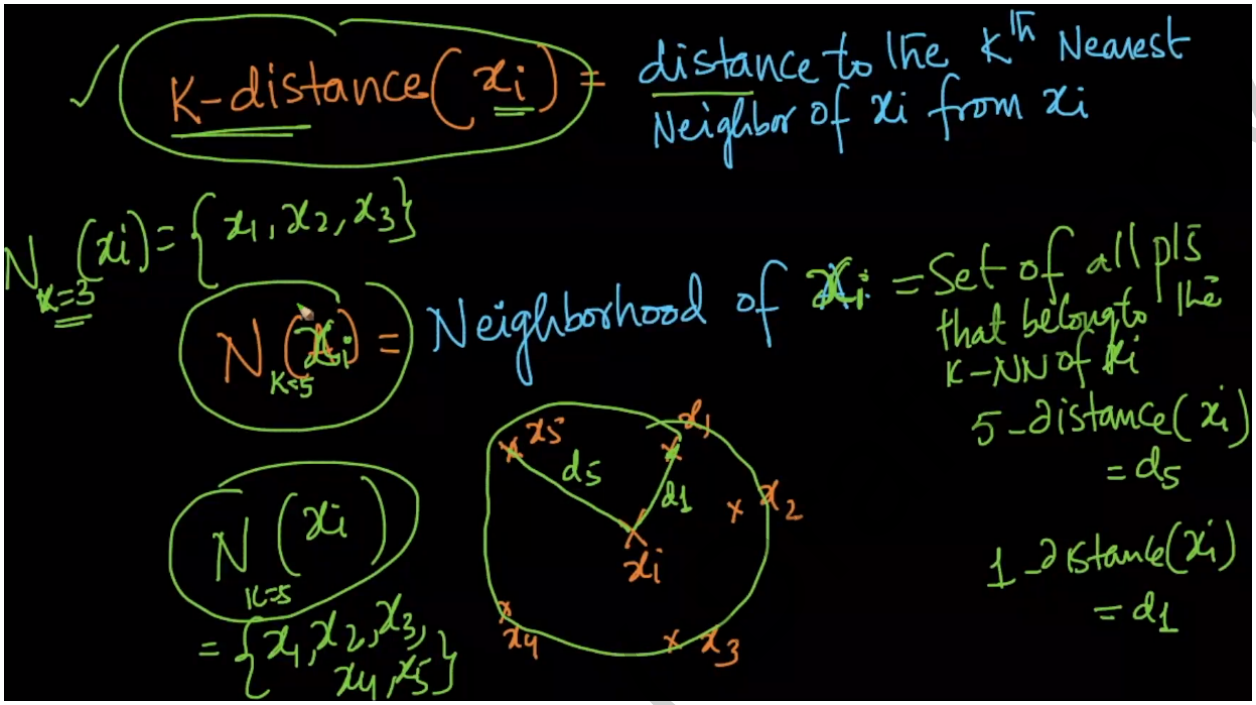
But this approach fails if we consider 'x_1' as an outlier because the distance of the point 'x_1' to its neighbors in the cluster 'C_1' is almost equal to the distance between the points in the cluster 'C_2', as 'C_2' is a sparse cluster, and the distances between the points in the cluster 'C_2' is high when compared to the distances between the points in the cluster 'C_1'. Hence this approach fails in detecting the outliers, as it doesn't take the density of the points into consideration.

If we had all the points in denser form, and there are no sparse clusters, then this approach works well in detecting the outliers. But in the real world, it is not guaranteed that the data we get always is present in denser form. The technique used to detect the outliers should be able to do its job in both the simple and complex cases perfectly.

Hence while declaring a point as an outlier, we also should take the local density into consideration. We have discussed another approach for LOF in the next set of lectures, which also takes the local density into consideration at the time of detecting the outliers.

31.8 K-Distance

K-Distance of a point 'x_i' is the distance of the point 'x_i' to its K^{th} nearest neighbor.



In the above example,

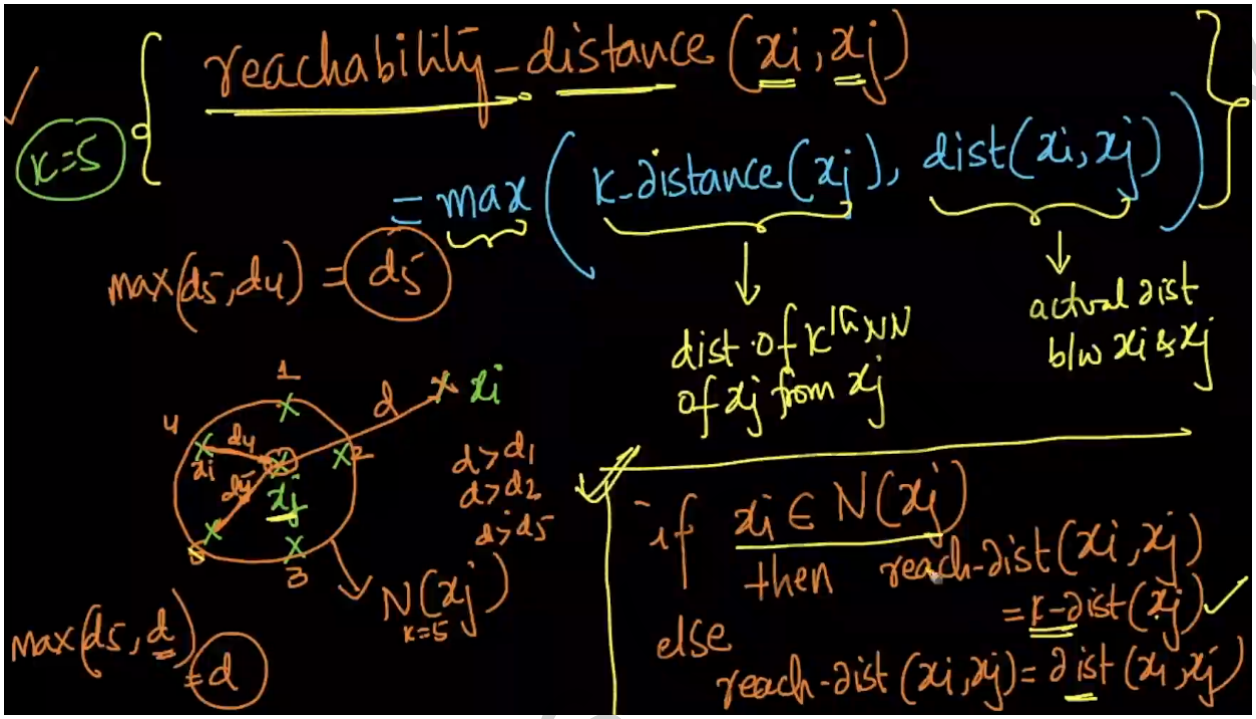
a) If K=5, then the distance of the point 'x_i' to 'x_5' is the K-distance of 'x_i'.
b) If K=3, then the distance of the point 'x_i' to 'x_3' is the K-distance of 'x_i'.

Notation:

K-distance(x_i) → Distance of Kth nearest neighbor of 'x_i' from 'x_i'.
$N(x_i)$ → Neighborhood of 'x_i'
For example,
$N_{K=5}(x_i)$ → Set of all the 5 nearest neighbors of 'x_i'. (ie., $\{x_1, x_2, x_3, x_4, x_5\}$)
$N_{K=3}(x_i)$ → Set of all the 3 nearest neighbors of 'x_i'. (ie., $\{x_1, x_2, x_3\}$)

31.9 Reachability-Distance(A,B)

Reachability distance between the two given points 'x_i' and 'x_j' is given as the maximum among k-distance(x_j) and distance between the points 'x_i' and 'x_j'.



Reachability Distance(x_i,x_j) = max(K-Distance(x_j), distance(x_i,x_j))

Let us assume we have two given points 'x_i' and 'x_j'.

1) If $x_i \in N(x_j)$, then it means 'x_i' is one of the 'K' nearest neighbors of 'x_j'.In such a case,

   a) If 'x_i' is the K^{th} nearest neighbor (ie.,if K=5, then 'x_i' is the 5^{th} nearest neighbor), then

      K-Distance(x_j) = distance(x_i,x_j)

      Reachability Distance(x_i,x_j) = K-Distance(x_j) = distance(x_i,x_j)

   b) If 'x_i' is not the K^{th} nearest neighbor, (ie.,if K=5, then 'x_i' is not the 5^{th} nearest neighbor), then

      K-Distance(x_j) > distance(x_i,x_j)

      Reachability Distance(x_i,x_j) = K-Distance(x_j)

2) If $x_i \notin N(x_j)$, then it means 'x_i' is not one among the 'K' nearest neighbors of 'x_j' (ie., if K=5 then 'x_i' is not one among the 5 nearest neighbors of 'x_j'), then,

   distance(x_i,x_j)>K-Distance(x_j)

   Reachability Distance(x_i,x_j) = distance(x_i,x_j)

31.10 Local Reachability Density (A)

The Local Reachability Density of a point 'xi' is given as

$LRD(x_i) = 1/((\Sigma_{xj \in N(xi)}$ Reachability-Distance(x_i, x_j))/|$N(x_i)$|)

|$N(x_i)$| → Number of points in the neighborhood of the point 'x_i'.

Local Reachability Density of a point 'x_i' is defined as the inverse of the average reachability distances of the point 'x_i' from all the points in its neighborhood.

The denominator term in the formula denotes the average reachability distance of the point 'x_i' from all the points in its neighborhood.

The same above given formula can be written as

$LRD(x_i) = |N(x_i)|/(\Sigma_{xj \in N(xi)}$ Reachability-Distance(x_i, x_j))

This equation is now in the form of (**Number of points/measure of distance)**, which is the standard form of a density.

Local → Because we are looking only the neighborhood

Reachability → Because we are considering only the reachability distances

Density → As the formula is in the form of (**Number of points/measure of distance)**

Hence we got the name **Local Reachability Density**.

31.11 Local Outlier Factor (A)

Local Outlier Factor (LOF) is a technique used to detect the outliers in the given data. The LOF is given by the formula as mentioned below

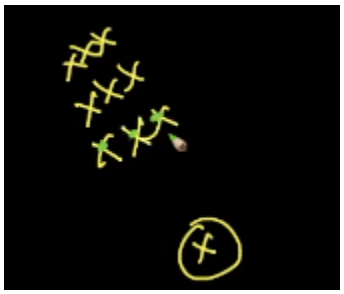
$LOF(x_i) = ((\Sigma_{xj \in N(xi)}\ LRD(x_j))/|N(x_i)|) * (1/LRD(x_i))$

Here we can see the LOF value is a product of two terms. The first term denotes the average LRD of points in the neighborhood of 'x_i'.

$LOF(x_i)$ is large when **$LRD(x_i)$ is small** (or) **average LRD of the points in $N(x_i)$ is large**.
$LOF(x_i)$ is small when **$LRD(x_i)$ is large** (or) **average LRD of the points in $N(x_i)$ is small**.

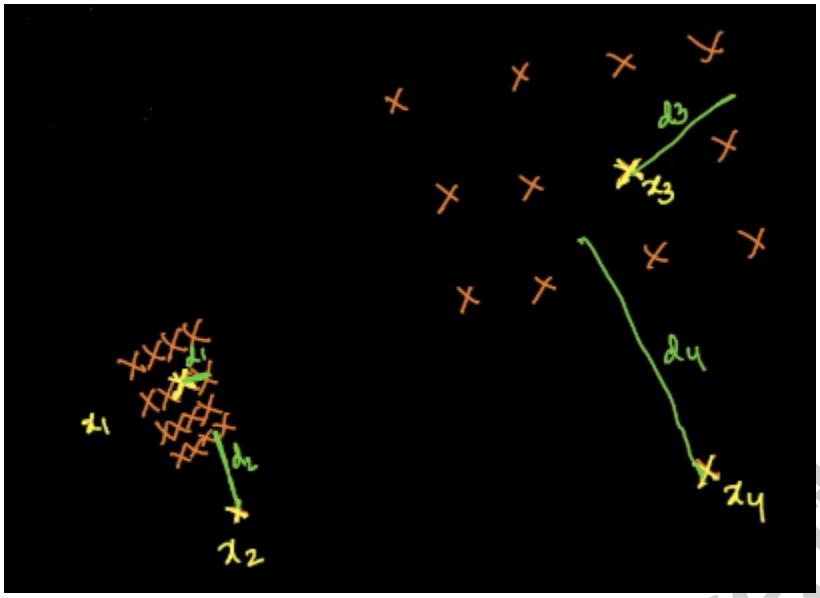
If **LOF(xi) is large**, then we call it an **Outlier**.
If **LOF(xi) is small**, then we call it an **Inlier**.

So we can say a point is an outlier if the density around that point is low when compared to the density around its neighborhood.



In the above points, the point circled is our query point, and its K-nearest neighbors are present in the given cluster of points. Here we see the density around the K-nearest neighbors of our query point is so high, but the density around our query point is low. Hence LOF concluded this query point as an outlier.

Let us now look at another example.



$d_1 \rightarrow$ Average Reachability Distance of 'x_1'
$d_2 \rightarrow$ Average Reachability Distance of 'x_2'
$d_3 \rightarrow$ Average Reachability Distance of 'x_3'
$d_4 \rightarrow$ Average Reachability Distance of 'x_4'
From the figure, we can say that $d_1<d_2<d_3<d_4$

Local Reachability Density $\propto$ 1/(Average Reachability Distance)
LSD $\propto$ 1/d

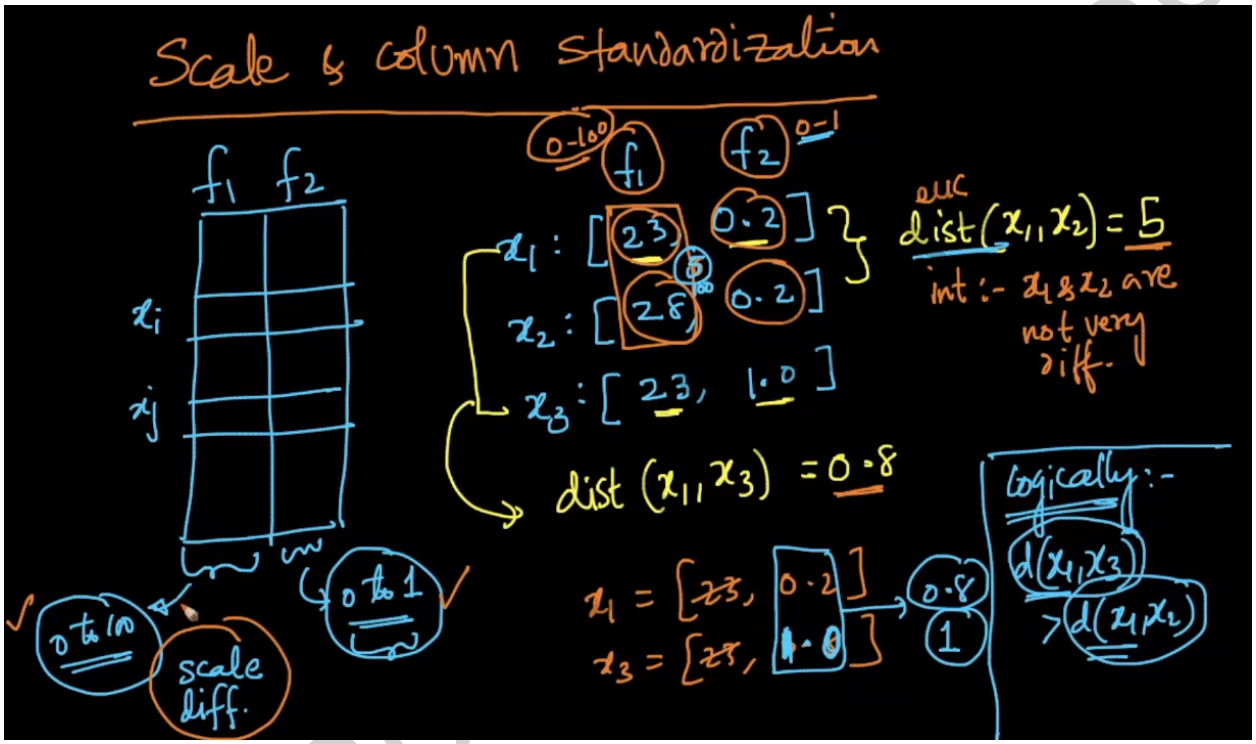
So as $d_1<d_2<d_3<d_4 \rightarrow 1/d_1>1/d_2>1/d_3>1/d_4 \rightarrow LRD(x_1) > LRD(x_2) > LRD(x_3) > LRD(x_4)$
From the above distribution of points,
$LRD(x_1) \approx LRD(x_j \in N(x_1))$
$LRD(x_2) \approx LRD(x_j \in N(x_2))$
$LRD(x_3) \approx LRD(x_j \in N(x_3))$
$LRD(x_4) \approx LRD(x_j \in N(x_4))$

The densities around the points 'x_1' and 'x_3' is almost same as the densities around $N(x_1)$ and $N(x_3)$ respectively, as 'x_1', $N(x_1)$ and 'x_3', $N(x_3)$ lie in the same clusters. The densities around the points 'x_2' and 'x_4' are less than the densities around $N(x_2)$ and $N(x_4)$ respectively, as 'x_2' and 'x_4' are away from the clusters that contain $N(x_2)$ and $N(x_4)$ respectively.
So for
$x_1, x_3 \rightarrow$ LOF is small
$x_2, x_4 \rightarrow$ LOF is large

31.12 Impact of Scale and Column Normalization

Let us assume we have two features 'f_1' and 'f_2'. Let us assume the values of 'f_1' lie in the range 0-100 and the values of 'f_2' lie in the range 0-1. Let us consider the below given points as an example

	f_1	f_2
x_1:	23	0.2
x_2:	28	0.2
x_3:	23	1



When we consider the pair (x_1, x_3), we see the values of 'f_1' are the same. Only the values of 'f_2' differ.

So Distance(x_1, x_3) = 1 - 0.2 = 0.8 (min-max range of 'f_2' is 0-1). Here the maximum value is 1 and the difference value is 0.8, which is 80% of the maximum value.

When we consider the pair (x_1, x_2), we see the values of 'f_2' are the same. Only the values of 'f_1' differ.

So Distance(x_1, x_2) = 28 - 23 = 5 (min-max range of 'f_1' is 1-100). Here the maximum value is 100 and the difference value is 5, which is 5% of the maximum value.

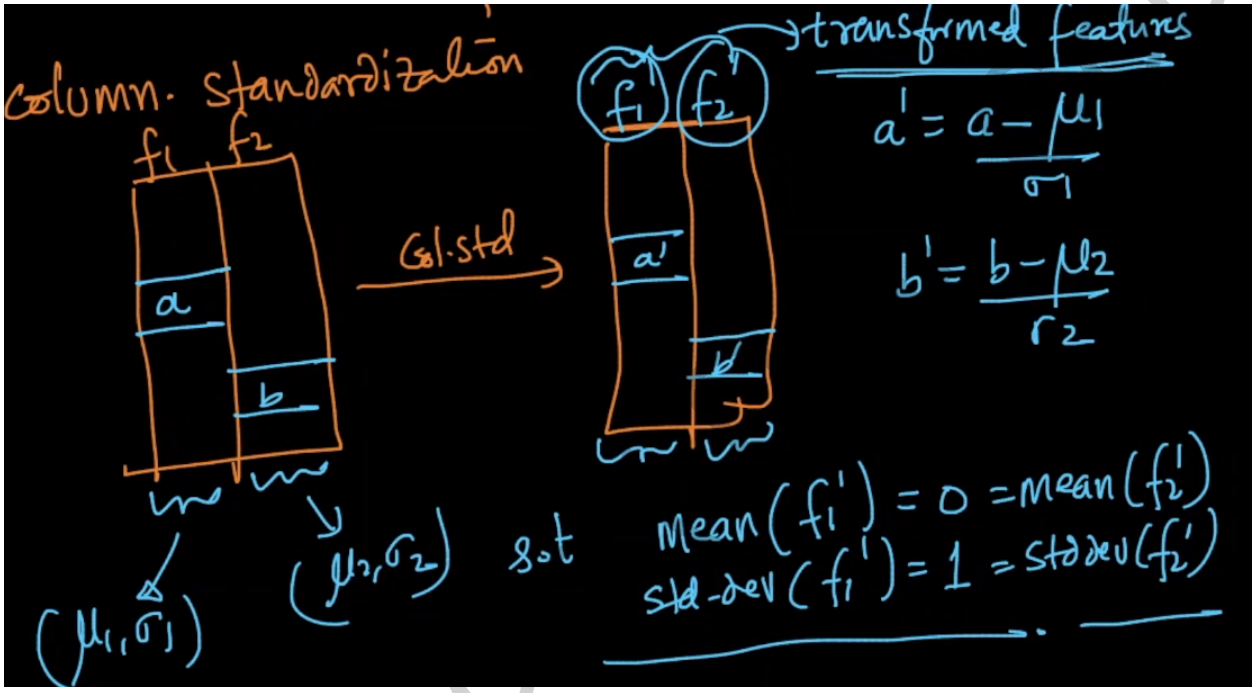
When we look into these differences, we can say **logically Distance(x_1,x_3)> Distance(x_1,x_2)**, but **mathematically Distance(x_1,x_2)>Distance(x_1,x_3)**.

So these mathematical and logical variations are due to change in the scales of the features. With such features, if we go ahead with model building, then definitely that

would mess up our models. Hence in order to get rid of this problem, we apply Column Standardization.

If we apply Column Standardization, all the features/columns will fall almost into the same range(not exactly the same range), and also the nature of the distribution of the features doesn't change at all.

Column Standardization is a technique which transforms our features in the dataset to a new set of features with each column having a mean '0' and standard deviation '1'.



If $\mathbf{f_1}$ and $\mathbf{f_2}$ are our original features, let us assume $\mathbf{f_1'}$, $\mathbf{f_2'}$ are the transformed features, $\boldsymbol{\mu_1}$, $\boldsymbol{\mu_2}$ are the means of $\mathbf{f_1}$, $\mathbf{f_2}$ and $\boldsymbol{\sigma_1}$, $\boldsymbol{\sigma_2}$ are the standard deviations of $\mathbf{f_1}$ and $\mathbf{f_2}$ respectively. Then

$\mathbf{f_1' = (f_1 - \mu_1)/\sigma_1}$

$\mathbf{f_2' = (f_2 - \mu_2)/\sigma_2}$

Since Euclidean distance can easily be impacted by the differences in the scales, it is important to perform column standardization. ML techniques like K-NN, Logistic Regression, SVM are scale dependent, whereas the techniques like Naive Bayes, Decision Trees, Ensemble models are scale independent.

Note: Standardization doesn't ensure that all the resulting values are in a fixed interval. It only removes the scale effect and ensures the mean = 0 and standard deviation = 1.

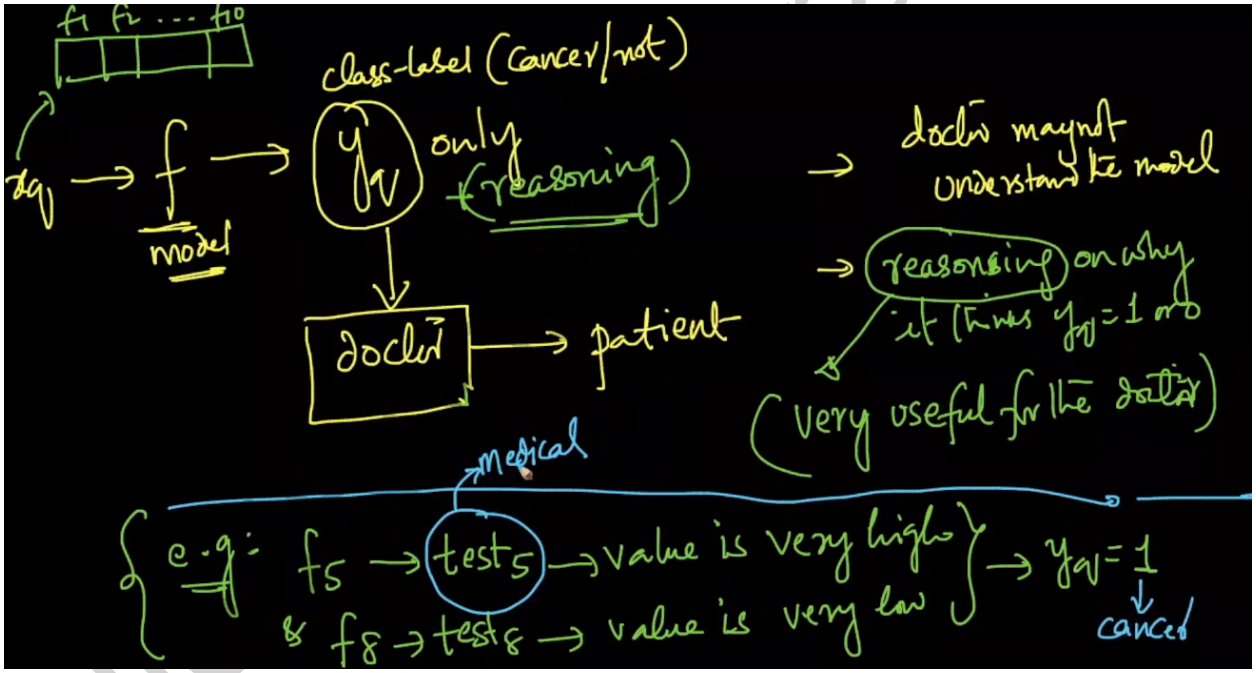
31.13 Interpretability (Model Interpretability vs Blackbox)

If a model just predicts the result without giving any reasoning, then we call it a **Blackbox model**.

If a model gives reasoning and explains the reason for the occurrence of a value as an output, such a model is called an **Interpretable model**.

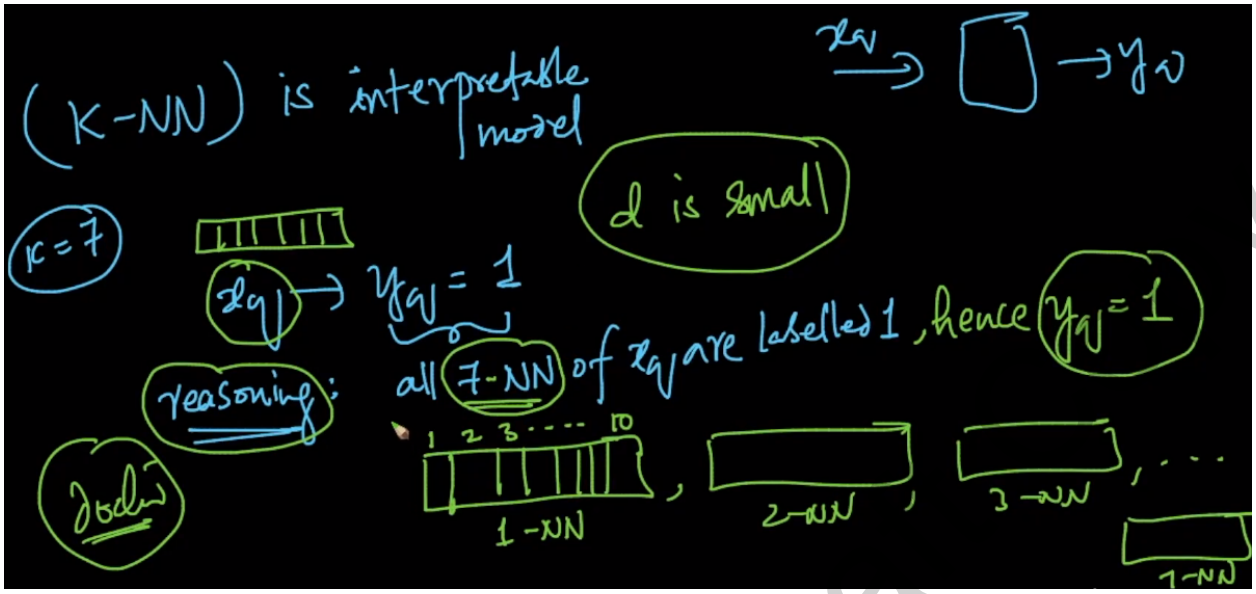
Example

Let us take an example from the medical domain. We have to build a model that takes the patient data as input and predicts whether the patient has cancer or not.
The ML model predicts the class label and all the times the model may not give an accurate prediction. Sometimes it goes wrong as well. Also making a decision, without knowing what exactly is happening is also not correct. In order to confirm whether a patient has cancer or not, the patient has to go through some tests, and depending on the results of those tests, the doctor can make a confirmation.



So we pass the results of those prior tests as the input data to our model. Without knowing the information about the data and which features are contributing more in our predictions, even though the model gives the result, still it could not be considered sometimes. Hence it is required to know what all features are contributing more to our model at prediction, and what all features are contributing less, is also very important. Making interpretations will add huge weightage and confidence to our analysis and the results. Such models which can also give interpretation along with the predictions are called **Interpretable models** and the models which could not give interpretations, but can only give the predictions are called **Blackbox models**.

Is K-NN an interpretable model?



If we are working on a K-NN problem (with K=7), then let us assume all the 7 patients have the class label y=1 (it means all of them are diagnosed as Cancer). In such a case, by taking all the 7 nearest neighbors into consideration, we can confirm that the patient is also having cancer. (It is because those 7 neighbors picked are nearest to our query patient. In such a case, we do not find much difference in the data vectors of these 7 patients. The prior test results are similar in all these 7 patients). Hence we can confidently confirm that the given patient is having cancer.

K-NN is interpretable when 'd' and 'K' are small. (It differs from domain to domain). It is because when 'd' is small, it is easier to check each of the features to understand what is happening amongst the nearest neighbors, and understand the reason behind the decision.

When 'K' is small, it is easier to go through each of the data points individually, making the interpretation easier. For example, if d=100 and K=21, the need to go through 100*21 feature values to understand the reasoning behind the decision is sometimes not possible for human beings.

31.14 Feature Importance and Forward Selection

Feature Importance is the process of sorting the features in the order of importance for the classification/regression task.

Feature Importance is useful in understanding a model better. Once we understand a model better, the model interpretability increases.

For example, we are predicting the height of a person (which is a regression task). The given features in our data are weight, hair color, hair length, skin color, gender, country, etc. Let us assume, the features weight, gender, country and skin color are playing a major role in deciding the height value, then we can say these are the important features. These important features add more weightage to the model at the time of prediction.

Q) Is there a way to get the feature importance directly using K-NN?

Ans) K-NN couldn't give the feature importance easily. ML models like Logistic Regression, Decision Trees, Linear SVM, Naive Bayes, etc can give the feature importance easily.

There are certain hacks that are used to obtain the feature importance for any model. Thes hacks are model independent. It means these techniques/hacks can be applied on any ML model.

Feature Selection is the process of discarding the least important features from the model, and keeping the more important feature only. One of the techniques of Feature Selection is **Forward Feature Selection**.

Procedure of Forward Feature Selection

1) Take the dataset 'D_n^d' (say d=10).
2) Fit a model(say K-NN) with only one dimension at a time. This wway, we get 10 models. Choose the one which gives the highest accuracy. Let's say the model built using only 'f_2' gives the highest accuracy.
3) Now keep the feature 'f_2' in our model, and try fitting the model including the other features also, but only one at a time. (ie., (f_2, f_1), (f_2, f_3), (f_2, f_4), (f_2, f_5), (f_2, f_6), (f_2, f_7), (f_2, f_8), (f_2, f_9), (f_2, f_{10})). Pick the combination whichever gives the highest accuracy.
4) Let us assume we got 'f_{10}' as the next important feature, then we say we have two important features 'f_2' and 'f_{10}'.
5) Now we have the two important features 'f_2' and 'f_{10}', we have to try combinations with keeping these 2 features and adding one feature at a time as the third feature. (ie., (f_2, f_{10}, f_1), (f_2, f_{10}, f_3), (f_2, f_{10}, f_4), (f_2, f_{10}, f_5), (f_2, f_{10}, f_6), (f_2, f_{10}, f_7), (f_2, f_{10}, f_8), (f_2, f_{10}, f_9)). Let us assume the model with the combination (f_2, f_{10}, f_5) gives the highest accuracy. Then 'f_5' is selected as the 3rd important feature. Next we have

to try fitting the models with 4 features, keeping 'f_2', 'f_5' and 'f_{10}' constant in the model.

This way, we have to continue the process. At each stage, given that we already have some features, we have to check which new feature adds the most value to our model. These iterations are continued till we don't find any improvement in the model accuracy, even after adding new features.

Time Complexity

At each iteration, we are training and testing d-models.

1st iteration → 'd' models

2nd iteration → 'd-1' models

3rd iteration → 'd-2' models

The time complexity is very high in Feature Selection Techniques. They do not care about the model we are building. (ie., type of the algorithm used)

The advantage of Feature Selection techniques is they are independent of the type of ML algorithm used, and the disadvantage is the time complexity.

31.15 Handling Categorical and Numerical Features

Let us assume we are working on the problem of predicting the height of a given individual. If a given feature consists of numerical values, it can be taken directly into building a model. If it consists of categorical values, then it has to be converted into numerical form, and then should be taken into building a model.

Let us assume our features are weight, hair color, country, hair length, gender, etc. The weight feature consists of only numerical values, so it can be taken directly into building a model.

The hair-color feature consists of categorical features. Let those values be Hair-color = {'black', 'brown', 'red', 'golden', 'gray'}. As these values cannot be directly used in model building, we have to convert them into numerical form.

Ways to handle these categorical features

1) **Giving a number to each value**

   Let us assign number to these categorical values as shown below
   'Black' = 1, 'Brown' = 2, 'Red' = 3, 'Golden' = 4, 'Gray' = 5
   When we give such numerical values, we see there is an ordering among the values. (ie., 3>1, 5>1 which mean 'Red'>'Black' and 'Gray'>'Black). This is totally absurd and all these values here are discrete. Logically there doesn't exist any ordering among these categorical features. Hence this approach is not at all recommended.

2) **One-Hot Encoding**

   One-Hot Encoding creates a binary vector of size equal to the distant number of elements. As we have 5 values of hair-color, we get a binary vector of 5-dimensions.
   Black: 10000
   Brown: 01000
   Red: 00100
   Golden: 00010
   Gray: 00001

   In case of one-hot encoding, only one bit will be set to 1, and the rest all bits are set to 0. Hence we get a sparse and large vector, if the number of distinct values of a categorical feature is high.

   One-Hot Encoding approach doesn't work on the test data, if any unseen categorical value gets added in the test data.

3) **Mean Replacement**

For example, if we have 200 distinct values in the 'country' column, then if we apply one-hot encoding on it, we get a 200-dimensional vector. Here as the number of distinct values in the categorical feature is very high, we get a large and a sparse vector.

One hack to handle this problem is to replace the 'country' column values with the average height of all the individuals of those countries. For example, if we have one of the 'country' column values as 'India', then 'India' has to be replaced with the average of the height values of all the individuals whose 'country' column value in the dataset is 'India'. This way, we could avoid occurrence of high dimensionalities.

So in this approach we are replacing each category of a categorical feature with the average of the output values(y_i) having the same category value. This approach is problem-specific.

4) Domain Knowledge

Taking a value as standard and replacing the value with the distance. For example, if we are considering 'India' as the standard value in the 'country' column, then we have to compute the distance from 'India' to that country, and fill this distance value in place of the country name.

This is because, we are taking a point as reference and are measuring/comparing the other values with respect to this point.

Ordinal Features

Ordinal features are the features which have categorical values, but there is an ordering between the values.

For example, if we want to rate a service/product, then the values could be

Very Good → 5
Good → 4
Average → 3
Bad → 2
Very Bad → 1

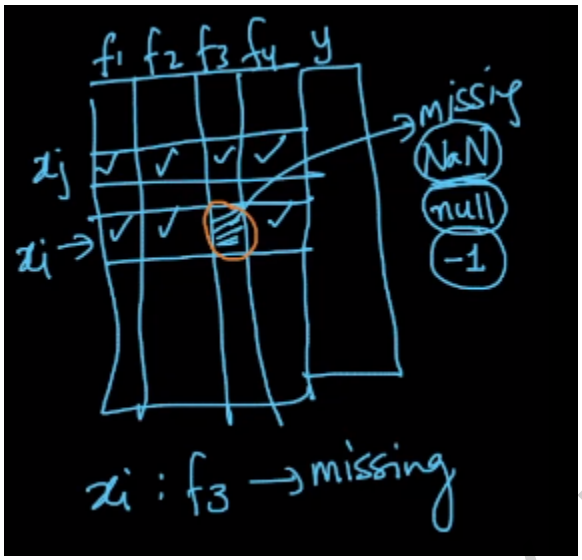
We can give numerical values for these ratings, and it is accepted.

Note: There is no strong rule of converting a categorical variable to numeric. We have to keep trying all the above discussed techniques and see which works best according to the problem context.

31.16 Handling Missing values by Imputation

Missing data occurs very frequently in the real-time. It could be due to

a) Data getting corrupted
b) Collection error (the researchers might have forgotten to collect)



1) **Imputation in case of a Regression problem**

One of the featurization techniques to handle the missing data is by **Imputation**. Imputation is the process of replacing the missing values with something else. There are 3 strategies in Imputation.

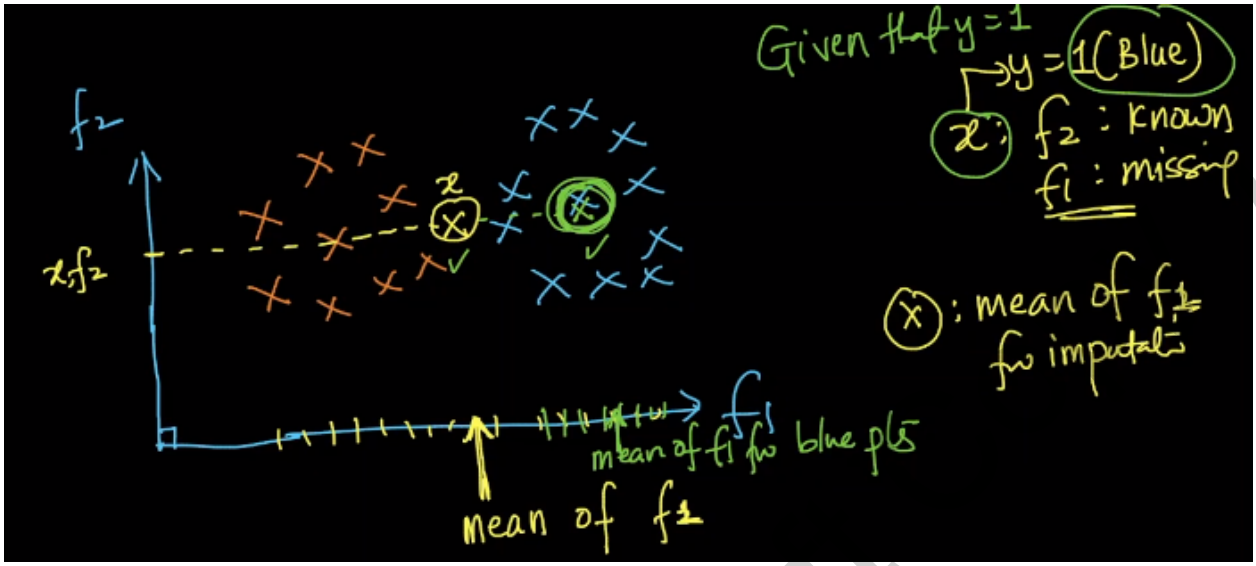
(i) Replacement with mean
(ii) Replacement with median
(iii) Replacement with mode

2) **Imputation in case of a classification problem**

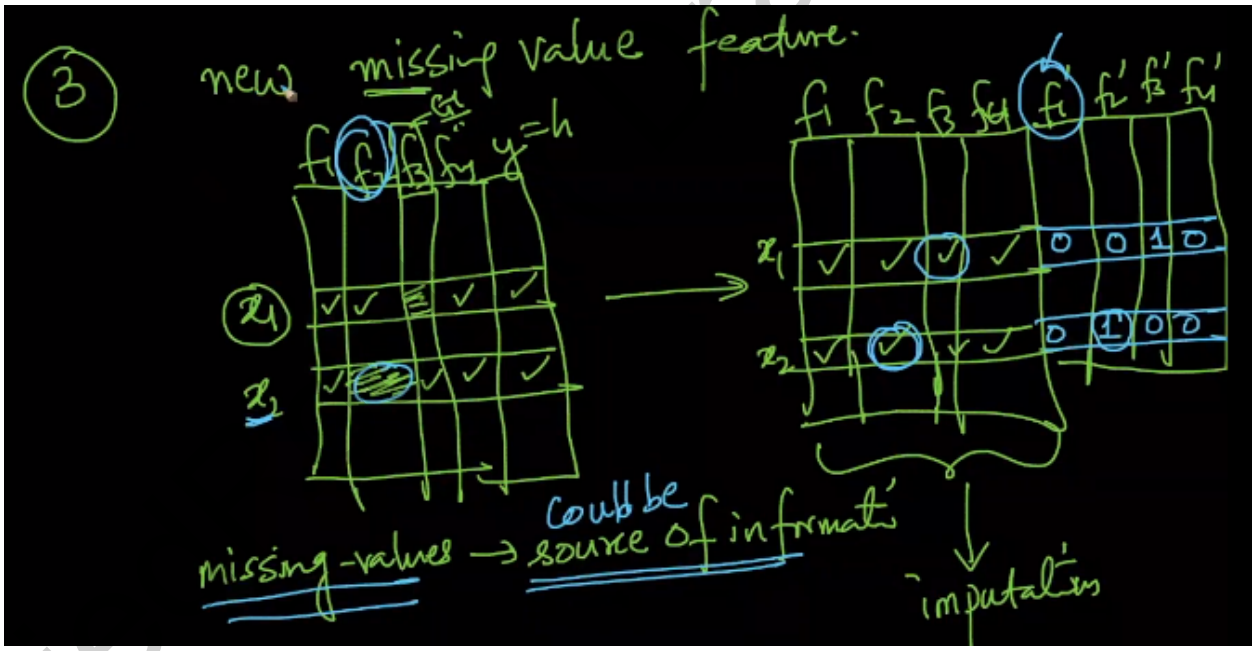
So far in the case of a regression problem, if $x_i \in R^d$ is our input, say a point 'x_i' has got missing values. Then we used to replace those missing values with the mean/median/mode of the non-missing values among those features.

In case of a classification problem, we have to replace a missing value of a data point with the average of all the values of that column , which have the same class label as that of the missing point. This gives a better result.

So far we have discussed 2 ways. They are *Imputation without the class label* and *Imputation with the class label.*



3) Creating New missing Value Feature

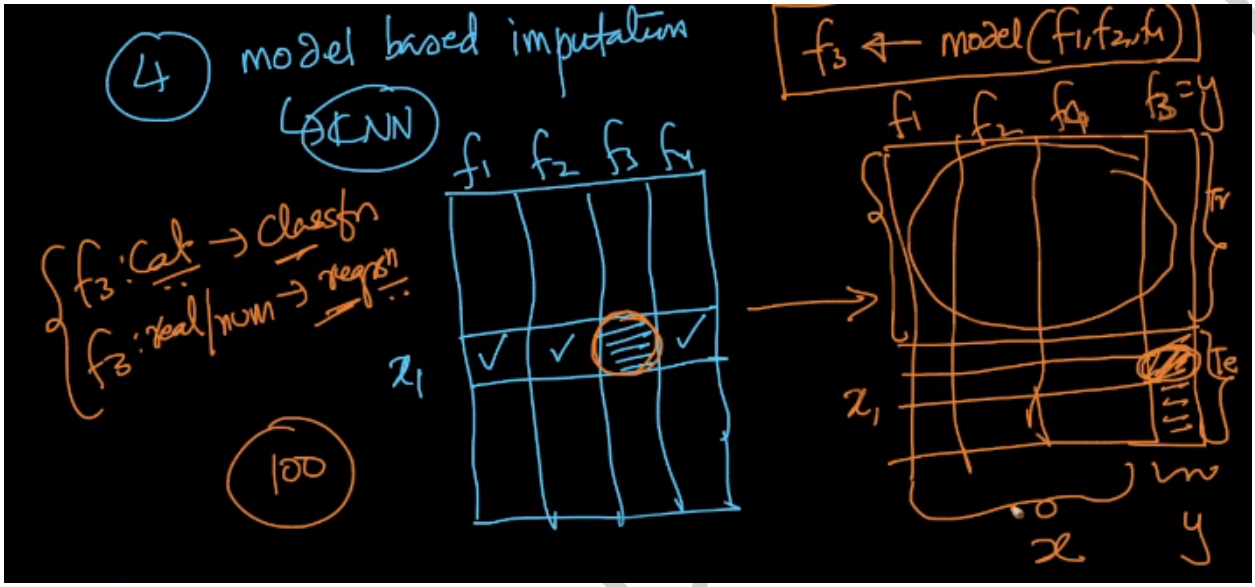


In this approach, we are first filling all the values by Imputation. We have to check how many features in the dataset contain missing values, and we have to create those many new features in the dataset. If we have the original features as 'f_1', 'f_2', 'f_3' and 'f_4', then if we have missing values only in 'f_2' and 'f_3' across the dataset, we then have to create only f_2' and f_3'. If we have the missing values in all the features, then we have to create f_1', f_2', f_3' and f_4'.

For a point 'x_i', if a feature 'f_d' is having a missing value, then only f_d' has to be filled with 1, and all new feature components should be 0. This is how we have to fill all the values in the data matrix, and then should go for model

building. This is because the missing values also could be a source of information.

4) Model based Imputation



Here we are taking the 'f_3' column as the output variable, and all the data points whose 'f_3' value is known are taken into the training dataset, and all those data points whose 'f_3' value is unknown are taken into the test dataset.

Now the model gets trained using the training data, and will predict the values of 'f_3' for the test data. The missing data values are replaced by the predictions obtained.

Here if 'f_3' is a categorical variable, we go with classification and if 'f_3' is a numerical variable, we go with regression. This is known as **Model based Imputation**.

The model based imputors take a lot of time when compared to the previous approaches.

As K-NN works on the concept of the neighborhood, if the points 'x_1', 'x_2', 'x_3' and 'x_4' are similar, then all features of those data points are also similar feature value wise.

31.17 Curse of Dimensionality

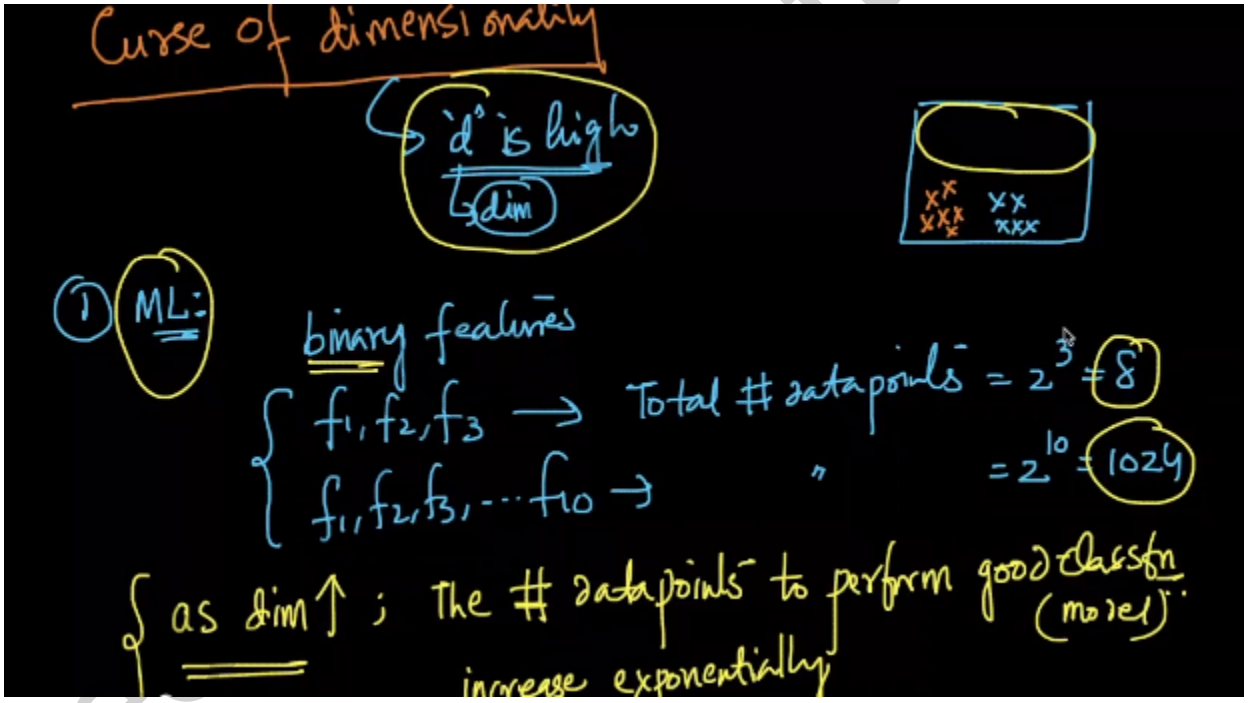
The curse of dimensionality is a concept that explains a bunch of things that happen when our dimensionality in the given problem is very high. The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high dimensional space.

1) **Machine Learning**

Let us assume we have the binary features. (Each feature can take either 0 or 1)

If we have 3 features(ie., 'f_1', 'f_2' and 'f_3') then total number of possible data points = $2^3 = 8$

If we have 10 features(ie., 'f_1', 'f_2', 'f_3', ….., 'f_{10}') then total number of possible data points = $2^{10} = 1024$.



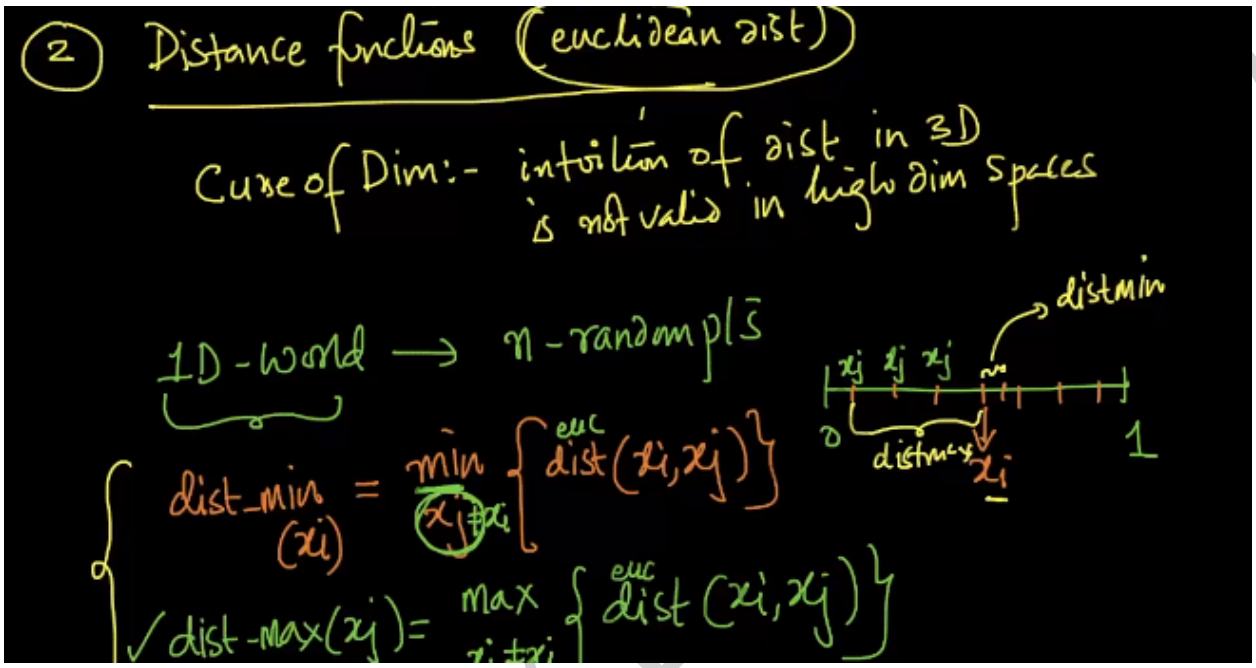
If there are no data points in the region, then we do not know what is happening in that region, and also we do not know to which class the points in that region belong to.

As the dimensionality increases, the number of data points needed for the model to perform well increases exponentially.

Hughes Phenomenon: If the size of the dataset is fixed, then the model performance decreases, as the dimensionality increases.

2) **Distance Functions (Especially Euclidean Distance)**

The intuition of distance in 3-D is not valid in high dimensional space. Let us assume we are working in 1-D space and we have 'n' random points.



$dist\_min(x_i) = \min_{x_j \neq x_i}\{dist(x_i,x_j)\}$ → (1)

$dist\_max(x_i) = \max_{x_j \neq x_i}\{dist(x_i,x_j)\}$ → (2)

$(dist\_max(x_i) - dist\_min(x_i))/dist\_min(x_i) > 0$ → (3)

(3) holds good only when d =1 (or) 2 (or) 3

As the dimensionality increases, the above term(ie., (3)) tends to become zero.

$\text{Lt}_{d \to \infty} (dist\_max(x_i) - dist\_min(x_i))/dist\_min(x_i) > 0$

In this case, **$dist\_max(x_i) \approx dist\_min(x_i)$**

So in the high-dimensional case, **$dist\_max(x_i) \approx dist\_min(x_i)$**

It means, every pair of points are actually equidistant from each other. In high-dimensional space, euclidean distance doesn't make any logical sense. As K-NN relies on distance, especially euclidean distance doesn't make any logical sense in higher dimensions. So ultimately K-NN doesn't work well in higher dimensions.

Solution to this problem

In order to build a K-NN model on higher dimensions, instead of using euclidean distance, we have to use cosine similarity.

Even cosine similarity also gets affected by the higher dimensional space, but very less when compared to that of euclidean distance.

Note: Techniques like BOW, TF-IDF, etc result in high dimensions. Hence it is recommended to use cosine-similarity when working on such data.

High Dimensional Space & Dense Matrix/Vector → Impact/Curse of Dimensionality is very high
High Dimensional Space & Sparse Matrix/Vector → Impact/Curse of Dimensionality is very low
The expression (3) holds good only if the data is in the form of a sparse vector/matrix as the data points are not distributed randomly in a sparse vector. That too it holds good only for euclidean distance.

3) **Overfitting and Underfitting**

Let us assume we use only K-NN for the time being to build a model. If the number of dimensions (d) increases, then overfitting also increases.

If the dimensionality is very low, then the underfitting increases.

Solution to be attempted in case of higher dimensions

1) We can go for the feature selection techniques to reduce the number of dimensions. But the feature selection techniques take the class labels into consideration.
2) We can go for dimensionality reduction techniques, which do not use the class labels, to reduce the number of dimensions.
3) When we are using K-NN on the text data, in order to reduce the impact of higher dimensionality, we can
   a) use cosine similarity instead of euclidean distance.
   b) use sparse representation of the matrix instead of dense representation of the matrix.

31.18 Bias-Variance Tradeoff

Bias-Variance Tradeoff is an important concept in the theory of Machine Learning. Bias-Variance Tradeoff gives the idea of mathematically analyzing overfitting and underfitting.

Generalization Error

Generalization Error is the error that is made on the future unseen data. Generalization Error is composed of three terms.

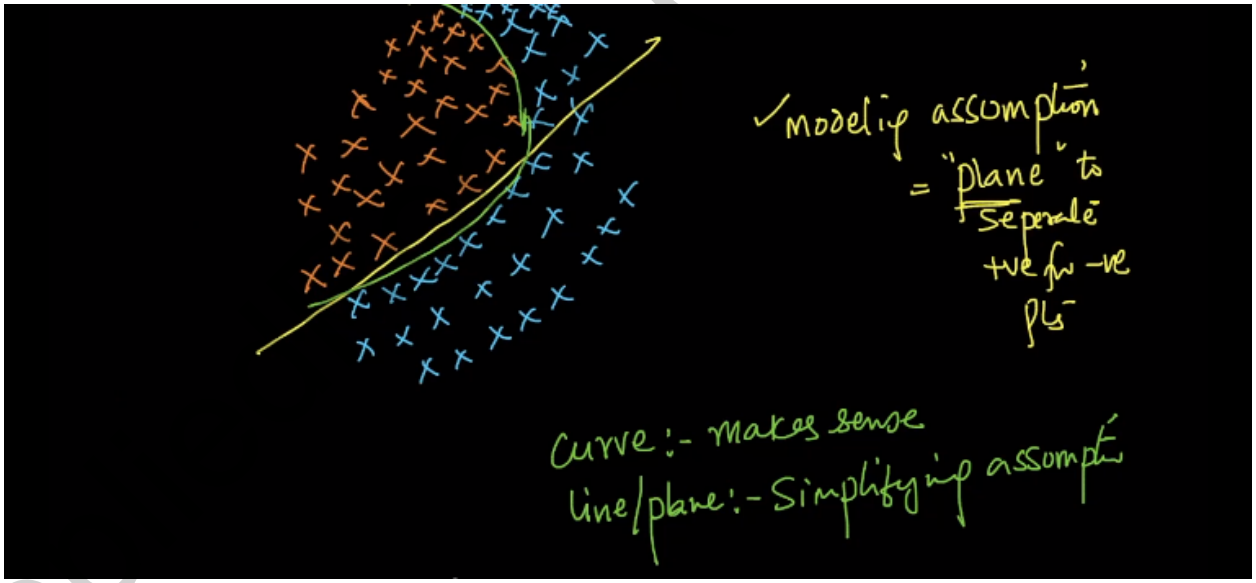
Generalization Error = Bias^2 + Variance + Irreducible Error

The main goal of Machine Learning is to keep the Generalization Error as low as possible. The irreducible error is the error that could not be reduced further, as there is no model that is perfect at predictions.

So if we want to reduce the generalization error, we have to reduce the Bias and the Variance.

Bias

Bias is the error occurred due to simplifying assumptions. High Bias means Underfitting. Let us assume we have the data points as shown below.



Modelling Assumption: We can use only a plane to separate the +ve and the -ve points.

So practically, a linear plane could not perfectly separate the '+ve' and the '-ve' classes. It leads to misclassifications.

Whereas a curve can perfectly classify both the classes and separates them. Here if we limit ourselves only to use lines/planes, then we are making simplifying assumptions about the separation of the two classes. The error occurred due to such simplifying assumptions is called **Bias**.

Let us assume we have 'n' data points in the training dataset with 80% of the points belonging to the -ve class and the remaining 20% of the points belonging to the +ve class.

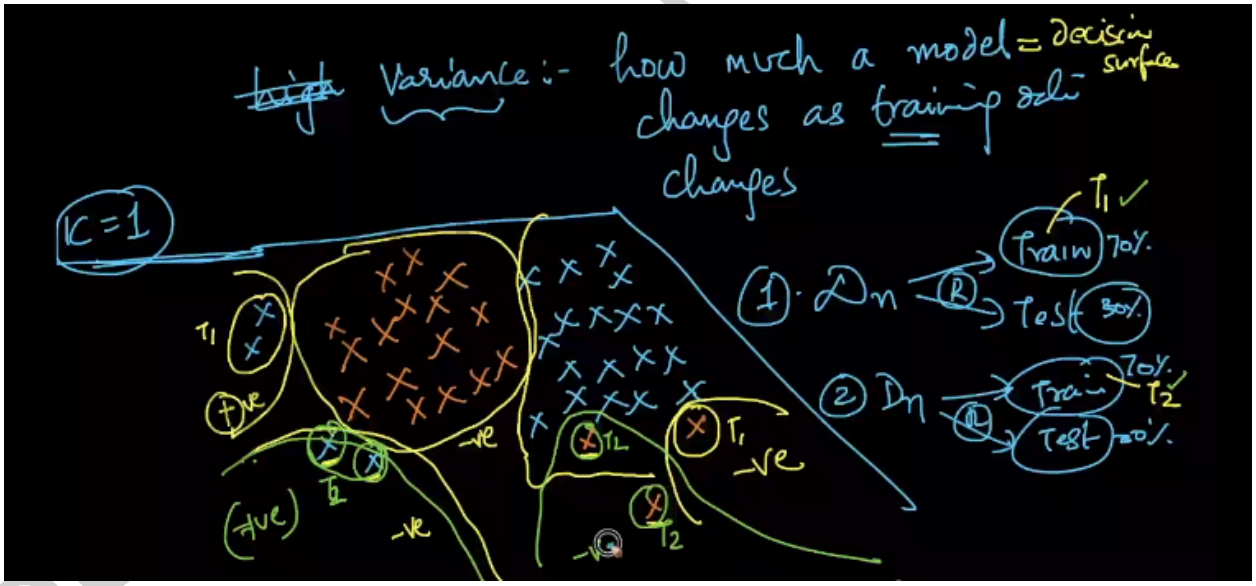
If we are using the K-NN model, with **K=n**, then each and every query point 'x_q' will be classified as -ve, as the given dataset is an imbalanced one. Here the dominant class will be the class label for any newly arrived query point. We made an assumption that the points belonging to different classes can be separated using a linear surface, but here there's no separation at all, as all the points are being classified as -ve. In such a case, we end up having high bias, and this scenario is known as **Underfitting**.

If K=n, then we say the model is having high bias, and is underfitting.

Variance

Variance is the measure of how much a model changes as the training data changes.

Let us assume we have the data points as shown below



Let us assume we have the dataset 'D_n' and we are splitting it randomly into 'D_{Train}'(70%) and 'D_{Test}'(30%). So 70% of the points come into 'D_{Train}' and the remaining 30% of them go into 'D_{Test}'.

If we combine both the sets, and split them randomly again into 'D_{Train}'(70%) and 'D_{Test}'(30%). This time we get slightly a different set of points into 'D_{Train}' when compared to the 'D_{Train}' obtained in the previous split.

When we generate different training datasets, the small changes in the training datasets result in very different models. (ie., The changes are the data points in one training set that do not occur in the other training sets. This leads to different decision boundaries for different training datasets)

Variance is the measure of how a model changes as the training data changes. So in K-NN, for the lower values of 'K' (ie., K=1), there will be large changes in the decision boundaries, for small changes in the training dataset. We call this High Variance (or) Overfitting and such a model is known as a High Variance model (or) Overfitting model.

As our main goal is to reduce the generalization error, in order to get it done, we have to reduce the bias and the variance (as it is not possible to reduce the irreducible error).

Generalization Error = Bias^2 + Variance + Irreducible Error

Reducing the Bias in our model means ensuring there is **no underfitting**. **Reducing the Variance** in our model means ensuring there is **no overfitting**.

Example

K-value	**Bias**	**Variance**	**Irreducible Error**	→	**Generalization Error**
K=1	10	100	3	→	113 (overfit)
.					
.					
K=5	12	10	3	→	25 (best-fit)
.					
.					
K=n	100	2	3	→	105 (underfit)

The best solution occurs if the 'K' value is in between 1 and 'n'.

High Bias, Low Variance → Underfitting
Low Bias, High Variance → Overfitting

As the 'K' value tends to increase towards 'n', then the K-NN model underfits.
As the 'K' value tends to decrease towards '1', then the K-NN model overfits.

31.19 Intuitive Understanding of Bias-Variance

Let us assume we have a dataset 'D' that is split into 'D_{Train}' and 'D_{Test}'. We build a model using 'D_{Train}', and compute two types of errors. They are the **Training Error** and the **Test Error**.

Training Error = difference(y_i, y_i^) on D_{Train}

Test Error = difference(y_i,y_i^) on D_{Test}

1) **When do we say the model has High Bias (or) the model Underfits**

   If the Training Error is high, then the bias is also high in the model. It means the model underfits.

   In K-NN, when K=n, the Bias is high and we say the model underfits. If the Training Error is low, then the Bias in the model is also low.

2) **When do we say the model has High Variance (or) the model Overfits**

   a) If the Training Error is low and the Test Error is high, then the model is an overfit model, and the variance in the model is high.

   b) If the Training data changes slightly, and the model performance changes drastically due to the change in the Training data, then we can say the model has high variance.

   In K-NN, if a single point changes in the Training data, and if it leads to severe changes in the model performance, even then we say the model overfits, having high variance.

Note: As the section 31.20 is on the revision question, we are not giving any notes for them. You can find the links on the web page itself.

31.21 Best and Worst case of an algorithm

1) **High Dimensionality**
   If the dimensionality of the data is small(say d<10), then K-NN is the best algorithm to go for.
   If the dimensionality is high, then the model will face the problem of curse of dimensionality(especially when we use Euclidean Distance as the distance metric)
   If the dimensionality is high, the model interpretability also reduces and also the runtime complexity of K-NN/KD-Tree/LSH increases.

2) **Low Latency**
   Low Latency Systems are the systems that give the results faster. (Example: Search Engines)
   K-NN should never be used in Low Latency Systems, as the runtime complexity of K-NN is very high. Even if there is a requirement to use K-NN, we should prefer either KD-Tree or LSH.
   Order of runtime complexities: LSH>KD-Tree>K-NN
3) If we can find the appropriate distance measure, then it is good to apply K-NN.
   If someone gives us the similarity matrix (or) distance measure (or) the formula to compute the distance, then K-NN is better.
   But in cases, when we do not know which distance measure has to be used, then K-NN will not be a good choice, irrespective of the context you are using.