Revision done

Ch 4: Classification algorithms in various situations

4.1: Imbalanced vs Balanced dataset

g) Consider eg of 2-class classification

D_n → n_1 +ve pts

D_n → n_2 −ve pts.

Let $n = 1000$

If $n_1 \approx n_2$ → Balanced dataset

eg → $n_1 = 580$, $n_2 = 420$ $\quad n_1 \neq n_2$, still $n_1 \approx n_2$

significantly less than

if $n_1 \ll n_2$ or $n_2 \ll n_1$ it is imbalanced dataset.

eg $n_1 = 100$, $n_2 = 900$ $\qquad$ $n_1 = 850$, $n_2 = 150$

Imbalanced dataset may create problem.

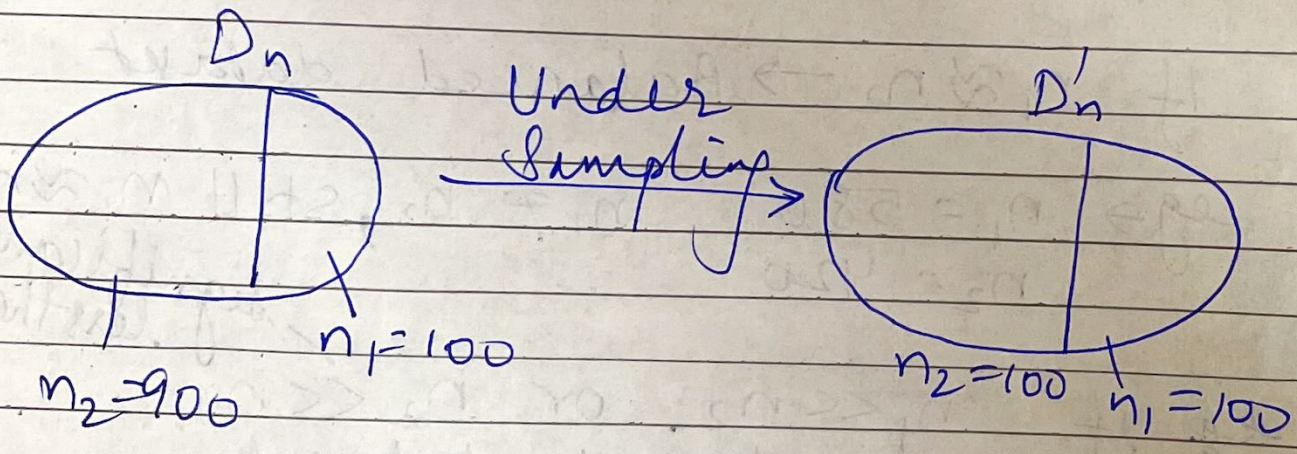
Just because there are more points of the majority class, it may be more biased towards it.

So, how to work around the issue of imbalanced dataset.

1) Undersampling

D_n → n_1 : 100 +ve

D_n → n_2 : 900 −ve.



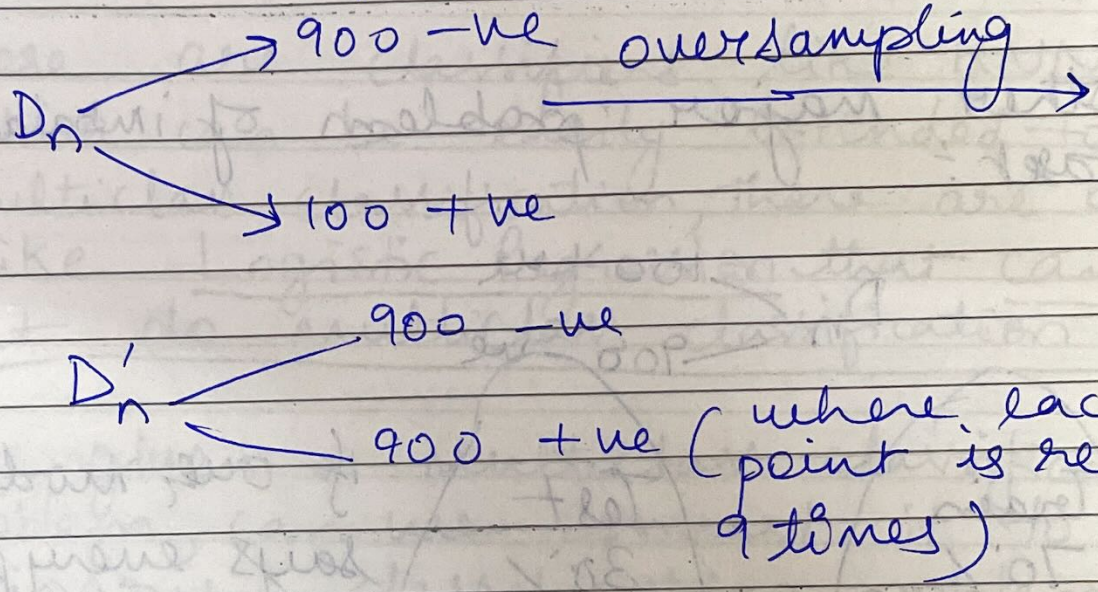
So basically we undersample -ve pts equivalent to n_1 to D'_n

Now dataset D'_n is balanced Further modelling done on D'_n.

Problem with Undersampling

$|D'_n| \ll |D_n|$, the model may not work that well as we are throwing away a lot of data.

2) Oversampling



1) Simply repeating the existing pts. is one of the very simplest technique.

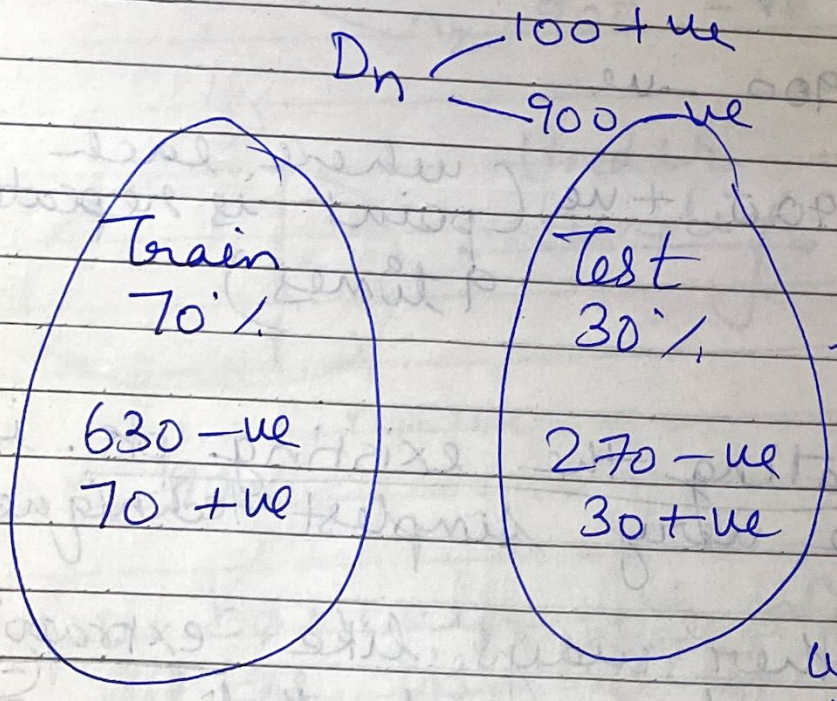
2) There are other ways like extrapolation (- or where we create new synthetic points in the region bounded by the original set of the pts. of that class.

3)

Another way of implementing repitition of pts instead of manually repeating is by assigning class weights (In sklearn this method is not implemented for KNN)

In the above example, class weight of +:9 (more weight to minority class), class weight of -:1.

Another major problem of imbalanced dataset:



If our model says every pt. is -ve.

Accuracy = 90%

which is wrong as it is a dumb model

So accuracy is not the right measure for imbalanced data.

Imbalanced datasets are very much prevalent in real world scenarios

eg- medical : 10% cancer
90% non cancer.

ecommerce 10% buy
90% don't buy.

Video 3 Multi class classification

There are classifiers like KNN which can be easily extended to do multiclass classification, there are other like Logistic Regression that can not do multiclass classification easily.

So, given a multiclass classification problem, can we convert it to a binary classification problem :-

Let $y_i \in \{1, 2, 3 \ldots C\}$

1) D_n $\begin{cases} \{(x_i, y_i) \mid y_i = 1\} \longrightarrow +ve \\ \{(x_i, y_i) \mid y_i \neq 1\} \longrightarrow -ve \end{cases}$

This now can be handled by a single binary classifier.

Similarly, for each of the classes we can build a binary classifier like 2 or ≠2 vly 3 or ≠3 vly 4 or ≠4 vly 5 or ≠5, etc.

vly c or ≠c

So, multiclass classification problem = C binary classification problem.

This technique is called **One vs Rest**

V4: KNN, given a distance or Similarity Matrix (logistic Reg. won't work when dist/sim matrix are given.)

There will be cases when we can get only the similarity matrix $S_{n \times n}$

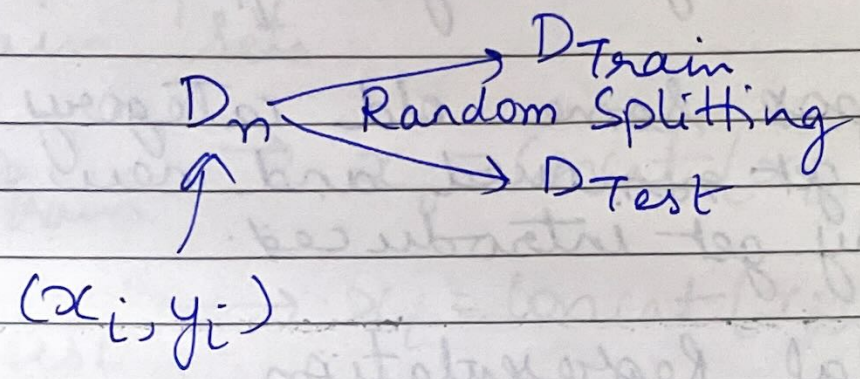
S_{ij} = Similarity (x_i, x_j), but can't get vector featurization of data points.

eg x_i = Paracetamol can't be converted to vector. But pair of chemical compounds can be written similarity

We can convert similarity matrix to distance matrix as they are reciprocal of each other.

KNN works great as ultimately from the given D_n, we care about $S_{n \times n}$ only in KNN.

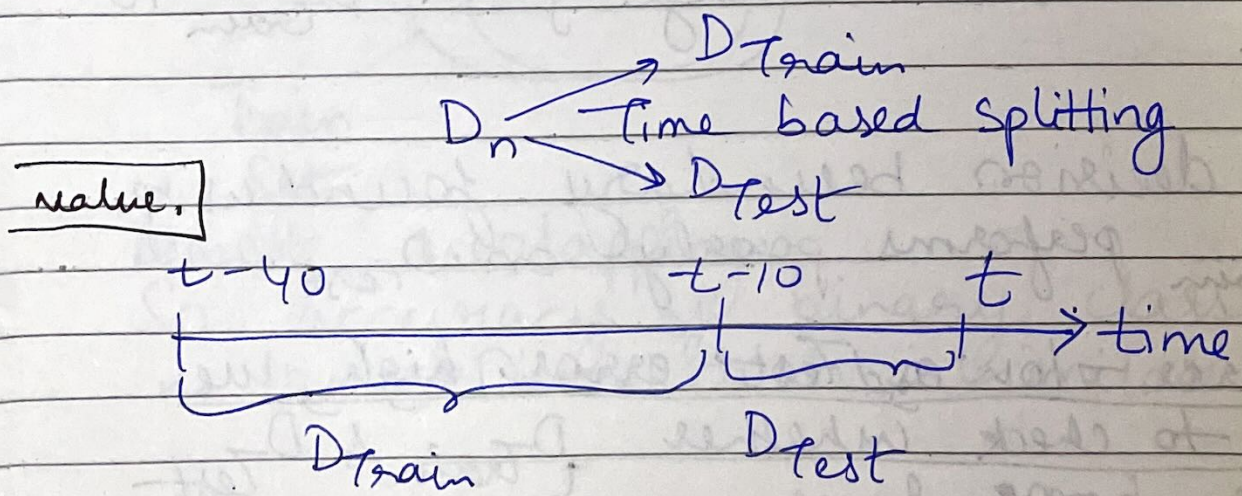
V5: Train and Test Set Differences



In random splitting each point has equal probability for going to D_{Train} & D_{Test}

So both D_{Train} & D_{Test} will be similar.

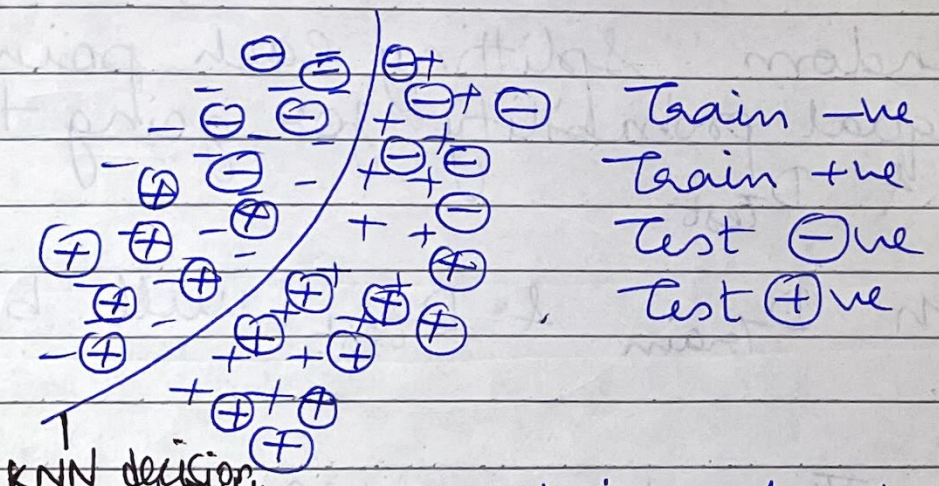
But in Time based splitting it is not the case.



In Time based Splitting D_{train} & D_{Test} could get very different.

eg → Suppose some old category of products get removed and new category get introduced.

Graphical Representation



D_{train} and D_{Test} being fundamentally different causes the distribution of -ve and +ve data change from D_{train} to D_{test}.

The decision boundary found using D_{train} performs poorly on D_{test}.

If cv error - low and Test error high, we need to check whether D_{train} & D_{Test} come from same distribution or not.

How to determine this?

1) Time split the data to test and train sets.

2) Time based splitting
D_{train} → $x_i = Concat(x_i, y_i), y_i = 1$

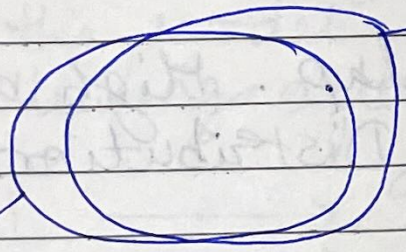
D_{test} → $x_i = Concat(x_i, y_i), y_i = 0$.

D_n'

This is to basically label all the training set points as 1 class and test set points as another. This gives D_n'.

Now train a binary classifier on D_n'

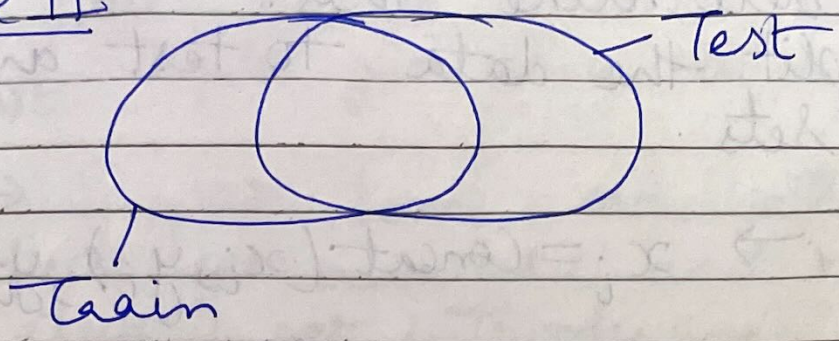
Case I:



Almost overlapping.

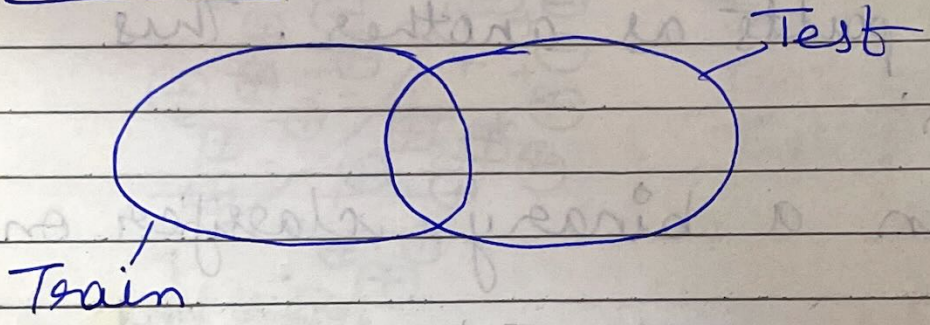
So accuracy of binary classifier is low ⇒ Distribution is similar.

Case II



Less overlap. Medium accuracy of classifier ⇒ Distribution not very similar.

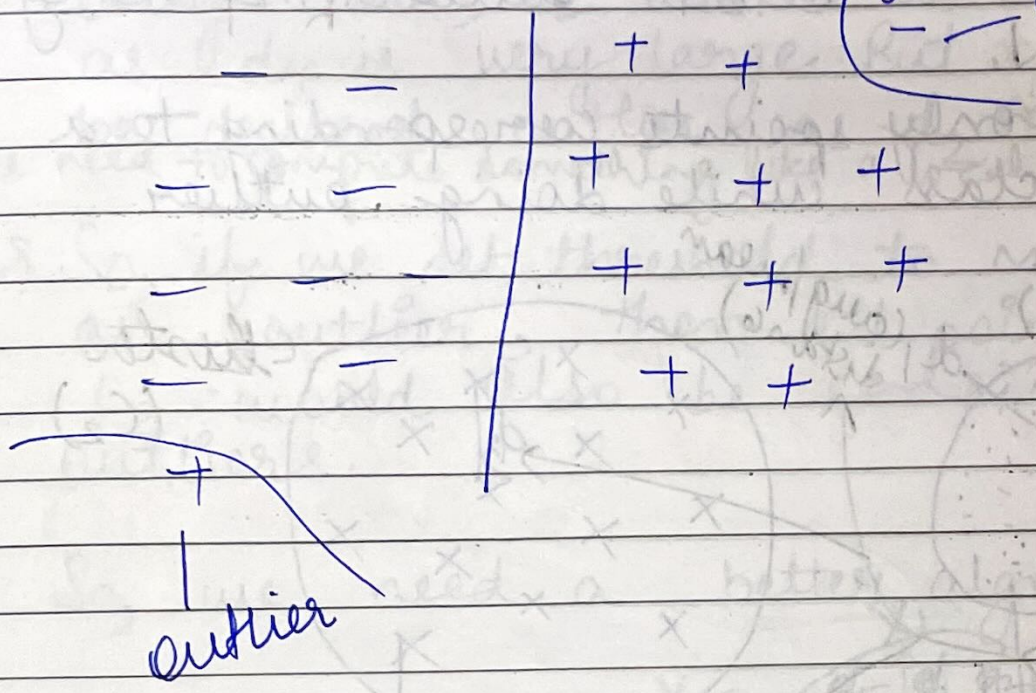
Case III



Very little overlap. High accuracy of classifier. ⇒ Distribution very different.

V6 Impact of Outliers

Let K be small say K=1.



When K is small, outliers can easily impact the model. So, if we get the following after 10 fold CV,

K=1 ⟶ 97%.
K=2 ⟶ 97%.
K=3 ⟶ 97%.
K=4 ⟶ 97%. — less prone
K=5 ⟶ 95%. to outliers
K=6 ⟶ 92%.

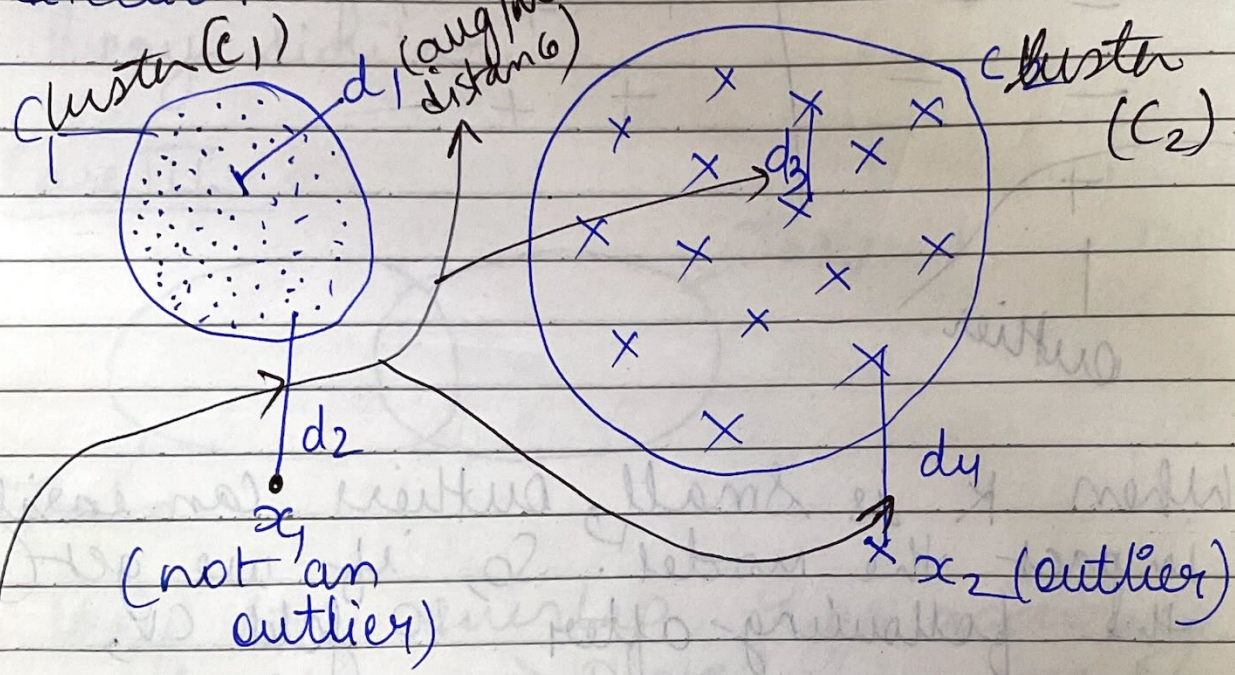
Always prefer to choose large K with high accuracy.

Methods to detect outliers:-

V7: Local Outlier Factor (Simple Solution: Mean distance to KNN):

Technique to detect outliers in data, Inspired by KNN.

Consider only points corresponding to a single class while doing outlier detection.



C_1 - Very dense cluster

C_2 - Sparser cluster.

x_1 & x_2 are outliers

Simple Solution :-

1) For every pt x_i compute its KNN. (mean distance from x_i to its KNN)
2) Calculate avg distance of x_i from its KNN
3) Sort x_i's by the avg. distance

4) If average distance is high → The point is outlier.

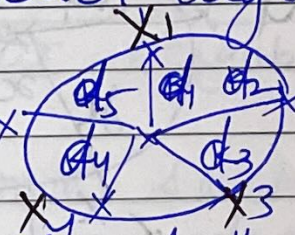
Using this, x_2 will be declared outlier as d_4 is very large. But x_1 wouldn't be declared outlier as $d_2 \approx d_3$

We need to compute something like a local density.

So, if we set threshold to make x_1 as outlier, then all points in C_2 would also be considered as outliers.

So, we need a better algo.

V8: K-distance



K-distance(x_i) = distance of the K^{th} Nearest neighbour of x_i from x_i.

5^{th} distance(x_i) = d_5; 1^{st} distance(x_i) = d_1

Neighbourhood of x_i [$N_K(x_i)$] = Set of all the points that belong to KNN of x_i which can be more than 'K' in cases of tie.

$N_{K=5}(x_i) = \{x_1, x_2, x_3, x_4, x_5\}$

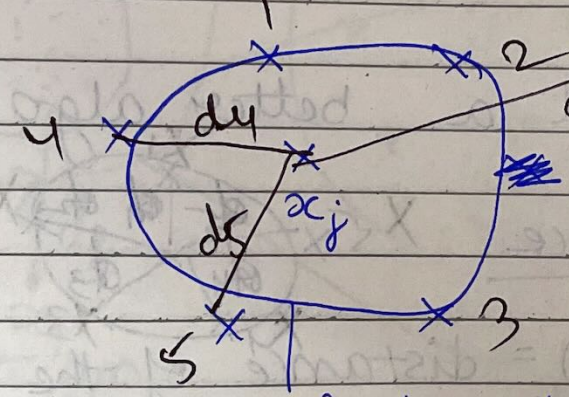
$N_{K=3}(x_i) = \{x_1, x_2, x_3\}$

V9: Reachability - Distance (A, B)

reachability distance (x_i, x_j) =

$\max(k\text{-distance}(x_j), \text{distance}(x_i, x_j))$

- $k\text{-distance}(x_j)$: distance of k^{th} NN of x_j from x_j
- $\text{distance}(x_i, x_j)$: actual distance b/w x_i and x_j



Neighbourhood of x_j ($N(x_j)_{K=5}$)
- $K=5$

Neighbourhood of x_j

if $x_i \in N(x_j)$
then reach-dist (x_i, x_j) = k-distance(x_j)
else if $x_i \notin N(x_j)$
reach-distance (x_i, x_j) = distance (x_i, x_j)

$\max(d_5, d_4) = d_5$

$\max(d_5, d) = d$

using neighbourhood

using Reachability distance

calculating some sort of dens

V10: Local Reachability-density (A)

$$lrd(x_i) = \frac{1}{\sum_{x_j \in N(x_i)} \left\{ \frac{\text{reach-dist}_A(x_i, x_j)}{|N(x_i)|} \right\}}$$

average reachability distance of x_i from its neighbours

denominator: no. of neighbours of x_i (Include duplicates with same distance)

$lrd(x_i)$ = inverse of average reachability distance of x_i from its Neighbours.

It is calculating some sort of density of the surrounding around that point.

OR.

$$lrd(x_i) = \frac{|N(x_i)|}{\sum_{x_j \in N(x_i)} (\text{reach-dist}(x_i, x_j))}$$

$|N(x_i)|$ → #points

(reach-dist(x_i, x_j)) → distance

VII: Local Outlier Factor (A)

$$LoF(x_i) = \frac{\sum_{x_j \in N(x_i)} Lrd(x_j)}{|N(x_i)|} \times \frac{1}{Lrd(x_i)}$$

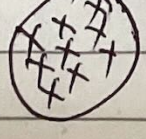
$Lrd(x_i)$ → lrd of only x_i

avg lrd of pts in the neighbourhood of x_i

$LOF(x_i)$ will be large if 1) $lrd(x_i)$ is small.
2) avg. lrd of neighbours of x_i is large.

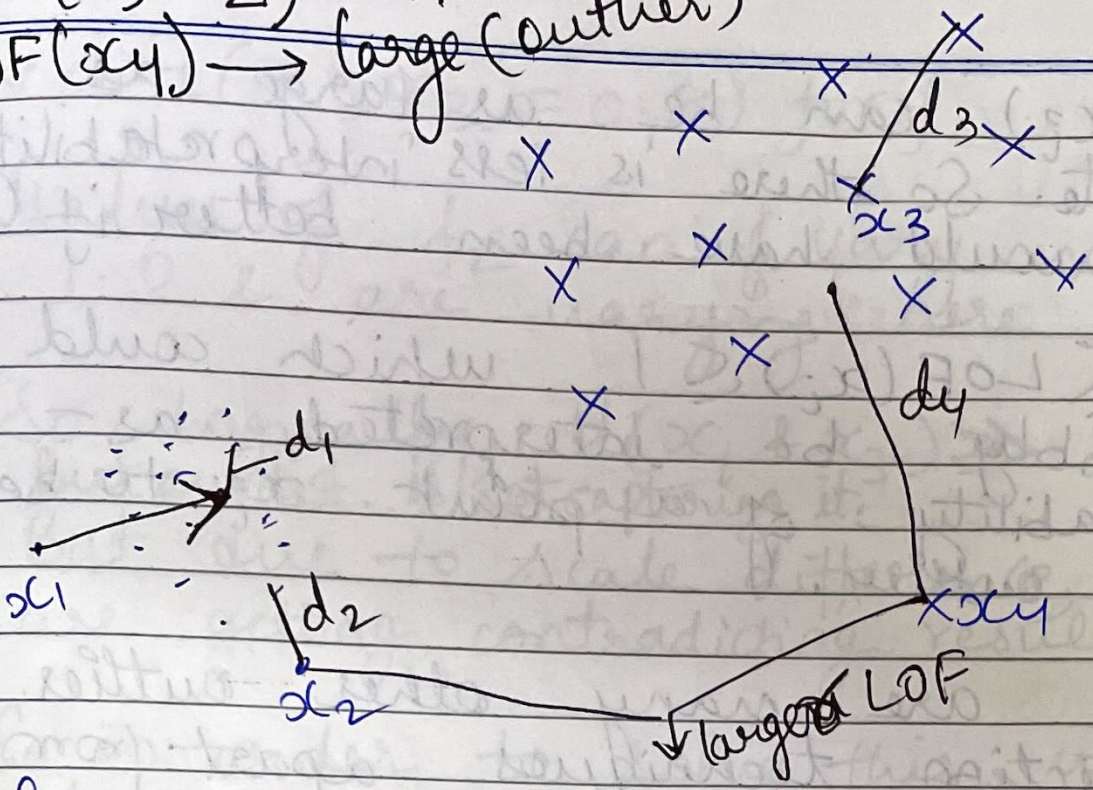
When $LoF(x_i)$ is ↑ → outliers.
↓ → inliers

This means density of points in neighbourhood is high but density of points itself is low.



x → outliers

LOF($x_1 \in C_1$) → Small
LOF(x_2) → large (outlier)
LOF($x_3 \in C_2$) → Small
LOF(x_4) → large (outlier)



So considering the figure, density of x_3 seems same as its neighbour same for x_4. hence both are inliers.

Avg reach. dist = d_1, d_2, d_3, d_4. Also, $d_1 < d_2 < d_3 < d_4$

But density of x_4 and x_2 are very less as compared to their neighbours. Hence outliers. $lrd(x_1) = \frac{1}{d_1} > lrd(x_2) = \frac{1}{d_2} > lrd(x_3) = \frac{1}{d_3} > \frac{1}{d_4}$

$lrd(x_1) \approx lrd(x \in N(x_1))$; $lrd(x_2) \ll lrd(x \in N(x_2))$

There is no hard and fast rule to set threshold for LOF value beyond which there will be outliers

But if we know from domain knowledge that 5% of points are outliers, then we can sort and remove top 5% points.

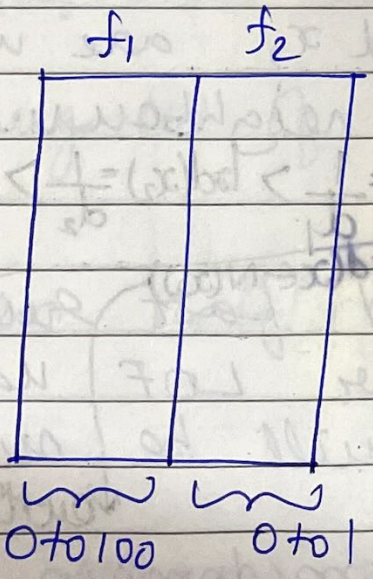
for each pt x_i compute $LOF(x_i)$
pick points with highest LOF → outlier

$LOF(x_i)$ can be as large as infinite. So there is less interpretability. It would have been better if

$0 \leq LOF(x_i) \leq 1$ which could possibly be interpreted as — probability of a point x_i to be an outlier.

There are many other outlier detection techniques apart from LOF.

V12: Impact of Scale and Column Standardization



Scale difference.

eg $f_1(0-100)$, $f_2(0-1)$

x_1: [23, 0.2]
x_2: [28, 0.2]
x_3: [23, 1.0]

i.e. $dist(x_1, x_2) = 5$ for f_1

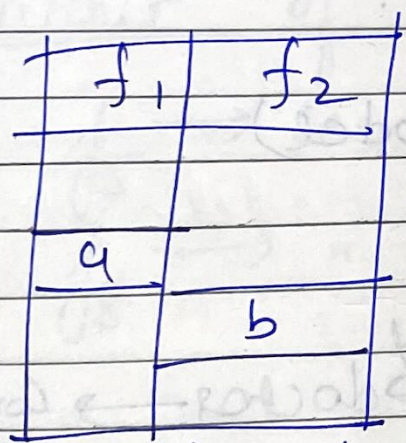
The values of feature f_1 can vary far by 100 units, but are varying only 5 units.

eu. dist $(x_1, x_3) = 0.8$) for f_2.

Values of f_2 can vary max by 1.0 & are varying here by 0.2

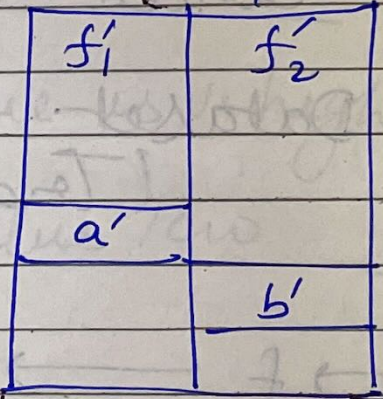
So logically, $d(x_1 \& x_3) \gg d(x_1, x_2)$
but eu. dist. is not capturing it.
But due to scale difference, we are getting contradicting results.

So in order to solve this, we do column standardization.



col Std.

transformed features



Mean $f_1 = \mu_1$
Mean $f_2 = \mu_2$
Std-dev $f_1 = \sigma_1$
Std dev $f_2 = \sigma_2$

Mean f_1' = Mean f_2' = 0

Std dev (f_1') = Std dev (f_2') = 1

$$a' = \frac{a - \mu_1}{\sigma_1}$$

$$b' = \frac{b - \mu_2}{\sigma_2}$$

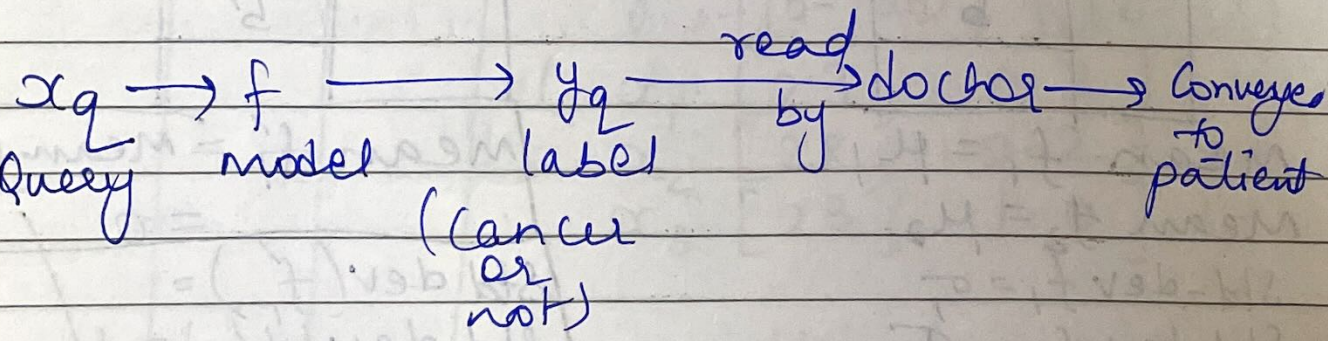
1) If we are using KNN with euclidean distance then 1st do col-std. Then apply KNN on transformed features.

2) There are models like decision tree that are scale independent also.

V18: Interpretability

Consider a medical domain problem be cancer detection.

Dataset $\longrightarrow$ f (model)
(Training)



If the model just gives the class label, then it might not help the doctor much as to why the model choose to do this. This kind of models are called black box model.

But if the model along with giving some class label, also gives some reasoning as to what made the model take the decision, then this reasoning is very useful for the doctor, such models are called Interpretable models.

eg- Say x_q is of dimension 10, where each feat. corresponds to result of a medical test.

Output of interpretable model →

f_5 → test 5 → value is v. high
f_8 → " 8 → value low

→ Hence patient has Cancer.

Doctor understands this additional info. as these are results of medical tests only.

Let's take KNN for eg.

$x_q \rightarrow$ model $\rightarrow y_q = 1$.

Reasoning: All KNN of x_q are labelled 1. Hence $y_q = 1$.

Lets say d & k both are small.
{d=10, k=7 - eg}

The model KNN can also give the 7-NN.

So, basically the doc. gets to know about 7 other patients' medical tests' results for 10 tests, which are similar to query patient.

Hence, doctor can compare the test results manually & get an indication as to why label = 1.

So, KNN is interpretable when d & k are small.

V14: Feature Importance and Forward Feature Selection

Suppose we are solving the problem of height prediction of an individual

We are given input features - weight, hair colour, hair length, skin colour, gender country.

clearly all the features are equally important.

From domain knowledge, we have idea that weight has a significant impact on height. Similarly, gender, country also play deterministic roles.

If the ML model also gives feature importance it will be useful for us in understanding the model better. Hence increasing interpretability.

KNN doesn't gives feature importance. easily but lacks are there

Logistic Regression & DT can give F Imp.

one of the technique for wrapper feature selection

selecting most imp features

Forward Feature Selection

Suppose we have 1k-dimension. data. This can cause us to face curse of dimensionality. Where euclidean distance doesn't hold much meaning of similarity anymore.

So we need to reduce dimensions. Our task here is classification (assumed). PCA & t-SNE preserve distance. They don't care about classification.

So, we make use of technique called Forward Feature Selection, which works irrespective of classification model.

1) Take the data and split to D_{train} & D_{test} $\{D_n \rightarrow D_{train} + D_{test}\}$

2) Take single feature & train the model for each feature in feature set.

3) Select the feature giving maximum test accuracy.

Repeat above steps each time for each ~~Dese Feature Selection~~ feature selected and the new feature. This is called forward selection

→ Don't care which model you give (advantage) / Highest time complexity (Disadv.)

Why can't we just calculate accuracy for each single feature, sort them in descending order and then take ~~8~~ 1st d' feature?

~~But its~~ Say f_{10} & f_6 are 1st and 2nd most accuracy feature.
But its possible that f_{10} & f_6 are very much co-related. Hence, adding it won't help much improvement.

Instead we want — Given that we ~~have~~ already have some feature which new feature adds the most value?

eg — Suppose $f_1, f_2 \cdots f_{10}$ → 10 features total.

After going through each feature say we get f_{10} with highest accuracy.

Next we'll train the model with features f_1, f_{10}, ie $[1, 10]$ & $i \neq 10$.

Say we found f_6, f_{10} have highest accuracy.

Backward Selection — removing features with lowest accuracy.
Time Complexity ⇒ each iteration :- Training and testing d (models 1) d models 2) d-1 3) d-2

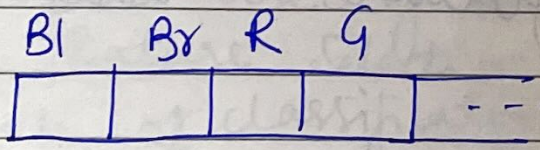
VI5: Handling Categorical and Numerical Features

1) Given a number

{ black, brown, red, gray }
1 2 3 4

But numbers have an inherent order
ie, 1<3.
But black<red is absurd.

2) One hot Encoding



Bl: 1 0 0 0 ...

Br: 0 1 0 0 ...

R: 0 0 1 0 ...

G: 0 0 0 1 ...

Binary vectors of size of number of distinct elements.
These vectors can't be compared & that's how these hair colours should be.

Dis-advantage:-

country = 200.

So, we'll have to use a 200 dim vector just for a single feature. which is the problem we faced for BoW encoding.

3) Suppose the task is predicting height of person.

So, in place of country, what if we replace it by avg. height of people in that country.

The idea is to replace categorical value with avg. output value for that category.

This is very much problem specific.

4) Domain knowledge

Let's say from domain knowledge we know that a distance from equator ↑, height ↑.

So using this knowledge, we can convert categorical feature to numeric features.

features that can be ordered.

5) Ordinal features:-

feature :- How do you like Indian food?

→ V. good
good
avg
bad
V. bad

} logical ordering so, we can assign them numbers / numeric feature.

V16: Handling Missing values by Imputation

It is very common in real world.
It may be due to data corruption, also called as or collection corruption.
or missing data.

1) Imputation (Replacement)

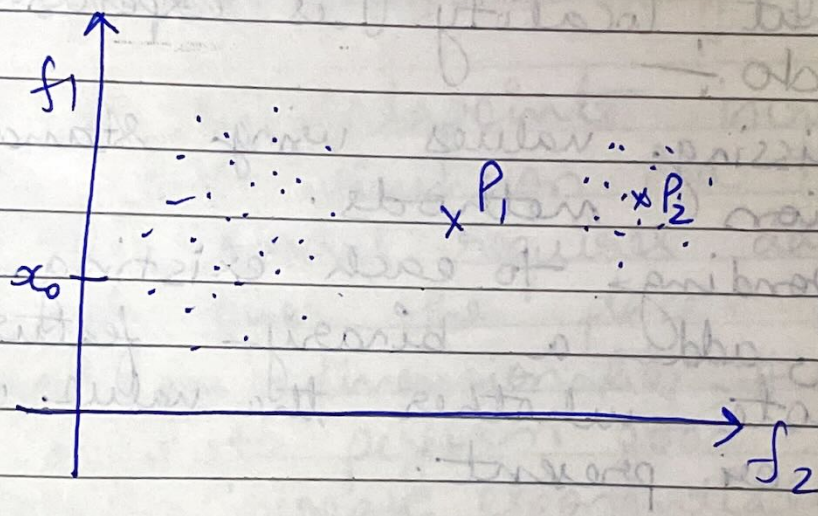
Take all the non-missing value in that column & find its mean, median or mode - most frequent value and substitute those to all missing values.

Sklearn.preprocessing.Imputer

2) Class based imputation

Take all the non-missing value in a column corresponding to a particular class and find their mean/median/mode and substitute it to missing values of that class.

Geometric Intuition



Let's say for x_0, f_2 is missing. $y=1$ for x_0
If we take mean of f_2 of all points then x_0 will be at P_1 and if we only consider pts. with $y=1$ then x_0 will be at P_2.

Intuitively, P_2 makes more sense.

3) Creating a New missing value feature

In this we treat missing values as source of information.

eg - Say blood group of some persons is missing.

It may imply that getting in blood group test in that locality is expensive.

Ways to do :-

① Fill missing values using standard imputation methods.

② Corresponding to each existing feature, add a binary feature to denote whether the value was missing or present.

4) Model based Imputation (Take significantly more time)

Usually KNN is used because of neighborhood concept.

We convert the missing values problem to standard classification and regression.

① Say for a bunch of points, f_3 missing

② Treat f_3 as output label (y).

③ $D_n \xrightarrow{f_3 \text{ present}} D_{train}$

$\xrightarrow{f_3 \text{ missing}} D_{test}$

Now train a model & predict values.

revised

VII: Curse of dimensionality

There are
~~It is a~~ weird things that happen when d is very high that are not common sensical.

For binary features, # of features = 3, then # of data points $= 2^3 = 8$

# of features = 10, then # of data points $= 2^{10}$.
$= 1024$

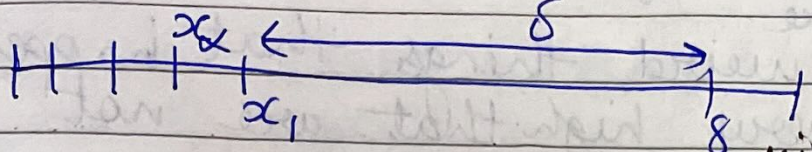
So. no. of datapoints increase exponentially with dimension. And to perform well, the model requires data points from all over the place. Hence with increase ~~in~~ in dimensionality, # of datapoints required to perform good classification/regression increase exponentially.
This is unlike Real valued ~~data~~ (Regression) features.

1) Hughes phenomenon
For fixed no. of data points, peryormance decreases with dimensionality & increases.

2) Distance Functions (Euclidean distance)
The intuition that we have of distances in 2-D, 3-D is not valid for high dimensional spaces.

$$\text{dist-min}(x_p) = \min_{x_q \neq x_p} \{\text{euclidean dist}(x_p, x_q)\}$$

Consider 1-D space → n-random points



Consider x_i, dist-min(x_i) = minimum distance from closest point to x_i to $x_i = \beta$

Similarly, distance-max(x_i) = $\max_{x_q \neq x_p} \{\text{euc dist}(x_i, x_j)\}$

So intuitively in 1D, 2D or 3D,

{(distance-max(x_i) − distance-min(x_i)) ÷ distance-min(x_i) is greater than 0.}

But mathematically it can be proven that as

$$\lim_{d \to \infty} \frac{\text{distance max}(x_i) - \text{distance min}(x_i)}{\text{distance-min}(x_i)} \longrightarrow 0 \quad (\text{tends to})$$

which implies that

$\Rightarrow$ distance-max$(x_i) \approx$ distance min(x_i)

So in higher dimension, every pair of points are roughly equally distant for n random pts, if

distmax$(x_q) \approx$ distmin(x_q)

$$d(x_i, x_j) \approx d(x_i', x_j')$$

KNN relies on Euclidean distance → in high dim → makes no sense

→ KNN doesn't work well.

Soln Use other distances like Cosine-similarity. It is also affected with increasing dimension, but not much as compared to Euclidean.

If data has high dimension and dense, curse of dimensionality is high. and if data has high dimensionality and sparse, then impact of dimensionality lowers.

The above limit has been done considering

1) Uniform distribution of points randomly. In sparse data, most points will have no component in many directions. Hence, less impact.
2) Euclidean distance – If other distances considered, it gives less impact.

3) Overfitting and Underfitting

As d ↑ → overfitting ↑

Solution of high dimensions :-

1) Forward feature Selection :- classification oriented, use class labels. (Model dependent)
2) Dimensionality reduction - PCA, TSNE - Don't use class labels (Model Independent) Not classification oriented.
3) Using cosine similarity instead of euclidean distance. for eg in KNN.
4) Sparse representation instead of dense representation. for eg. in KNN.

V18: Bias Variance Tradeoff

It was studied in Theory of Machine Learning.

Generalization error = Error to be made on future unseen data.

= $Bias^2$ + Variance + irreducible error

Error that you can't reduce further for a given model.

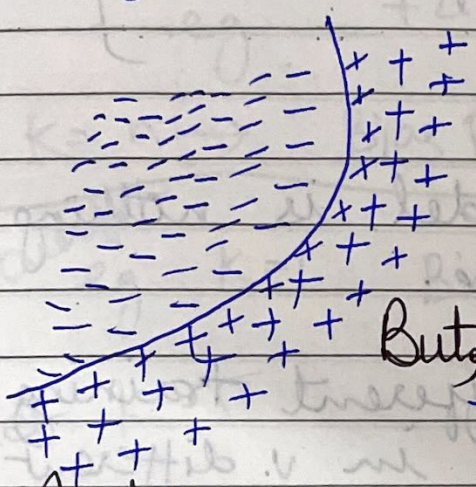
Goal of ML is Gen. error ↓.

1) Bias error = Errors occured due to simplifying assumptions.

High bias = Underfitting

Simplifying assumption

Modelling assumption
I'll use "plane" to separate +ve from -ve pts.



But, curve makes sense in this data.

And there is no way to separate my points using plane. Hence bias error occur.

→ In KNN, if $k = n$, we make simplifying assumption to assigning dominant class label to x_q.

→ High bias means underfitting which is you are making simplifying assumptions.

2) Variance error =

Variance means → how much a model changes as training data changes.

1) $D_n \rightarrow D_{Train}$, D_{Test} { Random Splits - 1 }

Geometrically a model is nothing but decision surface.

when you generate different training datasets, which result in v. different models or decision surface.

↓

Causes large changes in the model

↓

High variance model

↓

Overfit model.

As K ↑ variance ↓

↓

K in KNN

Gen error = $Bias^2 \downarrow$ + Var ↓ + Irr. error

↓ no underfit

| no overfit

K=1 → Bias ↓ Var ↑

K ↑ → Bias ↑ slightly +2 Var ↓ drastically −20

eg

K=n → Bias ↑ very high Var ↓

---

eg overfit

eg K=1 Bias = 10 + Var = 100 + Irr err = 3

let K=5 12 + 10 + 3 = 25 ← 113

K=n 100 + 2 + 3 = 105

Under fit

V19: Intuitive Understanding of bias-variance

Train error = difference $(y_i, \hat{y}_i)$ on D_{train}

Test error = difference $(y_i, \hat{y}_i)$ on D_{test}

1) High bias / ~~Overfit~~ Underfitting

→ If Train error ↑ = bias ↑

KNN, k=n

→ If Train error ↓ = bias ↓

2) High Variance / Overfitting

→ If Train error ↑ & Test error ↑

= Overfit

= ↑ High Variance

→ If Training data changes slightly, would the model change this causes Variance ↑.

eg KNN, k=1, High Variance.

V21: Best and Worst case of algorithm of KNN

1) When dimensionality is small (<10) not large → KNN is good.

When dimensionality is high we go into curse of dimensionality, interpretability ↓, runtime of kd tree/LSH ↑. KNN - not good

2) Low latency (→ delay) System. does not gives quick result. as it has high runtime complexity. KNN is not good

3) If you can Find right distance measure, then KNN works well.
→ For text data — Cosine-distance.
→ for genomic data for eg, If we are given similarity/distance matrix then use KNN.

Every ML technique, knowing whether it works best and worst is important, which will help in applying them better in future.