Classification Algorithms in various situations

① Imbalanced Vs Balanced datasets

consider example of 2-class classification.

$$\begin{tikzcd} & n_1 \text{ +ve pts} \\ D_n \arrow[ur] \arrow[dr] & \\ & n_2 \text{ -ve pts} \end{tikzcd}$$

let $n = 1000$

if $n_1 \approx n_2 \longrightarrow$ balanced dataset

ex→ $n_1 = 580$ $n_1 \neq n_2$ $n_1 \approx n_2$.
$n_2 = 420$

if $n_1 << n_2$ or $n_2 << n_1 \longrightarrow$ imbalanced dataset.

Ex→ $n_1 = 100$ $n_1 = 850$
$n_2 = 900$ $n_2 = 150$.

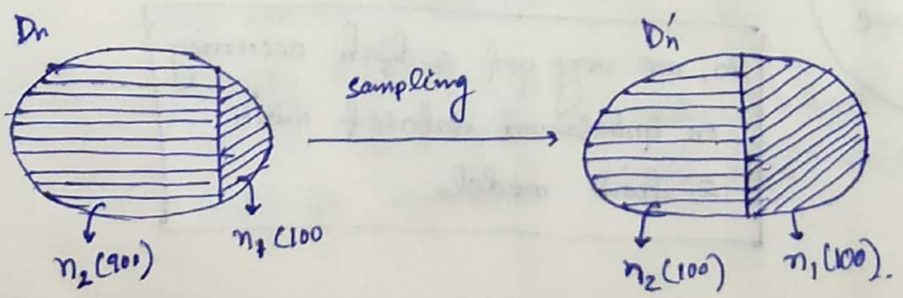
Imbalanced dataset may create problem.

* Just because there are more pts. of the majority class, it may have some advantage. (more biased towards it).

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

→ ① Undersampling

$$\begin{tikzcd} & n_1 : 100 \ (+ve) \\ D_n \arrow[ur, dash] \arrow[r, dash] & n_2 : 900 \ (-ve) \end{tikzcd}$$



So, basically we sample n_1 -ve pts to D_n'.
Now, dataset D_n' is Balanced
further modelling done on D_n'.

Problem with undersampling

$|D_n'| << |D_n|$, the model mayn't work that well as we are throwing away a lot of data.

② Oversampling

$$\begin{tikzcd}
 & 900\ (-ve) & & & 900\ (-ve) \\
D_n \arrow[ur] \arrow[dr] & & \arrow[r, "over", "sampling"'] & D_n' \arrow[ur] \arrow[dr] & \\
 & 100\ (+ve) & & & 900\ (+ve)
\end{tikzcd}$$

$\rightsquigarrow$ each pt repeated 9 times.

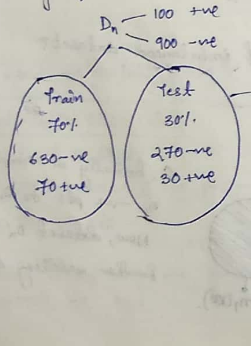
Simply repeating the existing pts is one of the very simplest techniques.

* There are ~~other~~ ways like "extrapolation", where we create new synthetic pts in the region bounded by the original set of pts of that class.

* Another way of implementing repitition of pts instead of manually repeating is by assigning class weights.

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

* Another major problem of imbalanced dataset. →



→ Even if our model says every pt is −ve, accuracy = 90%.

$$\boxed{\text{So, we may get a high accuracy on imbalanced dataset with a dumb model.}}$$

Imbalanced datasets are very much prevalent in real-world scenarios.

ex→	medical :−	10% cancer
		90% non-cancer.
	e-commerce :−	10% buy
		90% don't buy.

(II) Multi-class Classification

There are classifiers like K-NN which can be easily extended to do multi-class classification, there are other like logistic regression that can't. not do multi-class classification easily.

So, given a multi-class classification problem, can we convert it to a binary classification problem.?

↳ let $y_i \in \{1, 2, 3, \dots c\}$.

① $$\begin{tikzcd}
 & \{(x_i, y_i) \mid y_i = 1\} \arrow[r] & (+)ve \\
D_n \arrow[ur] \arrow[dr] & & \\
 & \{(x_i, y_i) \mid y_i \neq 1\} \arrow[r] & (-)ve
\end{tikzcd}$$

(This now can be handled by a simple binary classifier.), or not.

Similarly for each of the c-classes, we can build c-classifiers (binary)

So, multiclass classification problem ⟶ 'c' binary classification problems

This technique is called 1 Vs rest

(III) Given similarity matrix

There will be cases when we can get only the similarity matrix $S_{n \times n}$ but can't get vector featurization of data points.

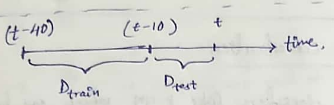
S_{ij} = similarity (x_i, x_j), KNN works great as ultimately from the given D_n, we care about $S_{n \times n}$ only in KNN.

We'll see how other models like logistic regression, NB etc could be extended to be used with S or not.

(iv) Train and Test set Differences

D_n — D_{train}
random splitting
D_n — D_{test}.
(x_i, y_i).

In random splitting, each pt has equal probability for going to D_{train} & D_{test}.
So, both D_{train} & D_{test} will be similar.

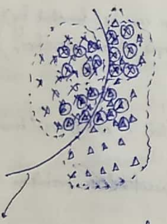


D_n — D_{train}
Time based splitting.
D_n — D_{test}

Ex→ Amazon food reviews.

In TBS, D_{train} & D_{test} could get very different.
Ex→ Suppose some old category of products got removed & new category got introduced.

Graphical representation



x :- train (-)ve
Δ :- train (+)ve.
⊗ :- test (-)ve
Ⓐ :- test (+)ve.

x D_{train} & D_{test} being fundamentally different causes the distributⁿ of (+)ve data change for D_{train} to D_{test}.
(-)ve.

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

CV error – low ?
Test-error – high } — We need to check whether D_{train} & D_{test} come from same distribution. or not.

How to determine this?

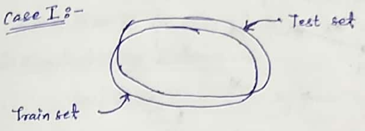
→ ① Time split the data to test & train sets. (D_{test} & D_{train}).

② $D_{train}' \longrightarrow \; x_i = (x_i, y_i)\,, \; y_i = 1$.
$D_{test}' \longrightarrow \; x_i = (x_i, y_i)\,, \; y_i = 0$. } $\longrightarrow$ D_n'

That is, basically level all the training set pts as 1 class & test set pts as another. This gives D_n'.

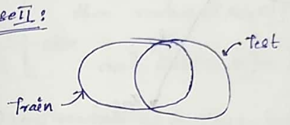
③ Now train a binary classifier on D_n'.

Case I :-



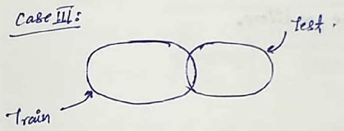
Almost overlapping.
So, accuracy of binary classifier is low.
⇓
distributions similar.

Case II :



less overlap.
Medium accuracy of classifier
⇓
distributⁿ not very similar.

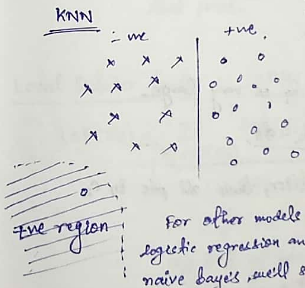
Case III :

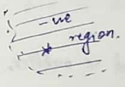


very little overlap.
High accuracy of classifier
⇓
distributⁿ very different.

(v) Impact of outliers

KNN





For other models like logistic regression and naive bayes, we'll see the impact later.

Next we'll know detecting outliers & remove them.

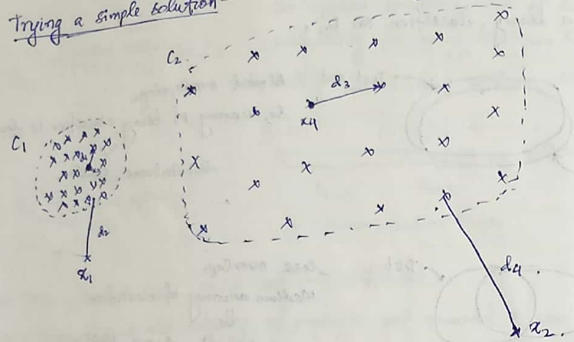
(K = 1) small.

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

	accuracy	
k = 1 →	97%	
k = 2 →	97%	
k = 3 →	97%	
k = 4 →	97%	→ less prone to outliers
k = 5 →	95%	
k = 6 →	92%	

Defecting Outliers – consider only pts ~~[illegible]~~ corresponding to a single class while doing outlier detection.

Trying a simple solution –



So, we are currently only considering (–)ve class pts.

C_1: dense cluster. x_1 & x_2 are outliers.

C_2: sparse cluster.

Simple sol^n algo:–

1. for every pt x_i, compute its k(say 5) NN.
2. calculate avg dist of x_i fm its k-NN.
3. sort x_i's by the avg. dist.
4. if avg-dist is high ⇒ the pt is outlier.

using this, x_2 will be declared outlier as d_4 is very large.
But x_1 won't be declared outlier as $\underline{d_2 \simeq d_3}$.

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

So, we need a better algo.

Some terminologies

1. **K-distance(x_i)** ⟶ distance fm x_i to its k^{th} nearest neighbour.
2. **Neighbourhood of x_i [$N(x_i)$]**
   ↳ set of all pts that belong to k-NN of x_i, which can be more than 'k' in cases of tie.
3. **Reachability distance** →

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

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

4. **Local reachability density :– lrd(x_i)**.

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

⟶ avg reachability distance of x_i fm its neighbours.

lrd(x_i) = inverse of avg reachability distance of x_i fm its neighbours.
↳ Its calculating some sort of density of the surrounding around that point.

Local Outlier factor :– LOF(x_i) – inspired fm KNN.

$$\text{LOF}(x_i) = \frac{\displaystyle\sum_{x_j \in N(x_i)} \text{lrd}(x_j)}{|N(x_i)|} * \frac{1}{\text{lrd}(x_i)}$$

↓ avg lrd of pts in 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.

i.e. density of pts in neighbourhood is high but density of pt itself is low.

So, $LOF(x_i) \rightarrow$ large $\rightarrow$ outlier $\longrightarrow$ density of x_i smaller compared to its neighbours.
$\quad\quad\quad\quad \hookrightarrow$ small $\rightarrow$ in-lier.

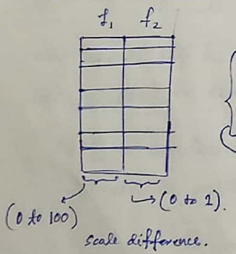
So, considering the 1st figure,
density of x_3 seems same as its neighbours.
Same for x_4. Hence, both are inliers.

But density of x_1 & x_2 are very less as compared to their neighbours.
Hence, outliers.

* There is no hard and fast rule to set threshold for LOF value beyond which there will be outliers.
* But if we know fm domain knowledge, that 5% of pts are outliers then we can sort & remove top 5% pts.
* $LOF(x_i)$ can be as large as infinite. So, there is less interpretability. It would have been better if $0 \le LOF(x_i) \le 1$ which could possibly be interpreted as – probability of a point x_i to be an outlier.

There are many other outlier detection techniques apart fm LOF.

(VI) Scale and Column Standardization



$f_{1\,(0-100)}$ $f_{2\,(0-1)}$.

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

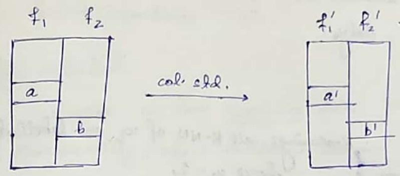
$dist(x_1, x_2) = 5$. The values of feature f_1 can very max by 100 units, but are varying only 5 units.

$dist(x_1, x_3) = 0.8$. values of f_2 can very max by 1.0 & are varying here by 0.8. (Significant)

So, logically $d(x_1, x_3) >> d(x_1, x_2)$

But due to scale difference, we are getting contradicting results.

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



f_1' f_2' $\longrightarrow$ transformed features.

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

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

$mean(f_1) = \mu_1$ $\quad mean(f_2) = \mu_2$
$std\text{-}dev(f_1) = \sigma_1$ $\quad std\text{-}dev(f_2) = \sigma_2$.

$mean(f_1') = mean(f_2') = 0$.
$std\text{-}dev(f_1') = std\text{-}dev(f_2') = 1$.

* If we are using KNN with euclidean distance then 1st do col-std. Then apply KNN on transformed features.
* There are models like decision trees that are scale independent also.

(VII) Interpretability

Consider a medical domain problem :- cancer detection.

Dataset $\rightarrow$ f (model).
Training

$$\begin{tikzcd} x_q \text{ query} \arrow[r] & f \text{ (model)} \arrow[r] & y_q \text{ (label)} \arrow[r, "\text{read by}"] \arrow[d] & \text{doctor} \arrow[r, "\text{conveyed to}"] & \text{patient.} \\ & & \text{[cancer or not]} & & \end{tikzcd}$$

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 blackbox 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.

Ex $\rightarrow$ say x_q is of dimension 10, where each feature corresponds to result of a medical test.

O/p. by interpretable model $\longrightarrow$
$f_5 \rightarrow$ test 5 $\rightarrow$ value is very high
& $f_8 \rightarrow$ test 8 $\rightarrow$ value low
} Hence patient has cancer.

Doctors understand this additional information as these are
results of medical tests only.

Let's take KNN for example.

$x_q \rightarrow \text{model} \rightarrow \underline{y_q = 1}$ Reasoning: all K-NN of x_q are labelled 1,
hence $y_q = 1$.

Let's say d & k, both are small { d = 10, k = 7, example }.

The model (KNN) can also give the 7 nearest neighbours.
So, basically the doctor gets to know about 7 other patient's 'medical
test' result for 10 ~~sets~~, tests, which are similar to query patient.

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

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

(viii)
Feature importance and forward selection

Suppose we are solving the problem of height prediction of an individu.
We are given input features — weight, hair_color, hair length, skin color,
gender, country, ~~etc~~.

Clearly, all the features aren't equally important.
~~[illegible]~~ 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'll be useful
for us in understanding the model better, hence increasing interpretability.

* KNN in its native form doesn't give feature importance.

* In models like logistic regression and decision tree, we can get
feature importance

Feature Selection

Suppose we have 1000-dim 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 (assume).
PCA & t-SNE preserve distances. 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 & 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 new feature selection.

Q? why can't we just calculate accuracy for each single feature, sort
them in descending order and then take 1st d' features?
→ say f_{10} & f_6 are 1st & 2nd most accuracy features.
But it's 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 features,
which new feature adds the most value.

Ex → Suppose $f_1, f_2 \ldots f_{10}$ — 10 features total.
→ After going through each feature individually; say we get f_{10} with
highest accuracy.
* Next we'll train the model with features f_i, f_{10} $i \in [1, 10]$ & $i \neq 10$.
Say we found f_5, f_{10} having highest accuracy. Similarly we'll
proceed.

(IX) Categorical & ordinal features

Real valued data $\longrightarrow$ readily used ✓

Text $\longrightarrow$ BoW, tf-idf, word2vec.

categorical data $\rightarrow$??

Ex $\rightarrow$ hair color $\rightarrow$ {black, brown, red, grey}.

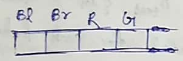
① Give a number.

{black, brown, red, grey}
$\;\;1\quad\;\; 2\quad\;\; 3\quad\; 4.$

But numbers have an inherent order, i.e.
~~[illegible]~~ $1 < 3$

But black < red is absurd.

② One-hot encoding



Bl: | 1 | 0 | 0 | 0 |

R: | 0 | 0 | 1 | 0 |

Br: | 0 | 1 | 0 | 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 colors should be.

Disadvantage $\rightarrow$ #country $\cong$ 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 o/p value for that category.

This is very much problem specific.

④ domain knowledge.

let's say frm domain knowledge we know that ~~[illegible]~~ as distance frm equator inc↑, height ↑.

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

Ordinal feature

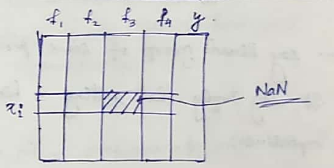
feature:- how do you like Indian food?

$\longrightarrow$ v.good.
good.
avg.
bad
v.bad.

} logical ordering So, we can assign them numbers.

Ⓐ [5, 4, 3, 2, 1] or Ⓑ [10, 6, 4, 3, 1] - Both are OK ✓ as long as order is maintained.

Missing value: imputation
↳ very common in real world.
due to $\rightarrow$ data corruption
$\;\;\;\;\;\;\;\;\;\;\hookrightarrow$ collectⁿ.



① Imputation
↳ mean/median/mode.

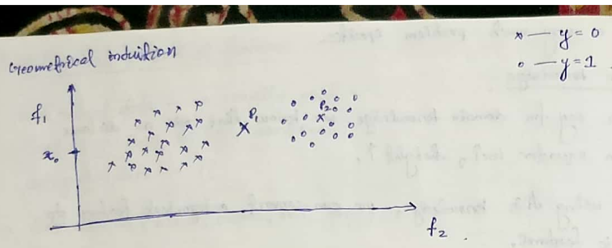
Take all the non-missing value in that col. & find its mean/med/mode & substitute this to all missing values.

sklearn.preprocessing.Imputer

② class based imputation

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

Geometrical intuition



let's say for x_0, f_2 is missing. $y = 1$ for x_0

* If we take mean of ~~[illegible]~~ f_2 of all pts, then x_0 will be at P_1 & if we only consider pts with $y = 1$ then x_0 will be at P_2

* Intuitively, $P_2 \rightarrow$ makes more sense.

③ New missing value feature

In this we treat missing values as source of information

Ex— say blood group of some persons is missing.

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

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.

Ex

	f_1	f_2	f_3	f_4	f_1'	f_2'	f_3'	f_4'
x_1	✓	✓	NaN	✓	0	0	1	0

✓

④ model based imputation

↳ usually KNN used.

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

→ say for a bunch of pts, f_3 missing.

→ treat f_3 as o/p label (y).

→
$$\begin{tikzcd} D_n \arrow[r, "f_3 \text{ present}"] \arrow[dr, "f_3 \text{ missing}"'] & D_{train} \\ & D_{test} \end{tikzcd}$$

Now train a model & predict values.

Curse of dimensionality → (wiki article).

↳ weird things that happen when d is very high. that are not common-sensical.

for Binary features, # features = 3 → # data pts = 2^3.

features = 10 → # data pts = 2^{10}

So, no of datapts inc↑ exponentially with dim.

And to perform well, the model requires data pts from all over the place.

Hence, with inc↑ dimensionality, #datapts required to perform good classification/regression increases exponentially.

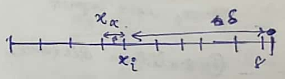
① Hughes phenomenon

For fixed ~~[illegible]~~ no. of data pts, performance↓ with dim↑.

② Distance functions (euclidean dist).

The intuition that we have of distances in 2-D, 3-D is not valid for high dimensional spaces.

Consider 1-D space → n-random pts



Consider x_i. dist_min(x_i) = dist ~~of~~ fm ~~[illegible]~~ closest pt to x_i to x_i. = β.

Similarly, dist_max(x_i) = δ.

So, intuitively

$$\frac{dist\_max(x_i) - dist\_min(x_i)}{dist\_min(x_i)} > 0$$ when 1D, 2D or 3D.

But mathematically it can be proven that as

$$\left[\lim_{d \to \infty} \frac{dist\_max(x_i) - dist\_min(x_i)}{dist\_min(x_i)} \longrightarrow 0\right]$$

$$\Downarrow$$

$$dist\_max(x_i) \approx dist\_min(x_i).$$

So, in higher dimensions, every pair of pts are roughly equally distant.

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

KNN $\xrightarrow[\text{on}]{\text{relies}}$ Euclidean distance. $\xrightarrow[\text{dim}]{\text{in high}}$ makes no sense $\downarrow$ KNN doesn't work well.

Solⁿ:- use other distances like cosine-similarity. Its also affected with increasing dim, but not ~~as~~ much as compared to euclidean.

* If data has high dim and dense $\longrightarrow$ curse of dim high↑.
* If data has high dim and sparse $\longrightarrow$ impact of dim lower↓.

The above limit has been done considering
① uniform distribution of pts randomly. In sparse data, most pts ~~~~ will have no component in many directions. Hence, less impact
② euclidean distance. – If other distance considered, less impact

⑧ <u>Overfitting and underfitting</u>
$d\uparrow \longrightarrow$ overfitting ↑.

Solⁿ of high dimensions.—

① forward feature selection:– classification oriented, – use class labels.
② dim-reduction:– PCA, t-SNE – don't use class labels.
└ not classification oriented.

③ using cosine similarity instead of euclidean distance.
④ sparse representation instead of dense representation.

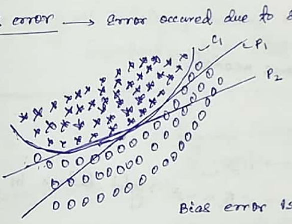
<u>Bias variance trade-off</u>

Generalization error (error on future unseen data) = Bias^2 + variance + $\underbrace{\text{irreducible error.}}$ └ (Proof in linear regression lesson).

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

<u>Bias error</u> $\longrightarrow$ Error occured due to simplifying assumptions.

Ex



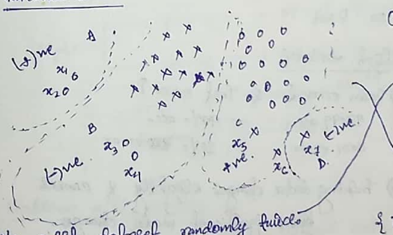
curve makes sense
If we simplify the assumption to a line / plane, $\longrightarrow$ introduces error
$\downarrow$
Bias error.

Bias error is also called underfitting.

In KNN, if <u>k = N</u>, we simplify the assumption to assigning dominant class label to 'x_q'.

<u>Variance error</u> $\longrightarrow$ how much a model changes as training data changes.

① D_n $\longrightarrow$ D_{Train1}, D_{Test1} {Random split – 1}.

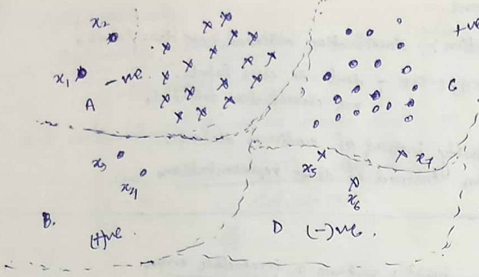


In D_{Train1}, x_1, x_2, x_7 ✓.
x_3, x_4, x_5, x_6 – In D_{train2}.

These are decision boundary for D_{Train1}.

{Geometrically a model is nothing but decision surface}.

We split dataset randomly twice.
Assume <u>k = 1</u>



x_3, x_4, x_5, x_6 — D_{train2}

So, just by randomly splitting 2 times, we get 2 very different decision surfaces (model).

Hence high variance for $k = 1$, $\Downarrow$ overfit model.

Ultimate goal → low generalization error.

$$\text{Gen. error} = \underbrace{\text{Bias}^2\downarrow}_{\text{no underfit}} + \underbrace{\text{var.}\downarrow}_{\text{no overfit}} + \text{irreducible error.}$$

k=1 → Bias ↓ var ↑.
} —— Balance needed.
k=n → Bias ↑ var ↓.

k=1 ;− 10 + 100 + 3 → 113.
k=5 ; 12 + 10 + 3 → <u>25</u> ⇝ right balance in between.
k=n ; 100 + 2 + 3 → 105

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

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

① **high-bias**

train error ↑ ⇒ high bias.

train error ↓ ⇒ low variance.

② **high variance**

ⓐ Train error ↓ & Test error ↑.
99.99 acc. 90% acc.
0.01 error 10% ~~[illegible]~~ error

ⓑ training data changes slightly & causes ~~huge~~ significant model change.

Best and worst cases (KNN).

① dimensionality
- not-large ✓
- large
  - curse of dim. (esp. euclidean dist).
  - interpretability ↓.
  - runtime of kd-tree/LSH ↑.

② low latency → quick results.
KNN shouldn't be used. → high runtime complexity.

③ find right distance measure, then KNN works well.
For text data, cosine-distance ✓

If we are given similarity/distance matrix, then use KNN.

* For every ML technique, knowing where it works best and worst is important, which will help in applying them better in future.

Performance Measurement of Models

Some classification metrics

$$\text{Accuracy} = \frac{\#\text{pts correctly classified}}{\text{Total } \#\text{pts in } D_{test}}.$$

Accuracy $\in [0, 1]$ — 0 (bad), 1 (good).

↓ easy to understand measure.

Measured on test set usually.

100pts —
- 60 +ve → 53 +ve, 7 −ve
- 40 −ve → 35 +ve, 5 −ve.

error = 12. correctly = 88.

$$\text{Accuracy} = \frac{88}{100} = 0.88.$$

Problems with accuracy

① Doesn't work well with unbalanced data.

Scanned by CamScanner