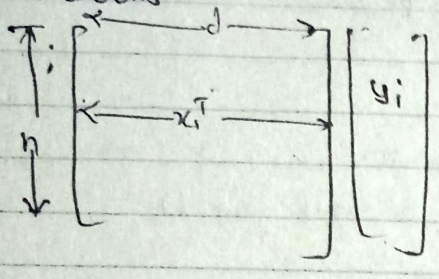
* Classification and Regression models K-NN

Classification:- given a new data points, predict its category
→ classification model.

$y = f(x)$
- y ← category
- x ← data point

classification is all about finding the func^n.

→ Notations



$D = \{(x_i, y_i)_{i=1}^{n} \mid x_i \in R^d, \; y_i \in \{0,1\}\}$

D ← Dataset

set of pair x_i, y_i where $x_i \in R^d$ and $y_i \in \{0,1\}$

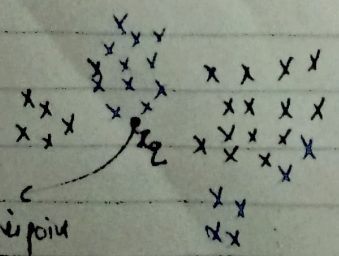
If y have only 2 values then it is called 2 class classifn or binary classification

In MNIST, $y \in \{0,1,2,3,4,5,6,7,8,9\}$ ∴ 10-class classification also called multi class classification problem.

But what if y_i is know more than to smallest finite set of classes and y_i can have any Real value. In that case classification Not work it In that case we use Regression.

→ K-NN Geometric intuition

2D-toy dataset: binary classification



x: +ve data point

x: -ve data point

$D = \{(x_i, y_i) \mid x_i \in R^2, y_i \in \{0,1\}\}$

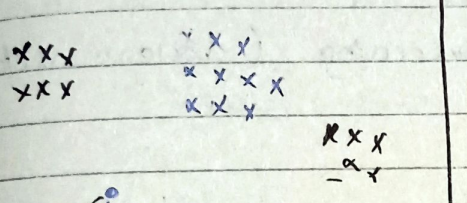
Take all the points which are close to x_q:

As we can see that, all the neighbours of x_q are +ve we can conclude that x_q is +ve.

step 1: find K-nearest point to x_q in D

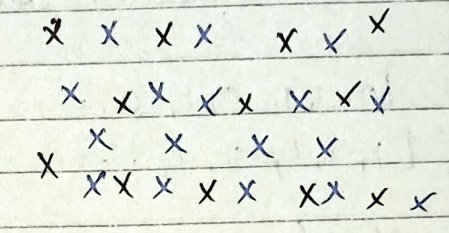
step 2: chose the majority y_i among neighbours and declear the y_i as y_q (class of x_q)

→ failure cases of K-NN



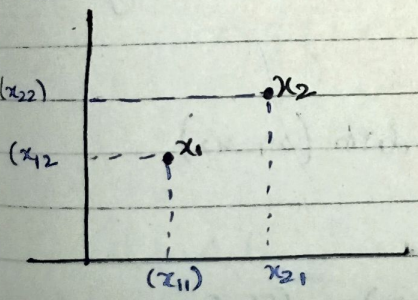
↳ farther from every node

And here it becomes hard to predict the class of these type of points



↳ In these types of plot it is hard to work K-NN.

* Distance Measures :-



$x_1(x_{11}, x_{12})$ $x_2(x_{21}, x_{22})$

Euclidian distance

$$\Rightarrow d = \sqrt{(x_{21}-x_{11})^2 + (x_{22}-x_{12})^2}$$

$\|x_1 - x_2\|_2$ → L_2 Norm (Euclidian dist is also called L_2 Norm)

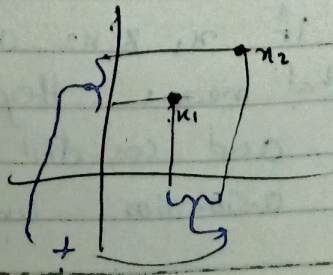
$\|x_1\|_2$ ⇒ distance of x_1 from origin

→ Manhattan's distance:

$$\sum_{i=1}^{d} |x_{1i} - x_{2i}|$$

This is L_1 Norm hence written as

$\|x_1 - x_2\|_1$



Lp norms → Minkowski distance.

$$||x_1 - x_2||_p = \left(\sum_{i=1}^{d} |x_{1i} - x_{2i}|^p \right)^{1/p}$$

If $p=2$, Minkowski distance = Euclidian distance
If $p=1$, Minkowski distance = Manhattan distance,

Lp Norm of single vector: $||x_1||_p = \left(\sum_{i=1}^{d} |x_{1i}|^p \right)^{1/p}$

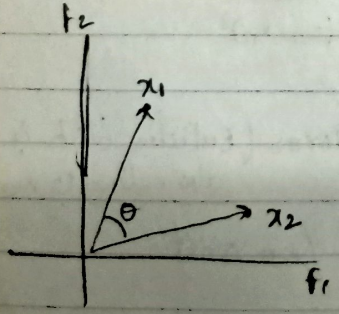
→ Hamming distance:- used in text processing (boolean vector)

$x_1 = [0, 1, 1, 0, 1, 0, 0 \ldots]$
$x_2 = [1, 0, 1, 0, 1, \ldots]$

Hamming dis (x_1, x_2) = # loc where binary value mismatch.

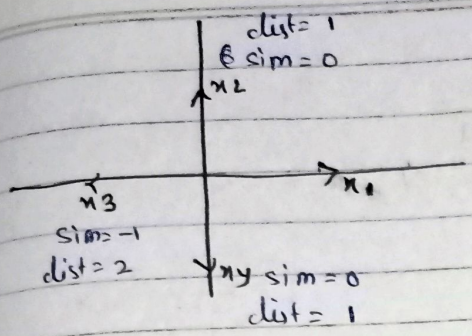
→ Cosine - similarity & cosine - distance:-
a similarity distance
If similarity increase distance decreases. They are inversely proportional.

$$1 - \text{cosine-similarity}(x_1, x_2) = \text{cosine-dista}(x_1, x_2)$$



cosin-sim $(x_1, x_2) = \cos\theta$
· θ = angle b/w x_1 & x_2

If x_1 & x_2 are in same dir then cos_sim $(x_1, x_2) = 1$
that mean they are very much similar.
and cos-dist = 1 - cos_sim = 1 - 1 = 0
Here, cosin-sim used give the angular distance.



$$\text{cos\_sim}(x_1, x_2) = \cos(\theta) = \frac{x_1 \cdot x_2}{\|x_1\|_2 \|x_2\|_2}$$

If x_1 & x_2 are unit vec : Hen $\|x_1\|_2 = \|x_2\| = 1$

$\cos\theta = x_1 \cdot x_2$ if x_1 & x_2 are unit vector.

what is Relationship b/w Euclidian dis and cosine sim

$$\|A - B\|^2 = (A-B)^T (A-B) = \|A\|^2 + \|B\|^2 - 2A^T B$$

If A & B are unit vector He $\|A\|^2 = \|B\|^2 = 1$

$$\|A-B\|^2 = 2 - 2A^T B$$
$$= 2(1 - A^T B)$$
$$= 2(1 - A \cdot B)$$
$$= 2(1 - \text{cos\_sim}(A, B))$$

∴ If A, B are unit vector Hen:

Square of Euclidi dis $\|A_0 - B\|^2 = (A-B)^T (A-B)$.
$= 2(1 - \text{cos\_sim}(A, B))$
$= 2 * \text{cos\_dist}(A, B)$

☆ ~~Measuring how good K-NN is:~~

☆ Time & space complexity:

Input: D_{Train}, K, $x_q \in R^d$;

o/p: y_q

KNNpts = [ ]

$O(n)$ → for each x_i in Train:

$O(d)$ → - compute $d(x_i, x_q) \rightarrow d_i$

- Keep the smallest K-distances (x_i, y_i, d_i) ↳ put this in KNNpts

↳ dimensions / # of features

$O(n*d)$

cnt = 0 ; cnt_neg = 0;

for each x_i in Knnpts

if y is +ve

cnt_pos ++

else

cnt_neg ++;

if cnt_pos > cnt_neg

store $y_q = 1$

else

store $y_q = 2$

Time : $O(n*d)$

space x_q : ~~Train~~ D_{train} should be in m/m

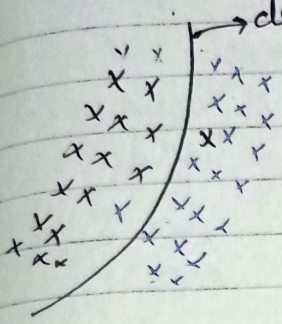
↳ $O(n*d)$

Extra space is not x_q but to store KNN we have to store D_{train}

∴ Time = space = $O(n*d)$

* Limitation of KNN: ☆ High time & space complexity. There are some other implementations of K-NN, like K-d tree & LSH. We will learn them later.

→ How to pick right K in K-NN :- K in K-NN is known as hyper-parameter.



How draw decision surface :- take many query points and check their output and using them draw a decision surface.

as K increase smoothness of decision surface increase.

what if K = n (i.e. ~~none~~ K = total no. of points.

Suppose there are n_1 positive points in data and n_2 negative points

Here if $n_1 > n_2$ then we get positive class for every test data

else we get negative class.

* overfitting & underfitting :-

Overfitting :- when we try to fit every point, extremely non-smooth surface. It may also try to fit outliers.

underfit :- We are really not fitting all data points there are still chance of improvement.

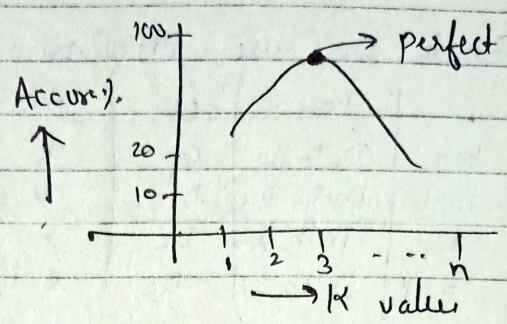
* Need for cross-validation:-

How to determine K

D → Dtrain (70%.)
D → Dtest (30%.)

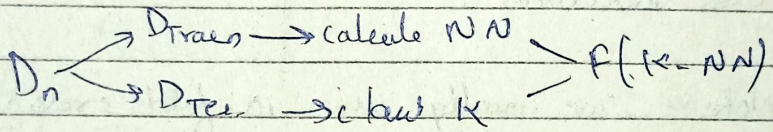
check for several value of K and check accuracy

	Acc
K = 1	0.78
K = 2	0.79
K = 3	0.84
⋮	

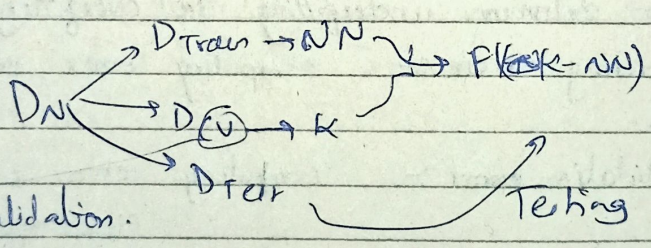


Now, one thing to be noted is we are using D_{Test} to calculate K and now we are testing on same D_{Test} so, results are not reliable.

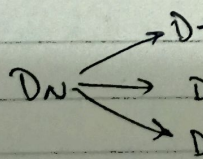
previous



New policy



* K-fold cross-validation

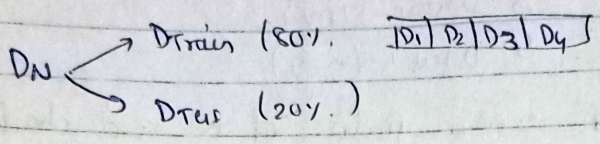


There is a problem we are only using 60 % of data as training. We can't do anything for 20% test but can we somehow use $D_{Train} + D_{CV} = 80\%$ of data. That's where the K-fold cross-validation is used.

Note: (K)-fold : (K) NN

≠ These need not be same.

What we do is we divide the D_{train} (80%) data into 4 equal parts



One by one we take one of D_i as cross valid.

	Train	CV	Acc
K=1	$D_1 D_2 D_3$	D_4	x
K=1	$D_2 D_3 D_4$	D_1	y
K=1	$D_1 D_3 D_4$	D_2	z
K=1	$D_1 D_2 D_4$	D_3	w

→ Take average and this is the value of 1-NN

same with 2-NN, 3-NN 4-NN and choose which NN have max accuracy.

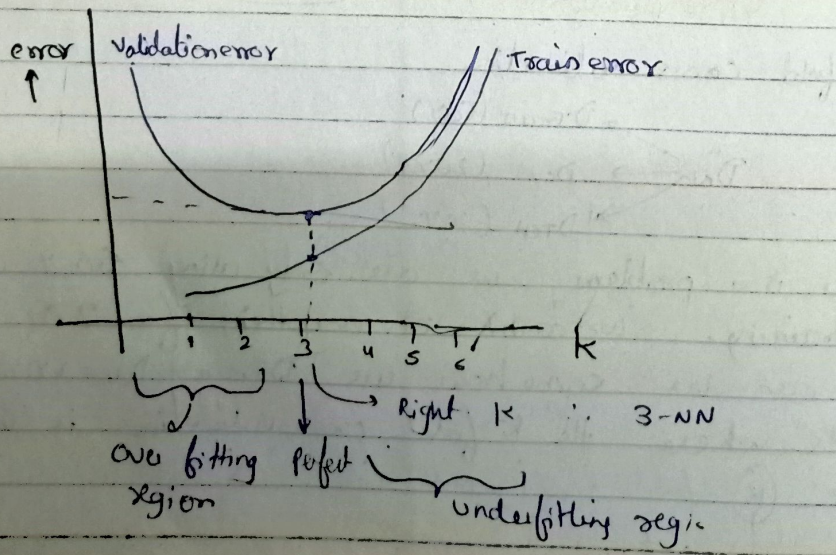
We are divided D_{train} into 4 folds, then it is called 4-fold cross validation.

Note :- we usually use 10-fold cross validation.

⇒ How to determine underfitting and overfitting

Training error :- computing error on Train data points.

validation error :- computing error on validation data.



- If training error and cross-validation error both are high then it signifies : Underfitting
- If train error is low and CV error in high then it is overfitting

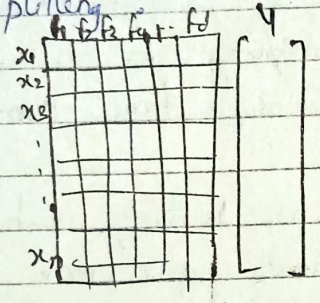
* Time-base splitting

$D_n \rightarrow D_{Train}$, D_{CV}, D_{Test}

Till now we are randomly splitting the data.

But random splitting is not the only way to split. like Time-based splitting.

In Time-base splitting :

$D =$ 

In our amaze food data, there was a col^ of time. so we can order the data based on time.

step1: sort your data on increasing order on time.

step2: choose first 60% as D_{Train}, next 20% as D_{CV} & rest as D_{Test}.

* K-NN for regression:-

Till now we have seen K-NN on classification problem

classificat data $D = \{(x_i, y_i)_{i=1}^{n} \mid x_i \in R^d, y_i \in \{0,1,\dots\}\}$

But in regression y can have any value

Regression Data $D = \{(x_i, y_i)_{i=1}^{n} \mid x_i \in R^d, y_i \in R\}$

step1: Given x_{query}, find K-nearest neighbour.

let neares neigh are $(x_1, y_1)(x_2, y_2) \dots (x_K, y_K)$

In classificatin we took majority vote : y_K : among neigh

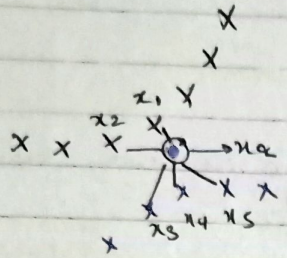
step2: to get y_q from $y_1, y_2, y_3 \dots y_K$

$y_q = \text{mean}(y_i)_{i=1}^{K}$

OR $y_q = \text{median}(y_i)_{i=1}^{K}$

→ Weighted K-NN

Till now we have seen simple K-NN, there is also weighted K-NN.



neighbour of x_q → x_1 x_2 x_3 x_4 x_5

x_1	y_1	$d_1 = 0.1$
x_2	y_2	$d_2 = 0.2$
x_3	y_3	$d_3 = 1.0$
x_4	y_4	$d_4 = 2.0$
x_5	y_5	$d_5 = 4.0$

As we can see that x_1 is very close to x_q. should not we give more weightage to x_1, because there are more chances that x_q have same class label as x_1

In normal K-NN, we are treating each Neighbour equally But logically we should give more importance to closer neighbour.

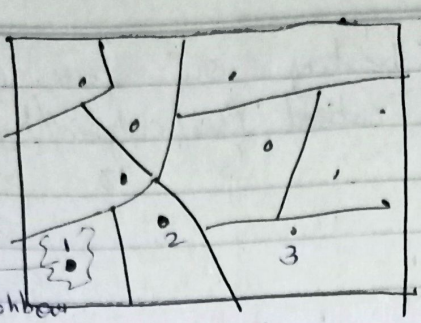
Hence, we can give weights to each point in neighbour. One way is assign weight $w_i = \frac{1}{d_i}$ { If distance increase w decrease & if distance is less then more weightage give

Now, to chose y_q for x_q.

sum up the ~~positive weight~~ weights of each category and choose y_q = category with max weight.

There are many other weighing scheme other than $w_i = \frac{1}{d_i}$

→ Voronoi diagram:- Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane.

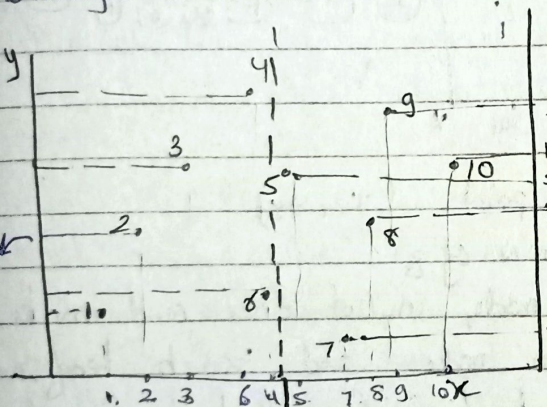


Any point in this region have 1 as its closest neighbour

for K = 1 in K-NN voronoi diagram divide the plane in region s.t any point in that region have closest neighbour as the point of that region

* K-d- tree:- As we have discussed earlier that Time complexity in K-NN = O(nd) * S : O(nd) we also discussed that we have other implementation of K-NN. one of them is Kd-tree which performs better:

Binary search tree can be converted to KD tree:-



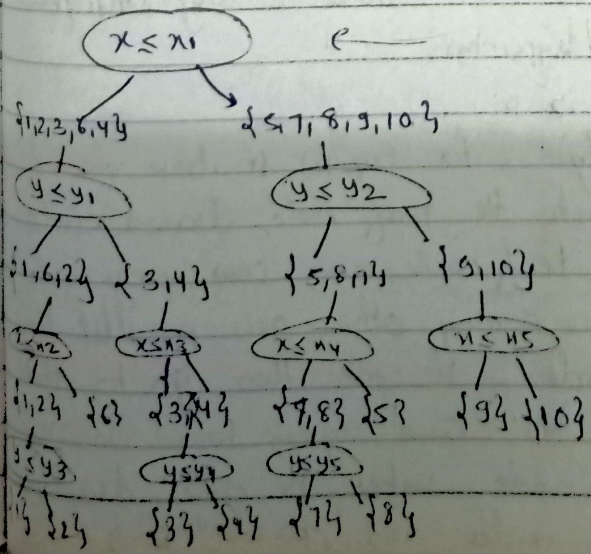
step1: pick first axis ie. x-axis in 2-D. projed each point on x axis. and compute median.

$(x \le x_1)$ Ya. {1,2,3,6,4} No {5,7,8,9,10}

5 point — median = x_1 — 5 point

step2: Now change the axis and perform same operation on child Node.

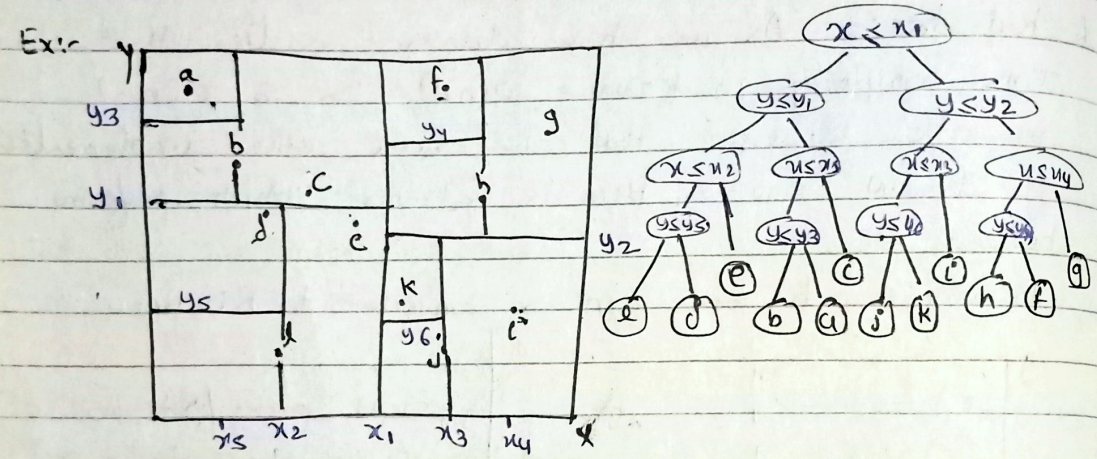
Now again use x-axis.



Actually we are breaking our space using axis-parallel planes into cuboid Rectangle/cuboid/hypercubes
2D 3D >3D

Now, that we have divided our space using Kd-tree strategy we learn how to find nearest neighbours using K-d tree.

Ex:-



x: query point (x_q, y_q)

what is 1-NN of q.

we can start at root node in Kd-tree and make comparison according to node and reach leaf node. Once we reach the leaf node that leaf node is ~~the nearest neighbour of xq~~. to x_q distance is 'd'. Now draw a circle with radius = d and centre as (x_q, y_q). If there is any other point inside the circle / hypershapes
2D >2D

how to do it algorithmically:- for (x_q, y_q) if I have drawn the tree once we reach the leaf node, draw a circle using distance b/w leaf node and query node. Now, check if there is any other axis parallel line is intersecting the circle. If yes then go to that axis parallel node in tree and again I have down until we reach a node where the circle drawn don't intersect other nodes axis parallel axis.

T: $O(\log(n))$ { best case }
$O(n)$ { worst case }

In K-NN
T: $O(K \log n)$ { best case.
$O(K \cdot n)$ { worst case.

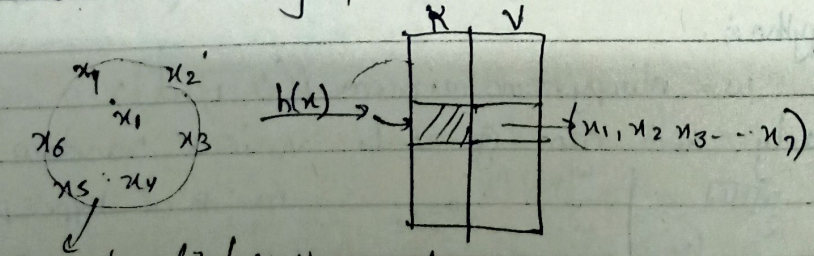
with d-dim.
~~T: $O(K * d * \log n)$ { best ca~~
~~$O(K * d * n)$ { worst~~

* → Limitation of Kd-tree.
In Kd we need to look at adjacent cells to check if they are nearer.
In 2D - we need to check 4 adjacent cells,
In 3D - 8
in nd - 2^d

Due to this Time complexity becomes:
$O(2^d * K * \log n)$ { In best case }
$O(2^d * K * n)$ { In worst case }

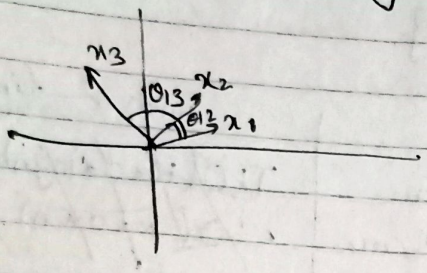
K-d tree is ok for only small dimension.
in bigger dimension Time complex become very large.

* Hashing & Locality sensity hashing
LSH uses hashing to find Nearest Nieghbours.
Let we have bunch of points



Same locality/Neighbourhood.

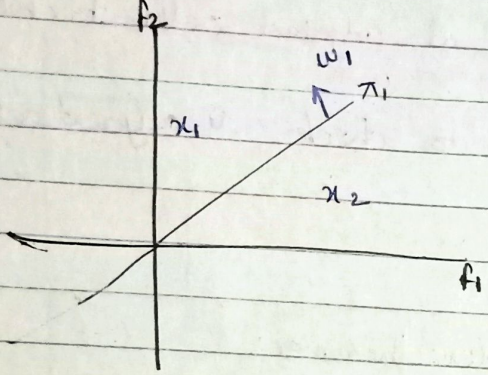
✡ LSH for cosine-similarity:-



$\theta_{12} \sim 0$ here x_1 & x_2 are similar.

K	V
h(x)	x_1, x_2

LSH : It is a randomized algorithm.



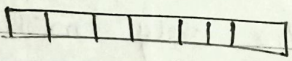
① break the whole space using Random-planes.

$w_i^T x_1 = +ve$ ∴ x_1 is in same dir

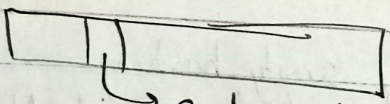
$w_i^T x_2 = -ve$ ∴ ...

How we will generate Random-hyperplane.

w is a d-dim vector



if we can generat a random d-dim vector then it can be treated as Random-plane.

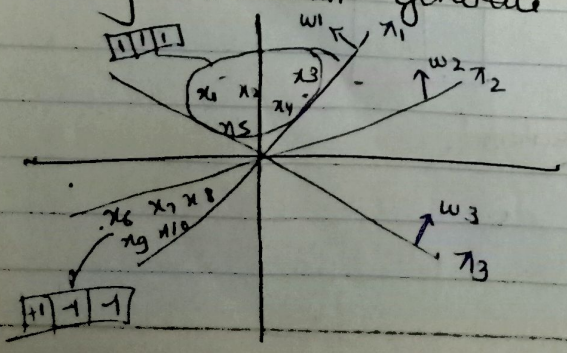


each point is randomly generated from a Normal dist (mean = 0, σ = 1)

// in python.

```
w = numpy.random.normal(0, 1, d);
```

similarly we can generate various random planes.



m = # of hyperplanes

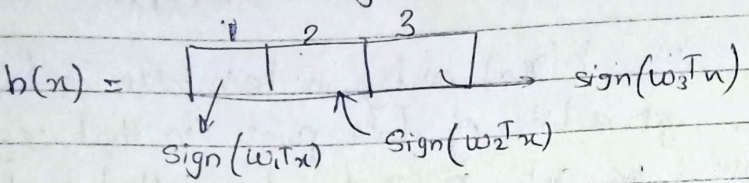
Each point is now in a region.

$x_i^T w_1 \geq 0 \rightarrow +1$

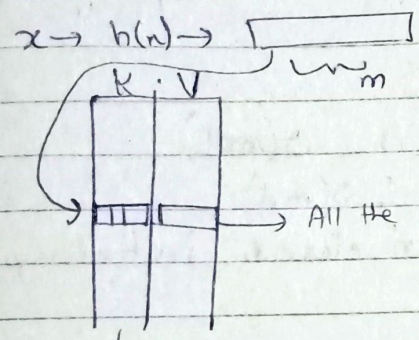
$x_i^T w_2 \geq 0 \rightarrow +1$

$x_i^T w_3 \geq 0 \rightarrow +1$

for every slice or region we create a m dim vector because there are m region

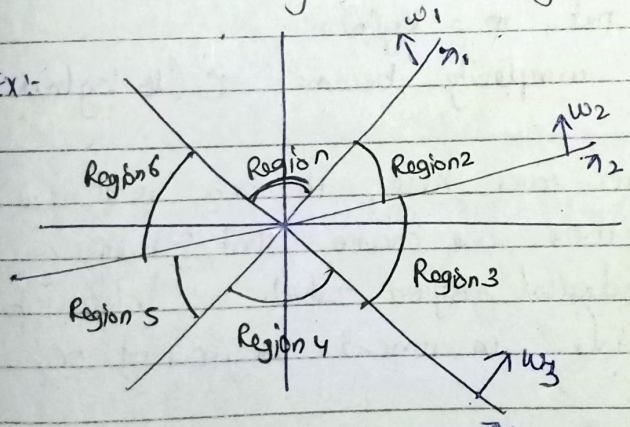


Note



↳ Key is not a single value, it is m-dim vector.

Ex:-



Each region have a m-dim vec

Region	w_1	w_2	w_3
1 →	+1	+1	+1
2 →	-1	+1	+1
3 →	-1	-1	+1
4 →	-1	-1	-1
5 →	+1	-1	-1
6 →	+1	+1	-1

→ These will act as key in hash table and all the points in the region will act as value in hash table.

How much Time & space to construct the hash table:-

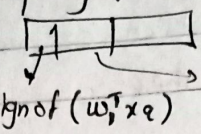
S : O(nd)

n points of d dim

* Every point has to be multiplied with region here
$\sim O(m*n)$

Now, imagine we are given a query point x_q.

Now step 1: find $h(x_q)$ =



use $h(x_q)$ as key in hash table and get the we get a list of data points in that region.

suppose we get n points from that region.
Now we can select k nearest neigh from these n points.

∴ Time of one query $\Rightarrow$ ~~O(md)~~
$x_q \rightarrow h(q)$: $O(md)$
If there are n' elements in that region,
$O(md)$

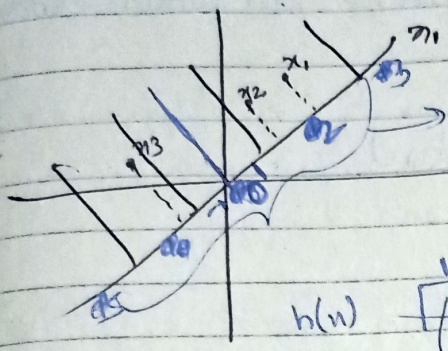
typically we put $m = \log(n)$
hence. Time complexity become $O(\log(n) * d)$

Note: There are cornes cases here, if x_q is at near bound of region then there are chance that there are some points in adjacent region which are closer to x_q than the points in current region of x_q.

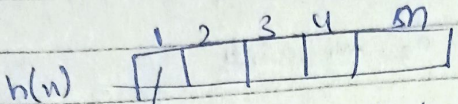
* We can miss a NN in cos-sine as our distance measure. To tackle this problem we need to create multiple hash table with diff diff regions for each hash table and then we need to traverse throw the list of each hash table and get a set of all possible NN and get K-NN.

So final T : $O(L * m * d)$
- L: No. of hash tables.
- m: # of hyperplanes.
- d: dimension/features.

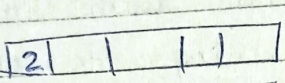
→ LSH for Euclidean distance.

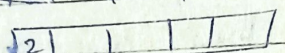


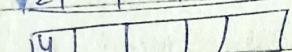
→ breaking the hyperplane in various parts and check how many points are projecting in that part.



→ part in which the projection of point lies in π_1 hyperplane.

$h(x_1)$: 

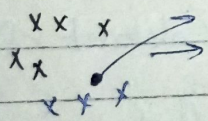
$h(x_2)$: 

$h(x_3)$: 

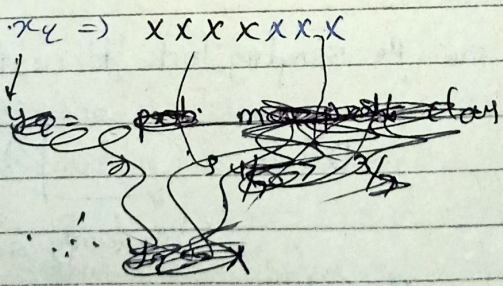
This is the only diff b/w LSH for Euclidean & LSH for cosine.

* LSH is not perfect.

K-NN : probabilistic class label.



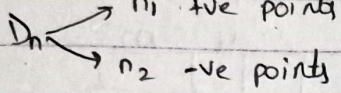
7-NN of x_q =>



y_q =  is +ve class with p_1 prob
−ve " " $1-p_1$ prob.

In this chapter we will learn how to apply classification with some real world cases. because in real world data may be in various forms.

→ Imbalanced vs balanced dataset



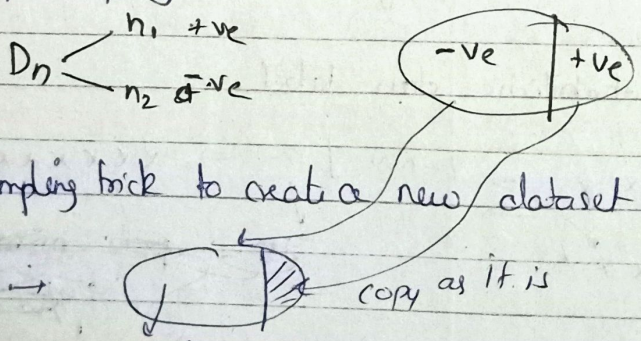
If $n_1 \approx n_2$ then D_n is balanced dataset
If $n_1 << n_2$ or $n_2 << n_1$ D_n is imbalanced dataset

So we do we deal with imbalanced datasets.

Suppose we are using K-NN, we can get biased result. Because there are more chances that all the data points are nearest Neighbours from dominating class. and only few data points from other class.

So, How to deal with imbalanced data set.

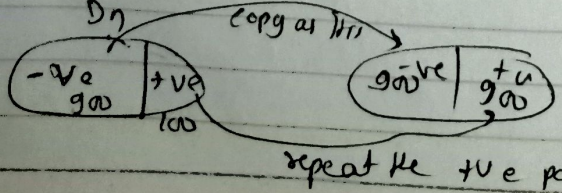
① Under sampling



Now both becomes equal.

But there is one problem with Under sampling : we are not using all the data. we building our model with only small set of data, hence model may not work good.

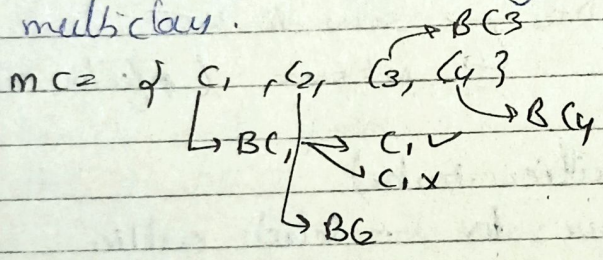
② Over sampling



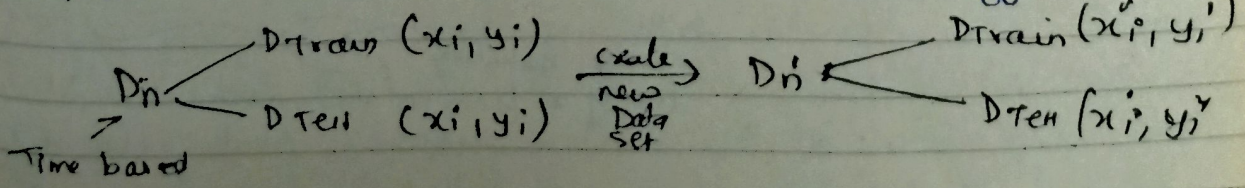
other way of doing over sampling is artificial / synthetic point we create new points using already points.

(iii) class weights :- let we have 100 +ve, 900 -ve so we assign weights to +ve & -ve class
+ve points have weight 9, -ve point have 1

→ multi class vs binary classify.
All the classification algo don't works on multi class like logistic regression works only on binary class
Q) can we convert a multi class problem to binary class:
Ans:- for each class in multi class problem we have to devise one binary classifier for each class of multi class.



→ Test & Train data set diff:-
when we are using Time based division of Data into Dtrain Dtest then there might be changes that Distribution of data may get changed and hence model perform bad. In Dynamic dataset where new point gets added time to time - we need to check if Dtrain & Dtest are following the previous distn or not.
Q) How to check if dtrain & dtest have diff distribution



for D training

$(x_i', y_i') \Rightarrow x_i' = \text{concate}(x_i, y_i)$

$y_i' = 1$

for D test

$(x_i', y_i') \Rightarrow x_i' = \text{concat}(x_i, y_i)$

$y_i' = 0$

Now build a Binary classifier on D'n

If D_{Train} & D_{Test} have different distribution then Binary classifier have high accuracy, as it can seperate well.

If D_{Train} & D_{Test} have same distribution then BC have low accuracy.

D_{Train} ; D_{Test} → same dis ✓

↳ Not same (⇒ feature are not stable).

* LOF (Local Outlier Factor)

It is a technique to detect outlier

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

$N(A)$ = Neighbourhood of A.

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

Local reachability density : $lrd(x_i) = \dfrac{1}{\left\{ \sum_{x_j \in N(x_i)} \left\{ \dfrac{\text{reach-dist}(x_i, x_j)}{|N(x_i)|} \right\} \right\}}$

average reachability dist of x_i from its Neighbour.

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

↳ average lrd of pts in the neighbourhood of x_i

If LOF (x_i) is large, if $lrd(x_i)$ ↓ or this is large.

↓ It is small when there are small points around x_i, i.e. density of x_i is small

↳ this is large when Neighbour of x_i are denser.

So, simply :- If neighbour average density is more but density around x_i is small. Then x_i is outlier.

If $LOF(x_i)$ ↑ x_i is outlier

→ Feature importance and forward selection.

Not all the feature are important for classification or regression. If we somehow sort the feature based on their importance. Then it become very useful. Feature importance is useful to understand the model better.

Feature importance is a numerical score to each feature

* K-NN doesn't provide feature importance (but we can make changes in it to get feature import)

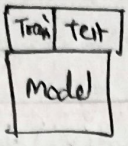
forward feature selection is a technique to get importance of feature importance.

If we have very large dimen/feat say 10000 it becomes important to decrease the dimensionality. Even though we can use PCA, t-SNE for dimensionality reduction. but they don't care about classification task. They just care about distance. Here if we some how know the important feature say 100 out of 10000 we can discard 9900 features

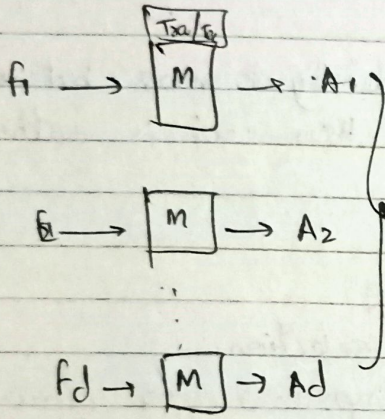
forward feature selection is used for the

$D_n \rightarrow D_{Train}$, D_{Test}

① $\{f_1, f_2, f_3, \ldots f_d\}$



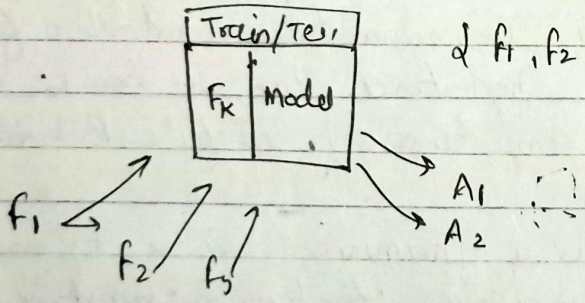
take only first feature and take Train & Test data only corr to that feature. Now, suppose we get some Accuracy $= A_1$



take max accu

Say f_K has highe Accu, then f_K is the most important feat.

② $\{f_1, f_2 \ldots f_{K-1}, f_{K+1}, \ldots f_d\}$



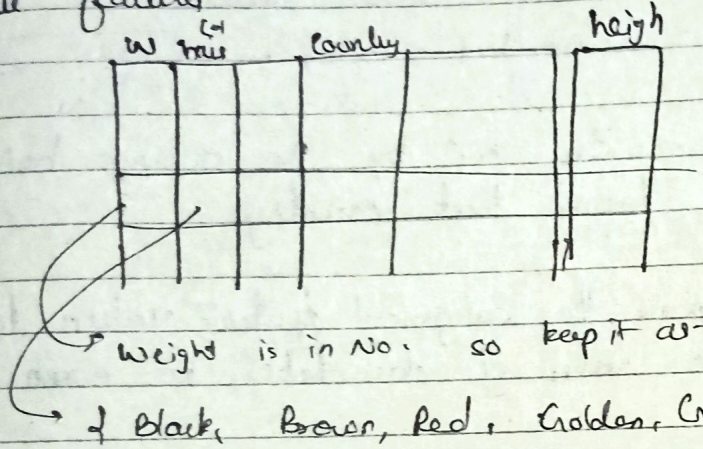
Same step is repeated

Note: One can argue that why we didn't selected Top 100 or K feature in the very first iteration. because we are choosing only Max accuracy feature, why not choose 2 & 3 or top 10 feature from 1st iter. The Ans:- There may be chances that two feature are similar hence their accuracy is high and simil. so we may end up selecting similar features.

In forward selection, we select one feature in each iteration
A similar concept is Backward selection in this we remove the least important feature at each iteration.

Limitation :- Time complexity ↑

* Handling categorical & ordinal features.
suppose we are given a data set of people where we have various features

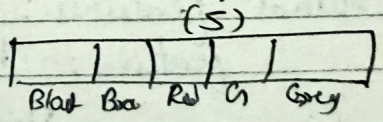


If we assign a number to them like { Black, Brown, Red, Golden, Grey }
1 2 3 4 5

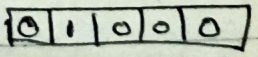
Then there is a problem because numbers comes with an inherent ordering i.e 1>2, 5>4, etc. But we can't say Black > Brown etc. so we cannot directly assign numbers.

→ One-hot encoding : { hair color : { Black, Brown, Red, Gold, Grey }
(5)

So create 5-d { vector



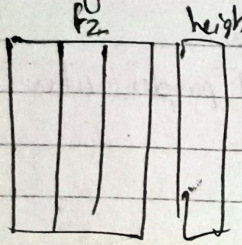
Let x_i = Brown Then its feature encoding



So one-hot encoding creates a binary vector of size of no. of unique values in that feature.
hair color is a categorical feature.
problem with one-hot encoding is some time there are lot of unique values in feature hence feature vector bigger larger.
Ex:- country feature has around 200 values, here we get a 200 size vector.

If the no. of distinct values for a categorical feature is high → then one-hot encoding : It can create sparse and large vectors
↳ because only one val = 1 and others are

(4) Solution for this is



if f2 = country
↳ 200 values
one hot-encoding :-
200

idea :- Replace country col^n by the average height of people from that country.

General idea :- Replace the categorical feature values to the average value of class label in each corresponding category.

But this don't work in every problem.

(5) There is another way :- Using domain knowledge.
Ex :- Country → distance from a chosen country.
(categorical) (numerical)

→ Ordinal features :
feature : how do you rate Indian food.
V. good
good
avg
bad
V. bad
} it is categorical, but one thing is It have ordering. V. good > good > avg > bad.

here we can simply assign them no. .

→ Handling missing values :-

① imputation
- ① mean replacement
- ② median "
- ③ most freq value "

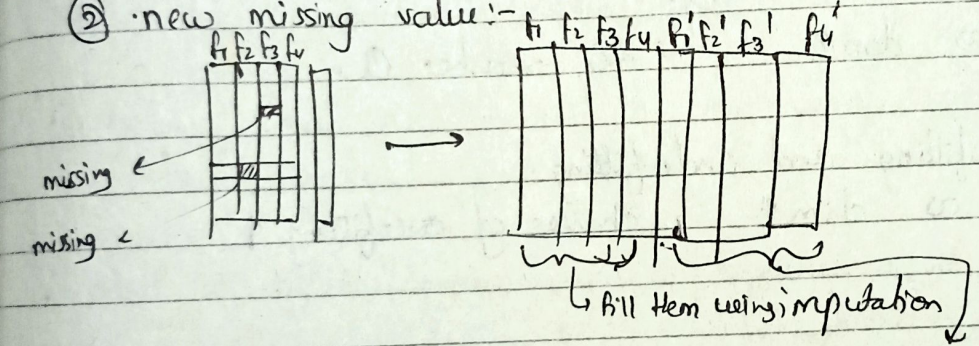
In classification we have to take care.

say $x_i : f_3$ is missing and $y_i = ①$

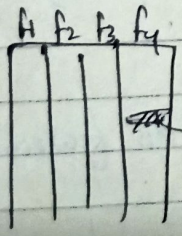
then replace $x_i : f_3$ with mean/median/mode of all the value have class label ①

- we have to find mean/median/mode from same class in classification problem.

② new missing value :-



③ model based imputation



→ we can make a model to predict the value of missing values.

★ Curse of dimensionality

It talks about the problems occurs due to higher dimensionality.

① As dim ↑ total no. of datapoints to perform good classification increase exponentially.

- Hughes phenomenon :- n → size of dataset.
  If n is fixed, performance ↓ as dim ↑

Hughes phenomenon :- on a fixed data set size, performance decrease as dim ↑.

② Distance func^n (specially euclidian distance)
intuition of dist in 3D is not valid in higher dimensions

$$\text{dist min}(x_i) = \min_{x_j \neq x_i} (\text{dist}(x_j, x_i))$$

$$\text{dist max}(x_j) = \max_{x_j \neq x_i} (\text{dist}(x_j, x_i))$$

$$\underbrace{\text{dist max}(x_i) - \text{dist min}(x_j)}_{\text{dist min}(x_i)} > 0 \quad \left.\right\} \text{ for 1D, 2D, 3D}$$

as dim ↑ , this reaches 0. ...

③ overfitting and underfitting :
as dim ↑ , chances of overfitting ↑.

☆ Bias - variance tradeoff :-
mathematical basis to understand overfitting & underfit
is Bias - variance tradeoff :-

generalization error : Error that is made on future unseen (future data error) error

generalized error = Bias^2 + Variance + irreducible error.

① irreducible error :- error that cannot reduced further.

② Bias error :- comes due to simplifying assumption.
high bias = underfitting.

③ Variance :- how much model changes as training data changes.
For example at k=1 in k-NN, model changes rapidly as train data change.

high variance → overfitting.

So, our final goal is low generalisation error in order to achieve this we should have ↓ Bias, ↓ Variance. i.e. to remove overfitting & underfitting.

→ But & worst case of K-NN:-

(i) when dim is small (< 10) → K-NN is good model.
(ii) when dim↑ we get curse of dimensionality, Runtime increase,

So, KNN is good for small dim.

(iii) KNN should not be used in low-latency applica. where we need fast result.

(iii) Find the right distance measure:- K-NN is good.