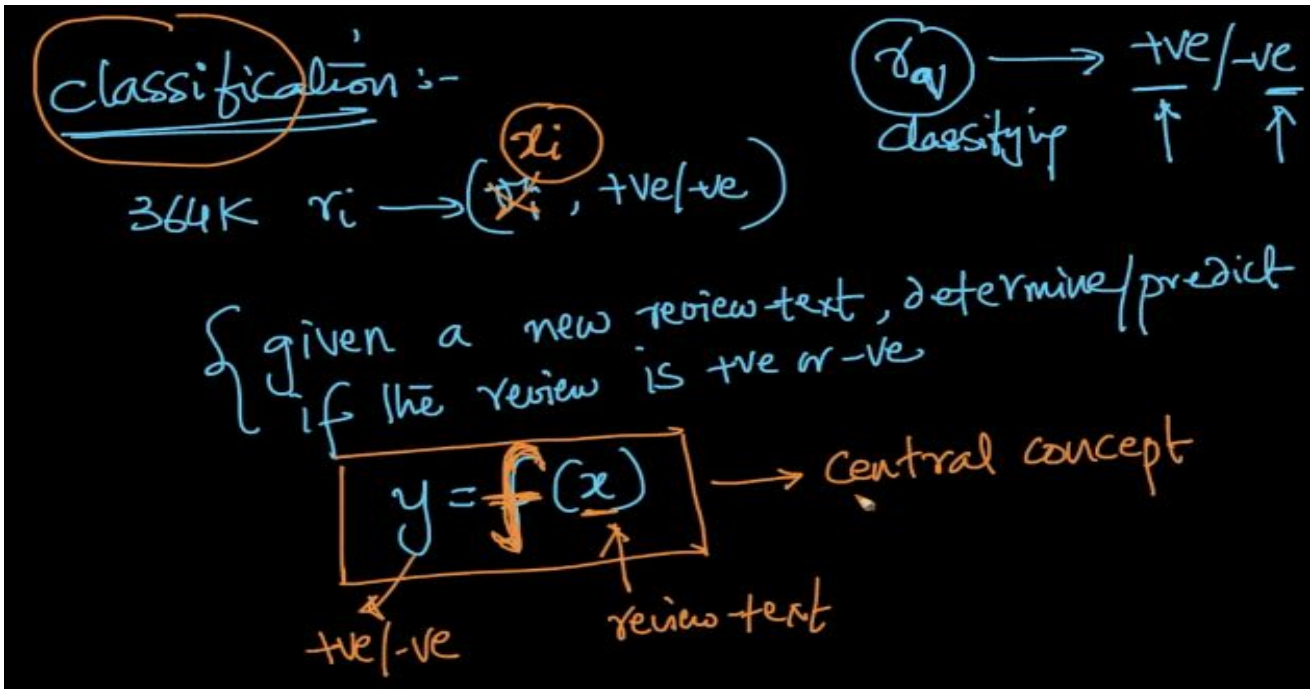
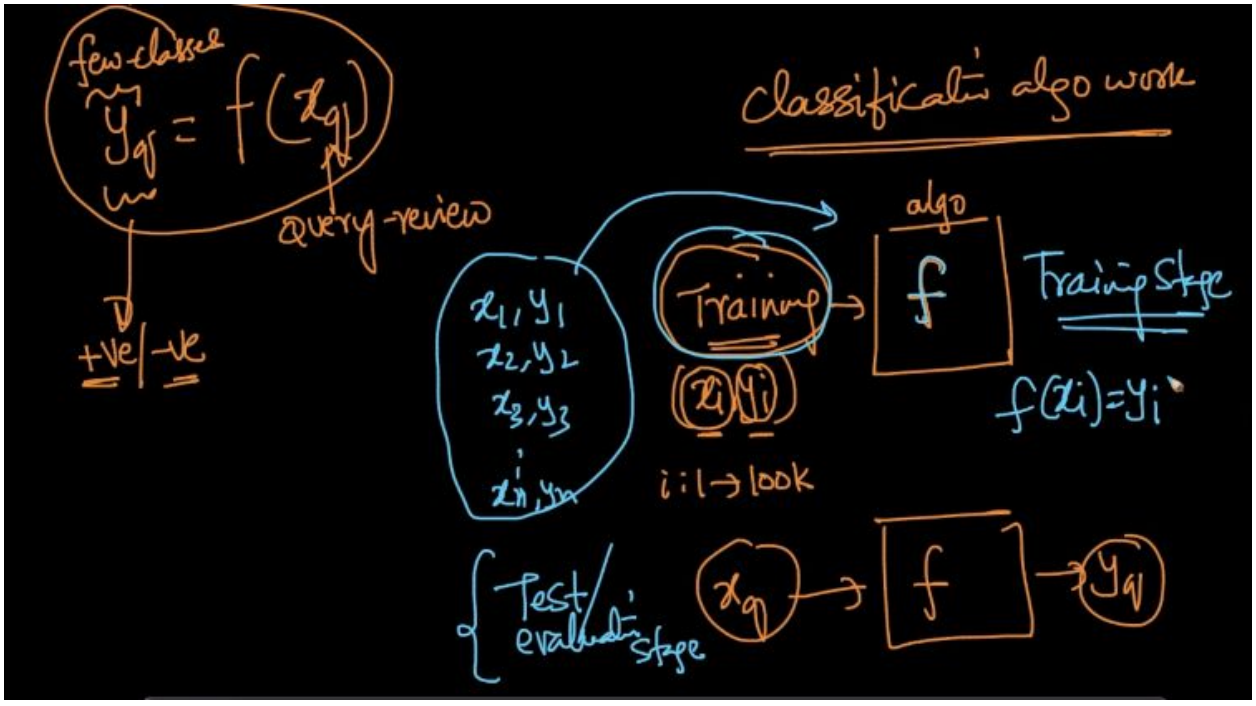
Module 3: Foundations of NLP and ML

Classification and Regression : KNN

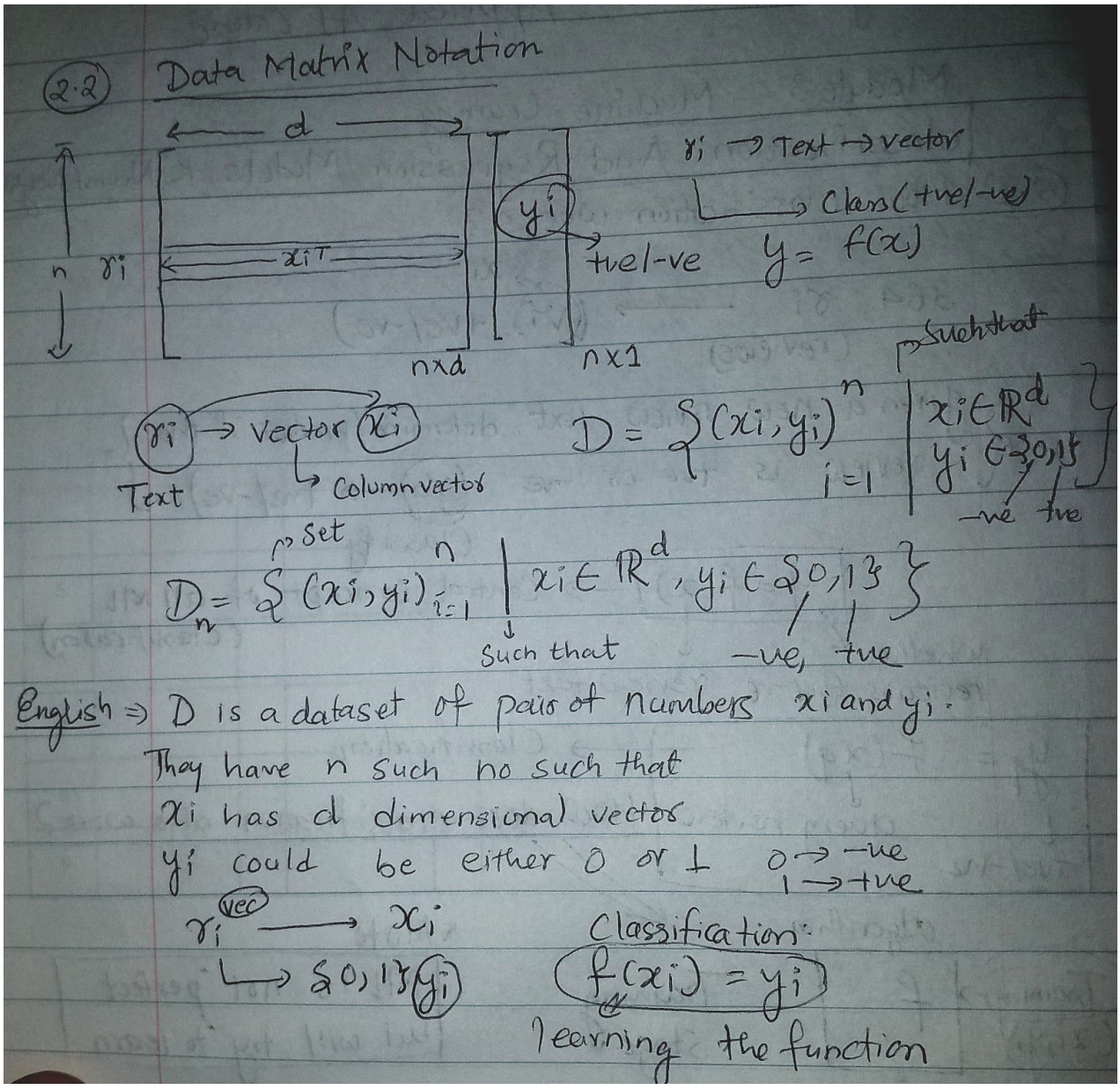
Objective of Classification :



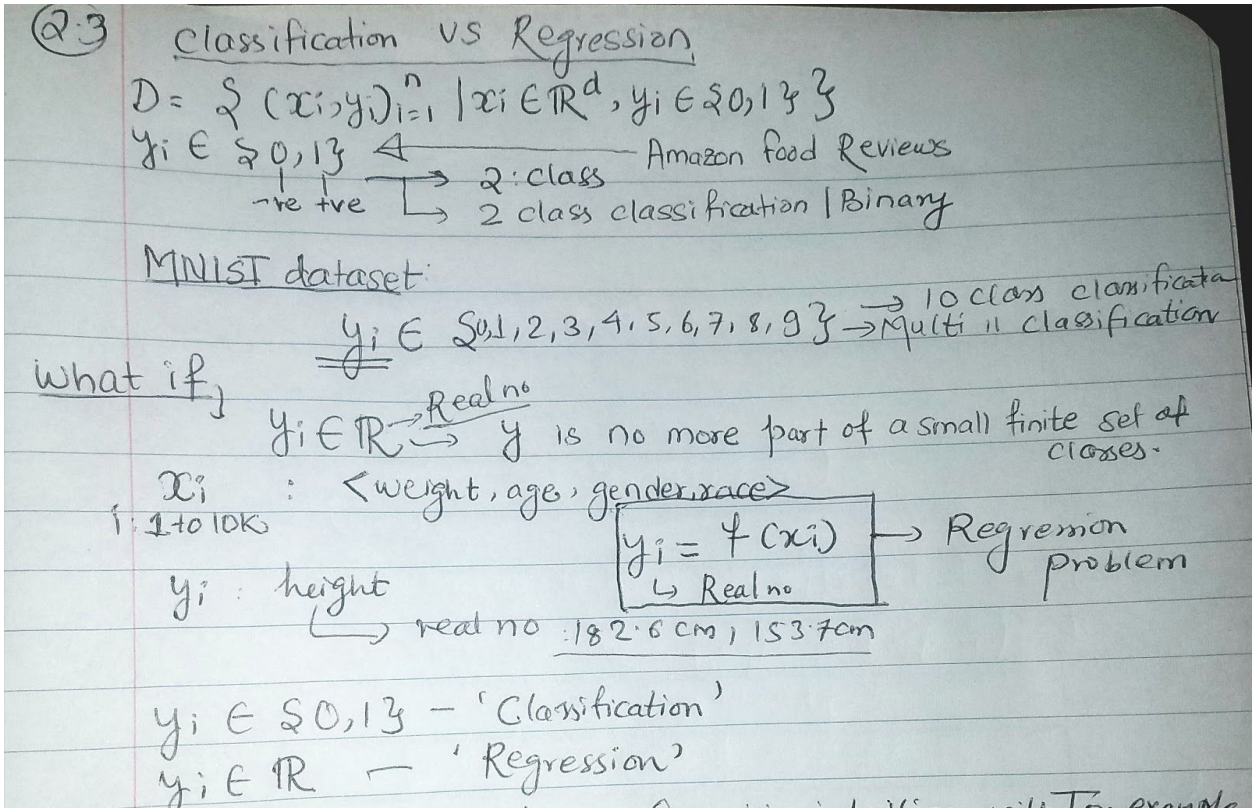


if we give new data xq, the algorithms should predict yq such that it has trained from various other datasets.

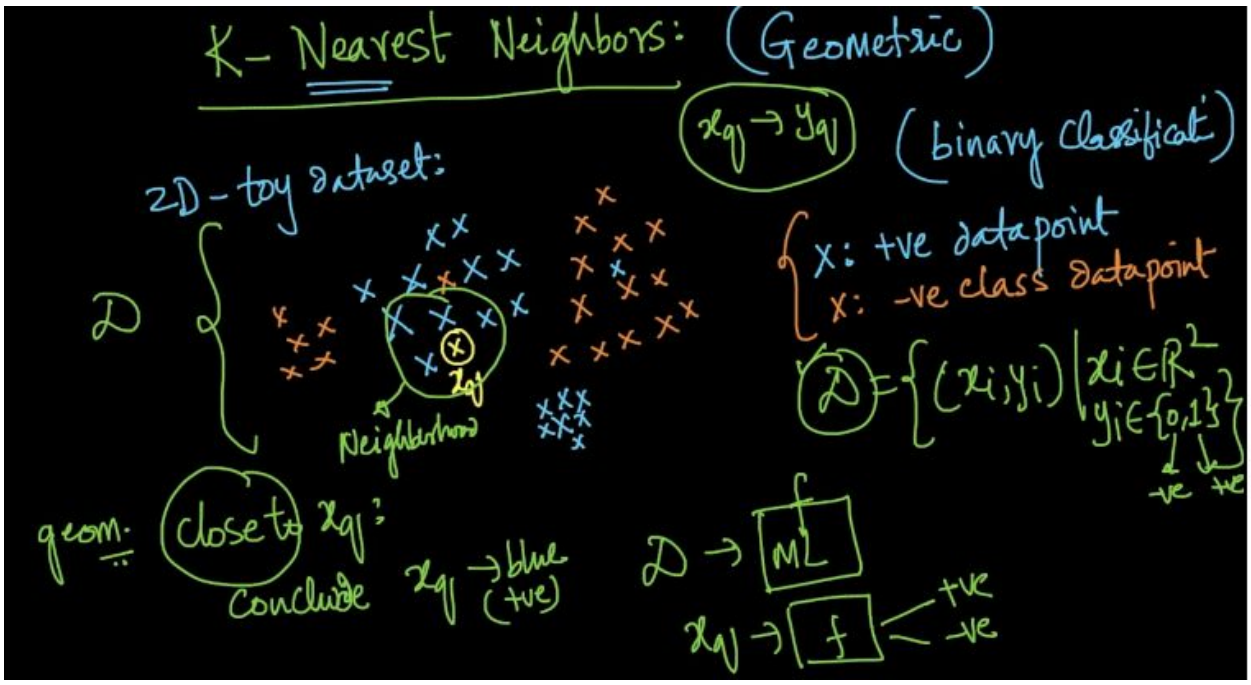
Data Matrix Notation



Classification VS Regression



K-Nearest Neighbours Geometric intuition with a toy example



$D = \{(x_i, y_i)\}$

given x_q → find $y_q \in \{0, 1\}$ (-ve, +ve)

(1) Find (K) "nearest" points to x_q in D

let $K=3$

$\{x_1, x_2, x_3\} \rightarrow$ 3 NN to x_q

$x_1 \downarrow y_1$, $x_2 \downarrow y_2$, $x_3 \downarrow y_3$

(2) $\{y_1, y_2, y_3\} \rightarrow$ Majority vote

K = odd ✓

$\{y_1, y_2, y_3\}$ → Majority vote

$+, +, +$ → (+) → y_q = +ve (or) blue

$+, +, -$ → (+) → y_q = +ve (or) blue

(X) if $K=4$

$+, +, -, -$ → (+) or (−)

---

x_q → k-NN of x_q

$x_1, x_2, \ldots x_k$

↓

$y_1, y_2, \ldots y_k$

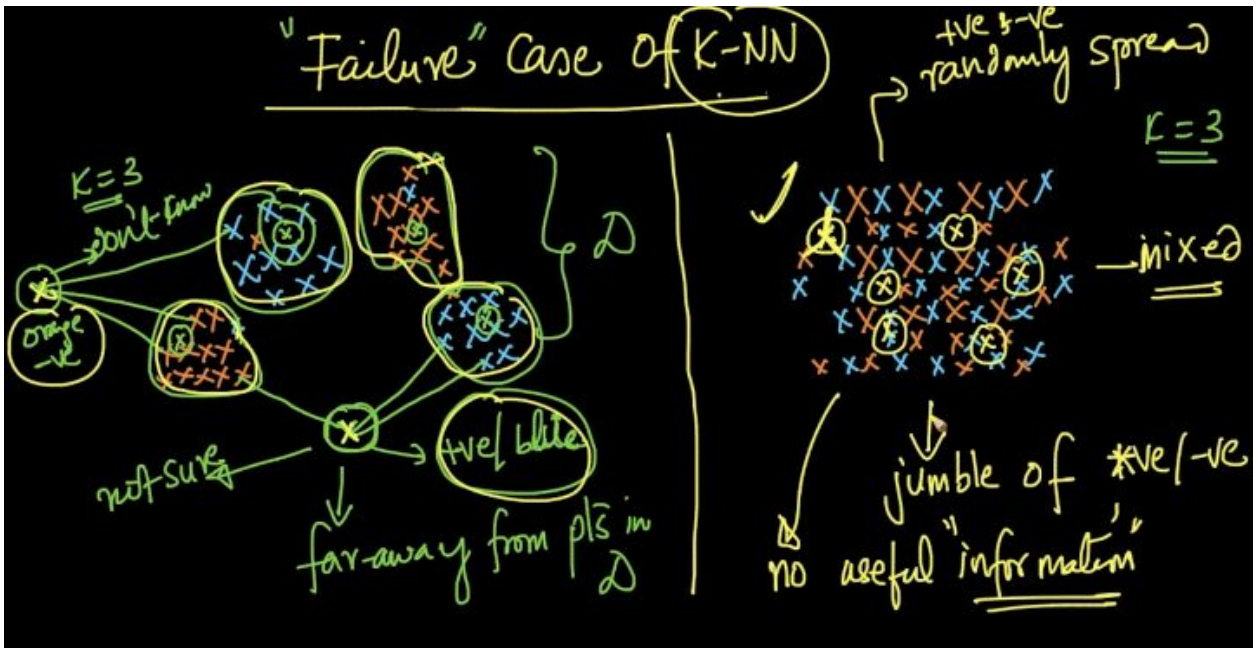
↓

Majority vote

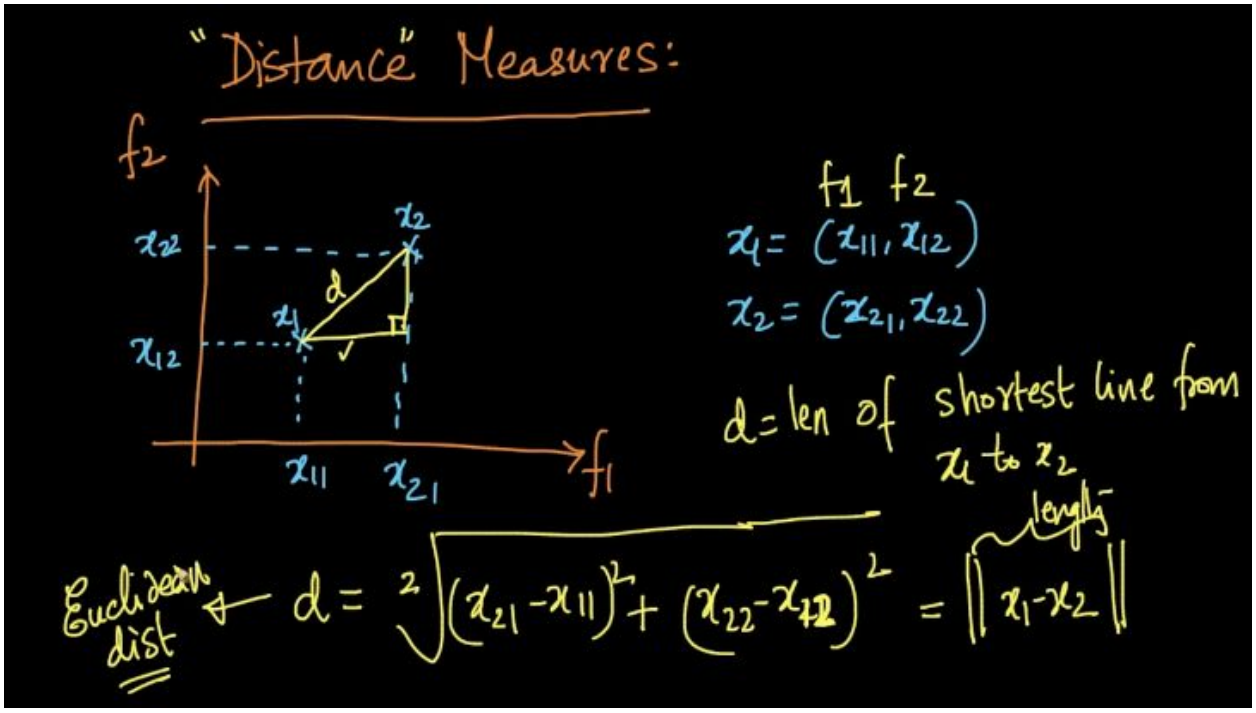
↓

y_q

Failure Cases of KNN



Distance measures: Euclidean(L2) , Manhattan(L1), Minkowski, Hammin



$x_1 \in \mathbb{R}^d, \; x_2 \in \mathbb{R}^d$

Eucl_dist: $\|x_1 - x_2\|_2 = \left(\sum_{i=1}^{d} (x_{1i} - x_{2i})^2 \right)^{1/2}$

$\|(x_1 - x_2)\|_2 \rightarrow$ L2 norm

$\|x_1\|_2 =$ euc. dist of x_1 from origin $= \left(\sum_{i=1}^{d} x_{1i}^2 \right)^{1/2}$

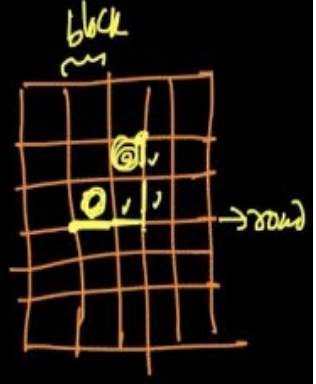
"Manhattan" dist:

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

L1-norm of vector $(x_1 - x_2)$

$\|(x_1 - x_2)\|_1$

$\|x_1\|_1 = \sum_{i=1}^{d} |x_{1i}|$ (absolute)



Lp-norms → Minkowski dist

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

$\{(-3)^2 = 9$
$\{(1-3)|^2 = 9$

= Lp-norm of $(x_1 - x_2)$

p=2 → Minkowski dist → Eucl. dist
p=1 → " → Manhattan dist

Hamming dist (boolean vectr)

x_1, x_2 → boolean vectr → Binary BoW

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

Hamming-dist (x_1, x_2) = # locations/dimensions where binary vectrs differ

↳ 3

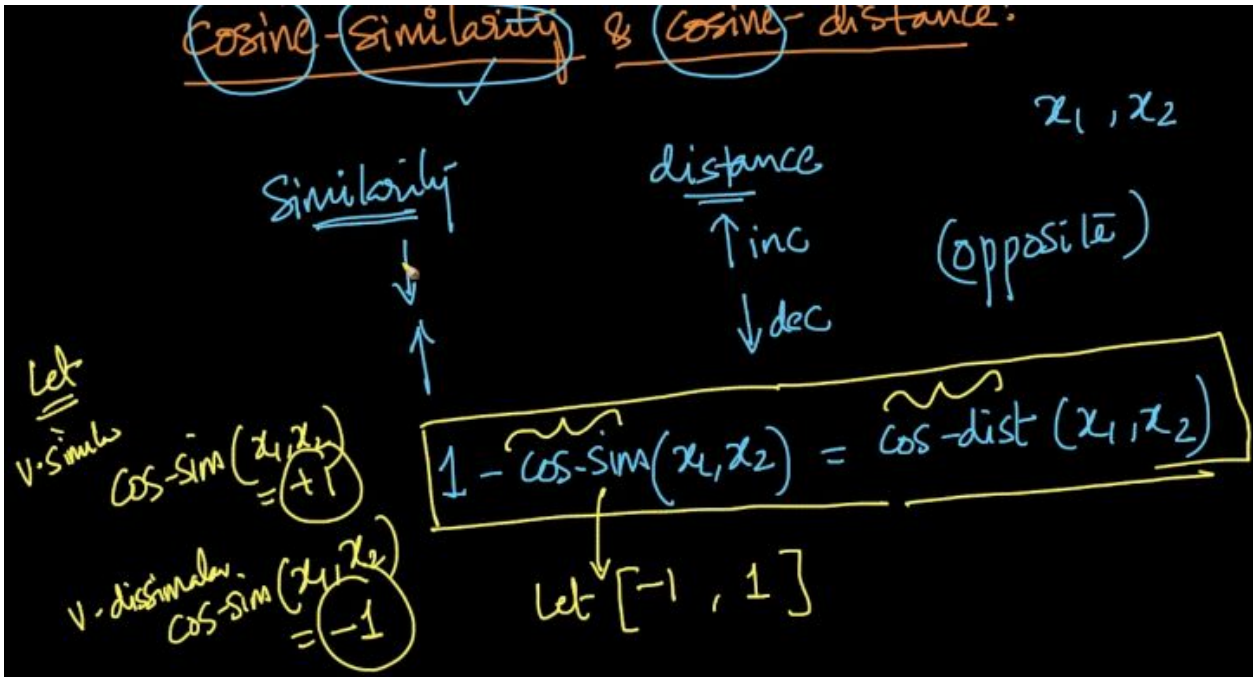
strings:

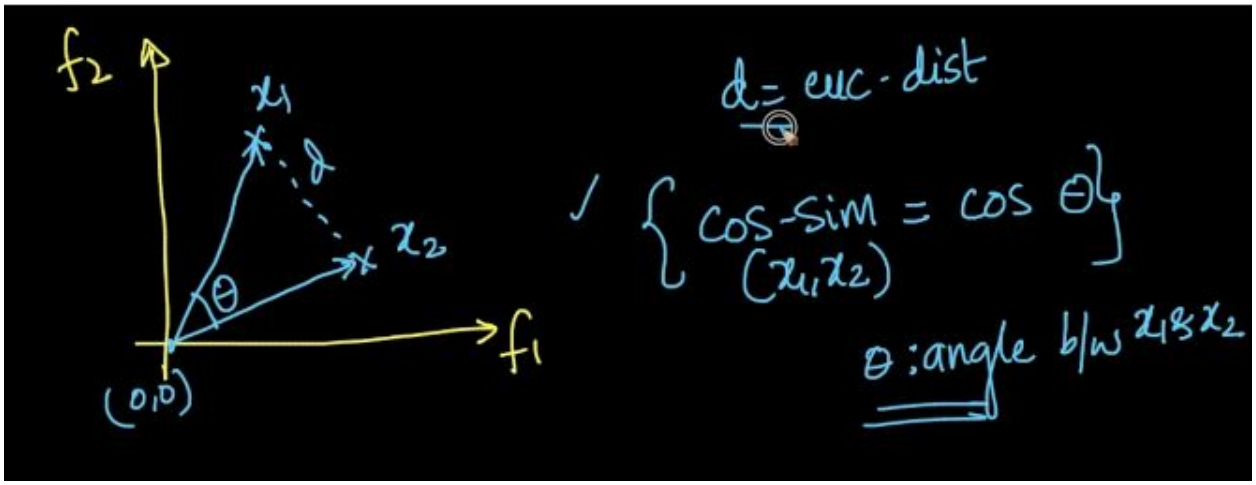
$x_1 = a\,b\,c\,a\,d\,e\,f\,g\,h\,i\,k$ ← Gene-code/seq
$x_2 = a\,c\,b\,a\,d\,e\,g\,f\,h\,i\,k$ AGTC

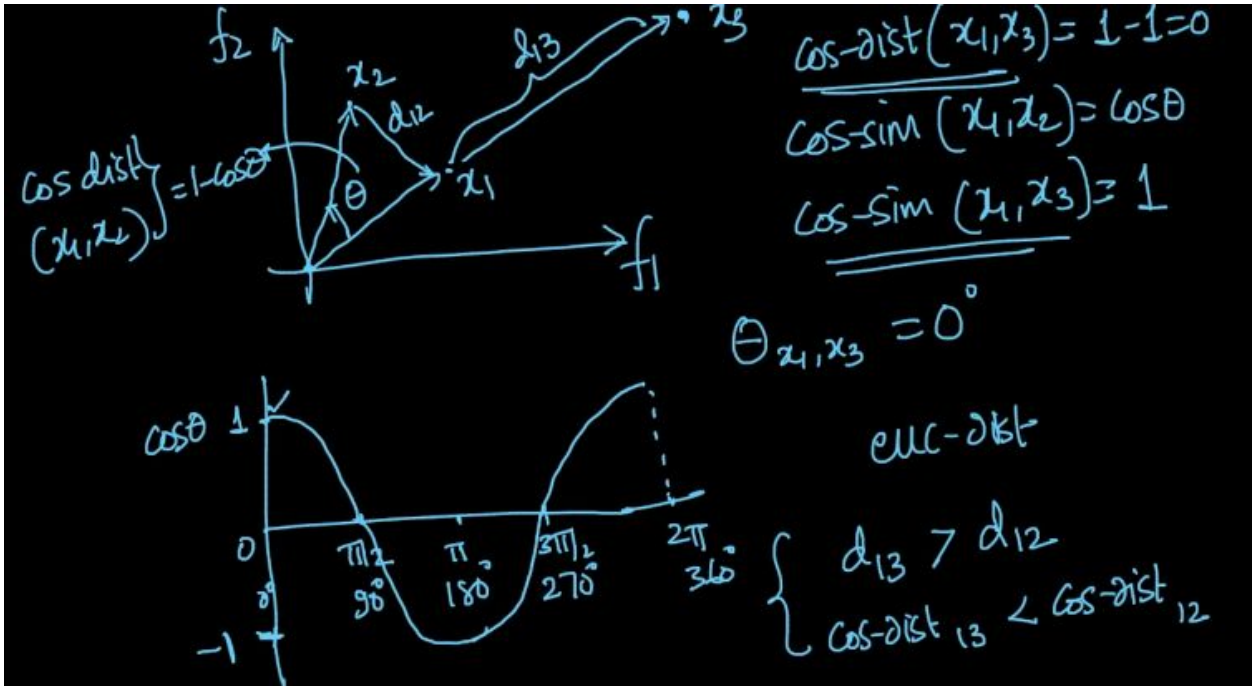
hamming dist (x_1, x_2) = 4

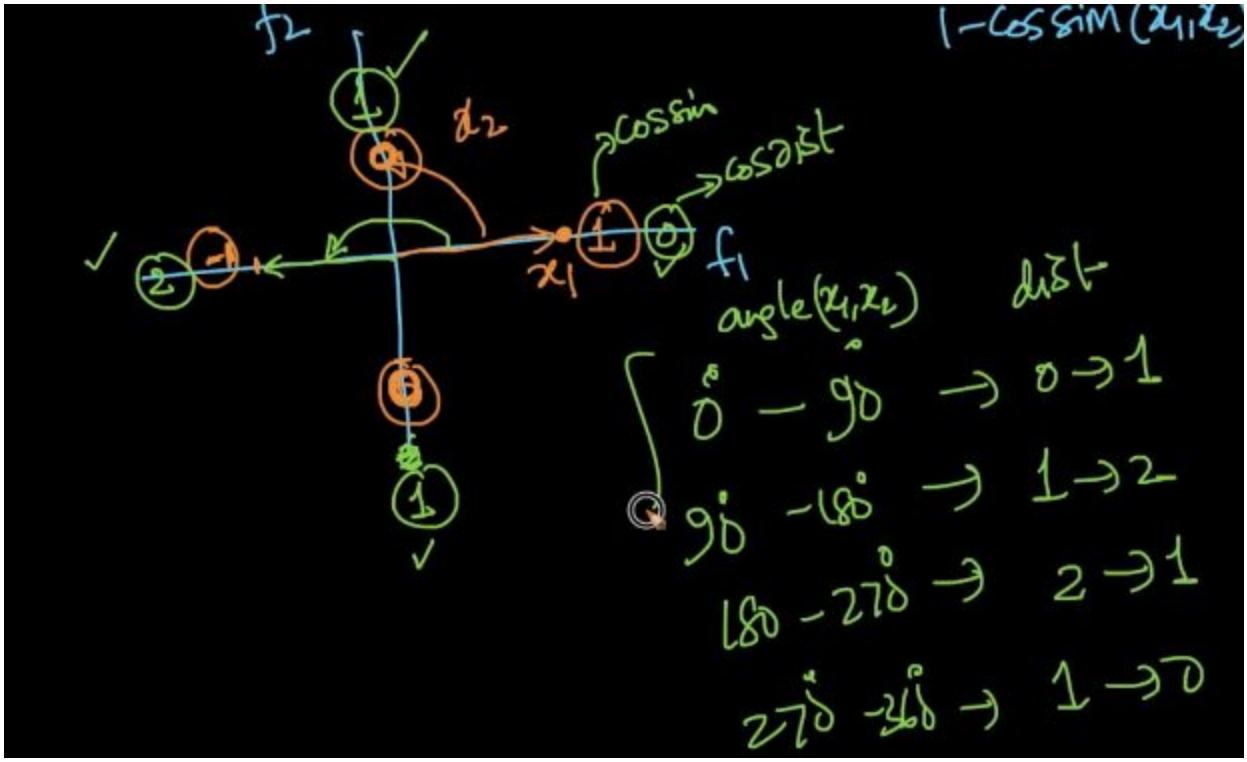
$x_1 =$ AAGTCTCAG
$x_2 =$ AGATCTGA

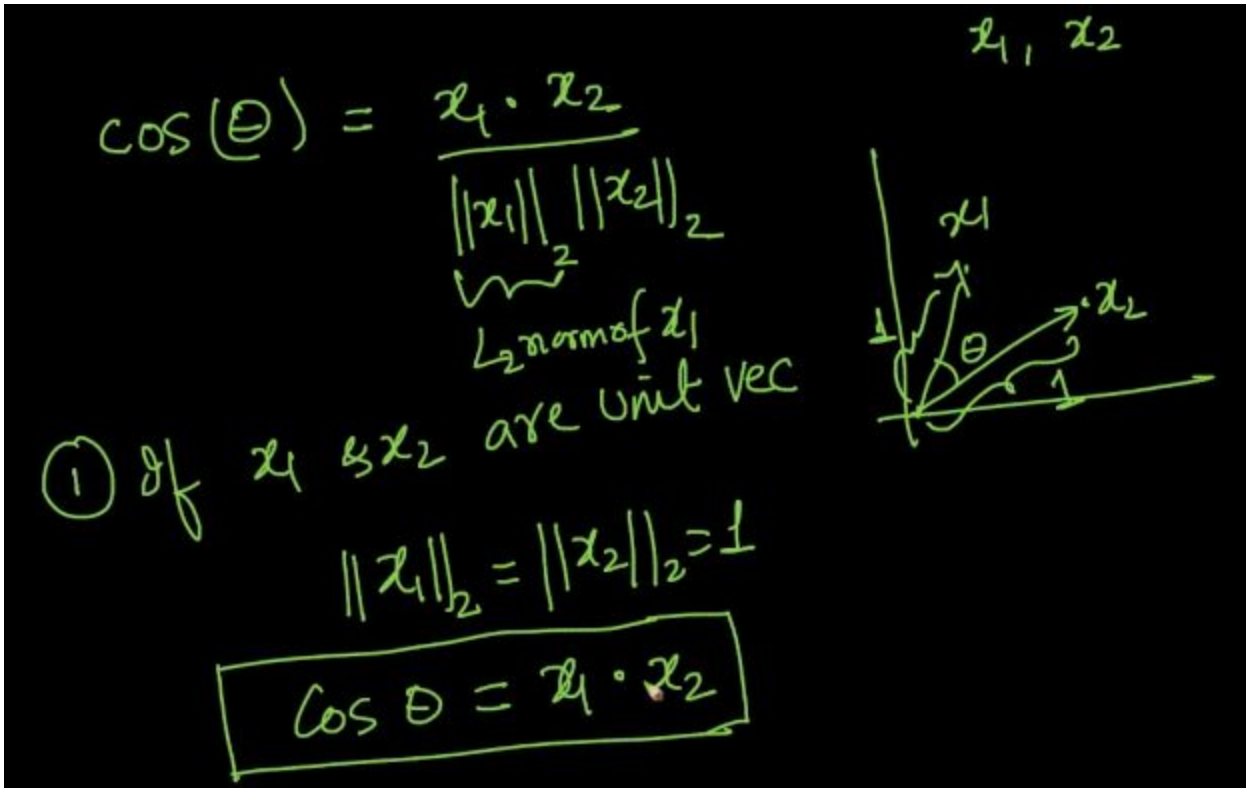
Cosine Distance & Cosine Similarity

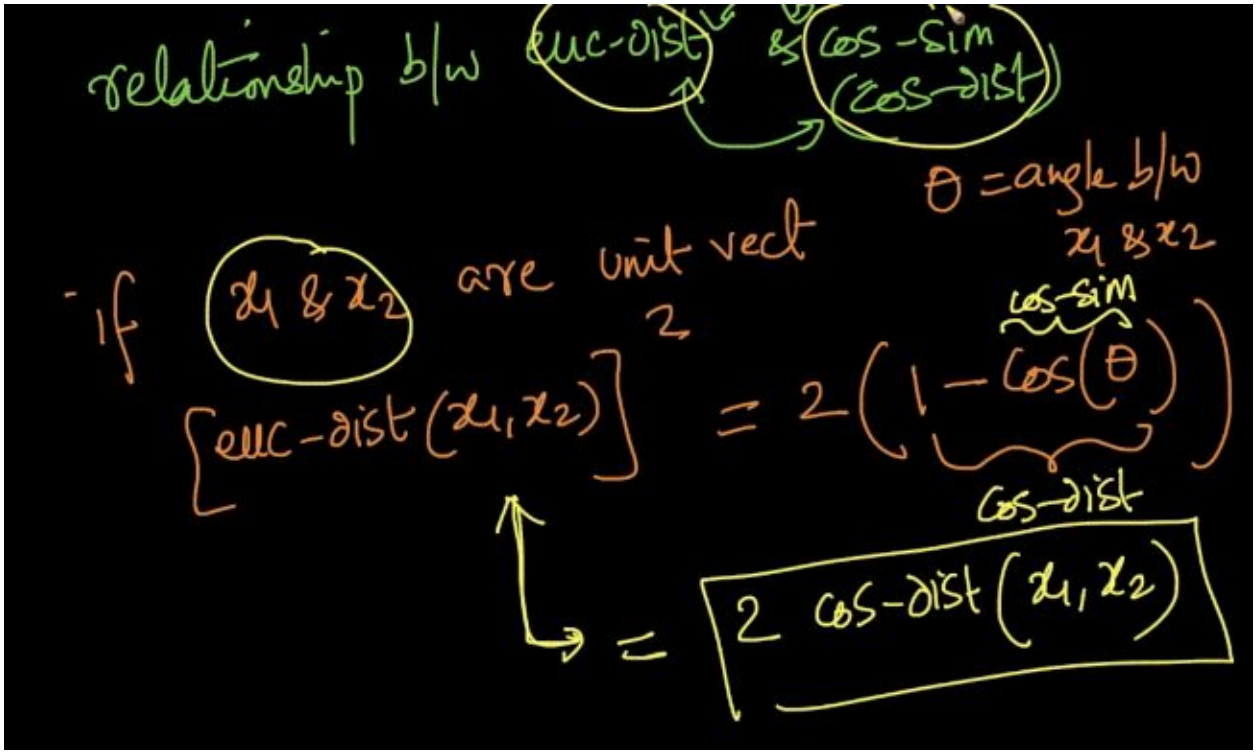





f2
f1
x2
x1
x3
d12
d13
θ
cos dist (x1,x2) = 1-cosθ
cos-dist (x1,x3) = 1-1=0
cos-sim (x1,x2) = cosθ
cos-sim (x1,x3) = 1
θx1,x3 = 0°
cosθ
1
0
-1
π/2
90°
π
180°
3π/2
270°
2π
360°
euc-dist
d13 > d12
cos-dist 13 < cos-dist 12

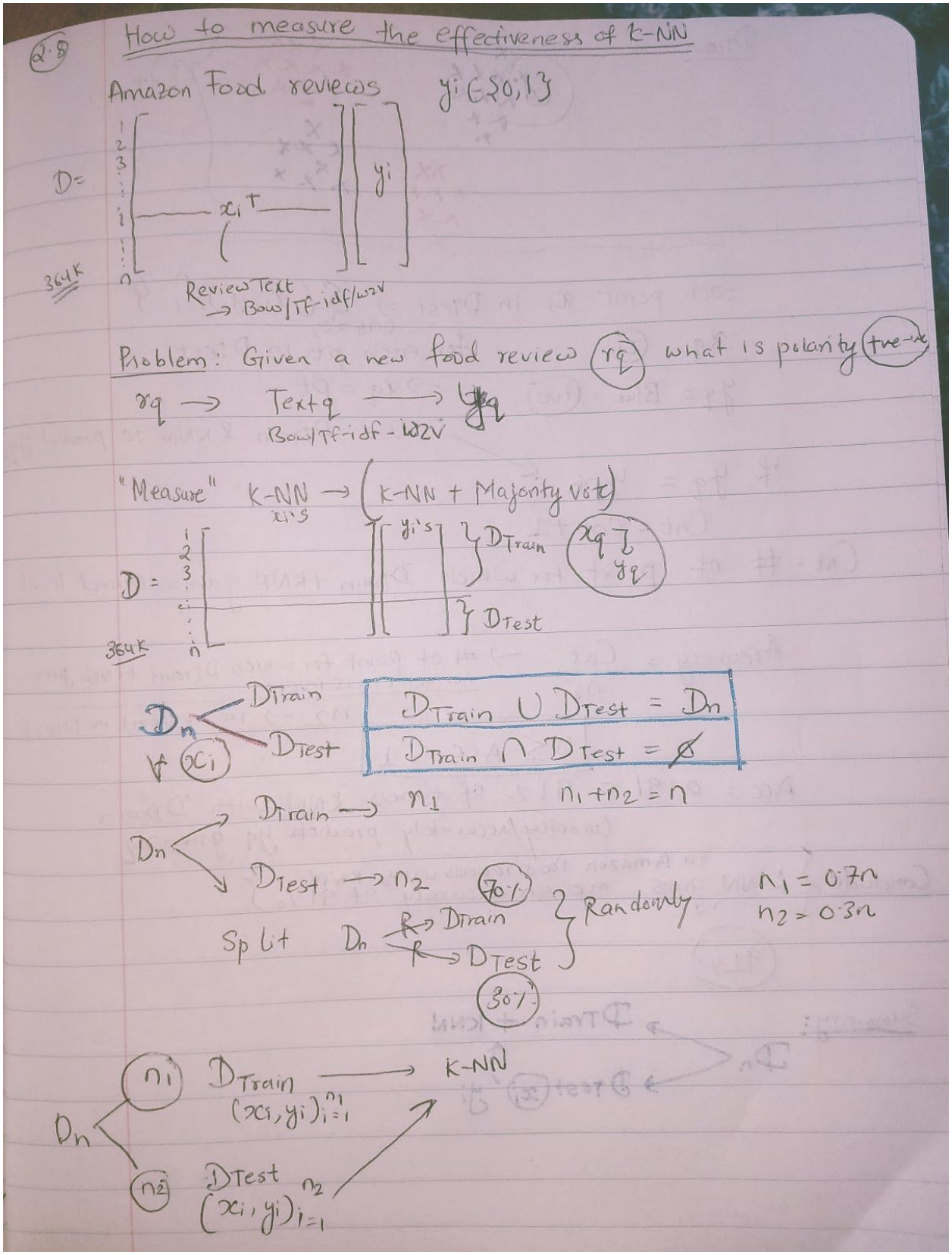

f2
f1
d2
xcossim
xcosdist
x1
1-cossim (x1,x2)
angle (x1,x2)
dist
0° - 90° → 0 → 1
90° - 180° → 1 → 2
180 - 270° → 2 → 1
270° - 360° → 1 → 0



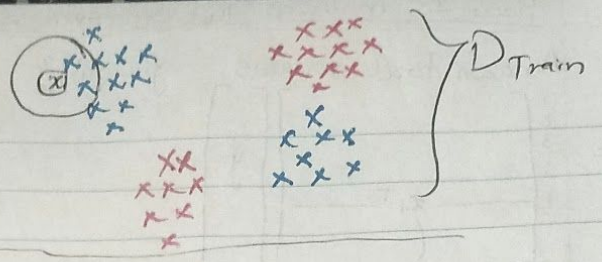


Link:
https://cmry.github.io/notes/euclidean-v-cosine

How to measure the effectiveness of k-NN?



D_{Train}



each point x_i in $D_{Test} \Rightarrow \{(x_i, y_i)_{i=1}^{n}\}$ (Cnt = 0)

$x_q = (x_i)$ for each pt in D_{Test}

y_q = Blue = (+ve) → x_q = pt

→ use D_{Train} & KNN to predict y_q

if $y_q = y_{pt}$

Cnt = Cnt + 1

Cnt = # of point for which D_{Train} + KNN gave a correct level

$$Accuracy = \frac{Cnt}{n_2}$$ → # of point for which D_{Train} + KNN gave correct class level

n_2 → no of point in D_{Test}

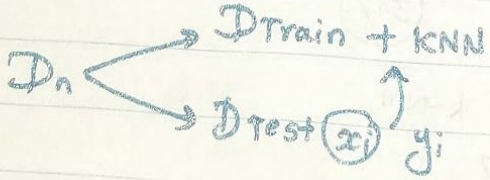
$$\boxed{0 \leq Acc \leq 1}$$

Acc = 0.91 = 91% of times KNN with D_{Train} Correctly/accurately predicts y_q given x_q

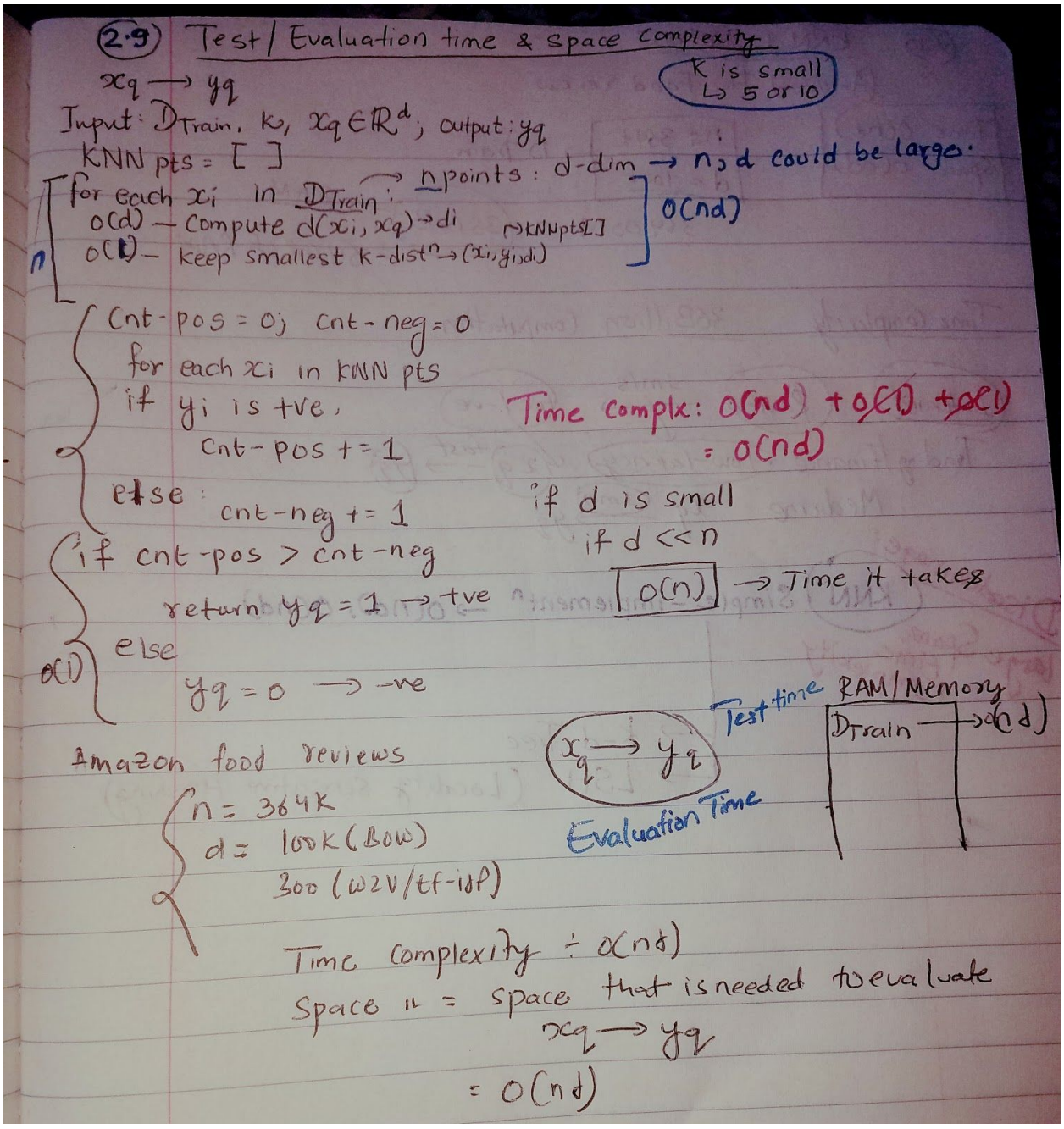
Conclude { K-NN on Amazon Food reviews using KNN gives me an accuracy of 91% }

(91%)

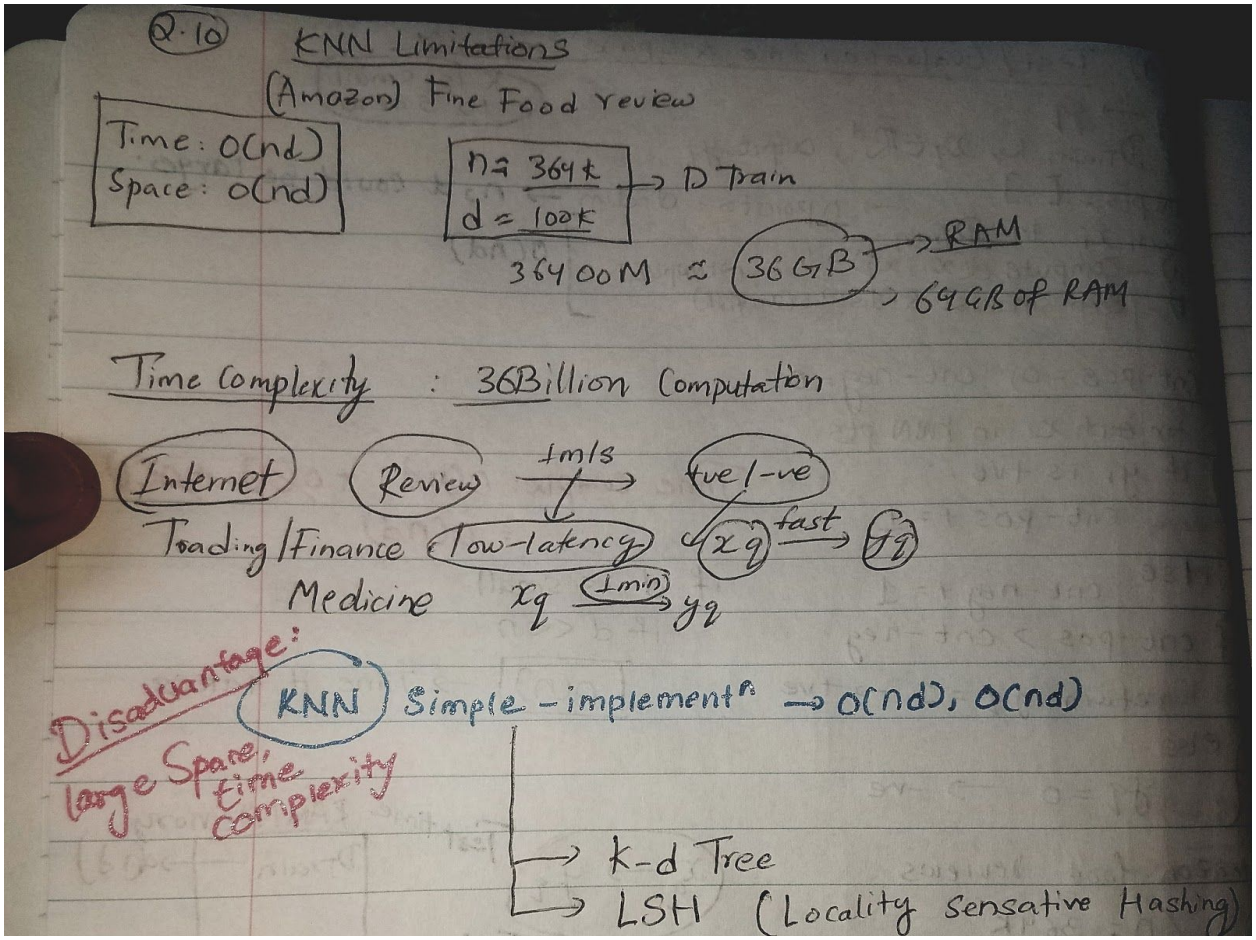
Summary:



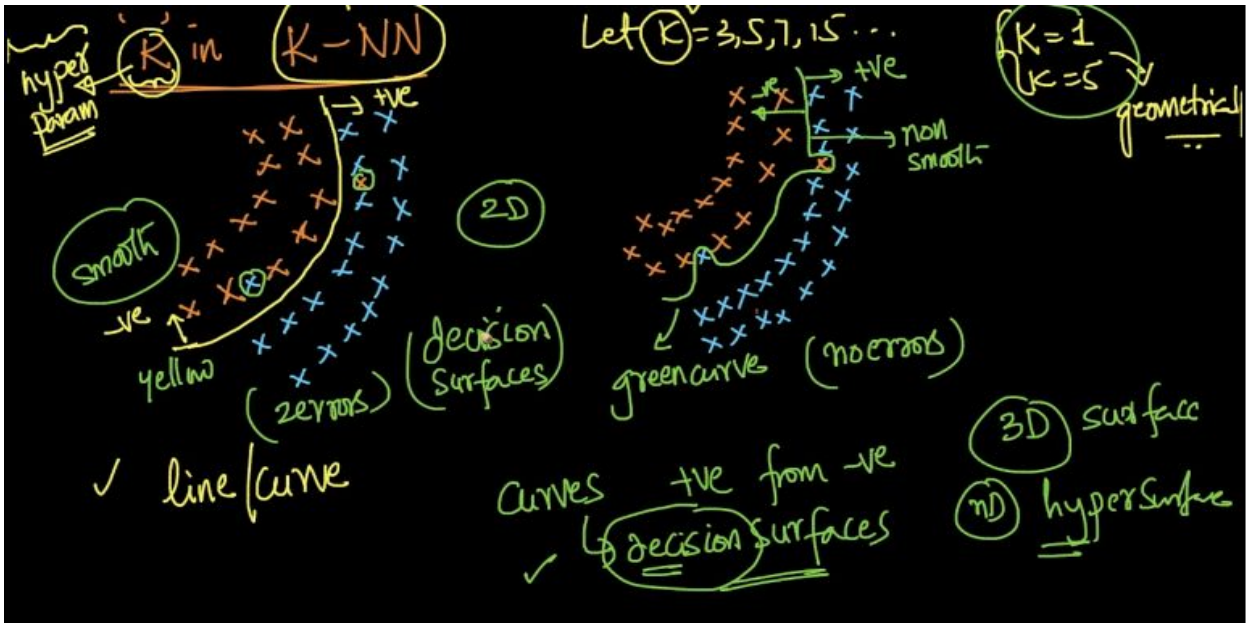
Evaluation Time and Space Complexity

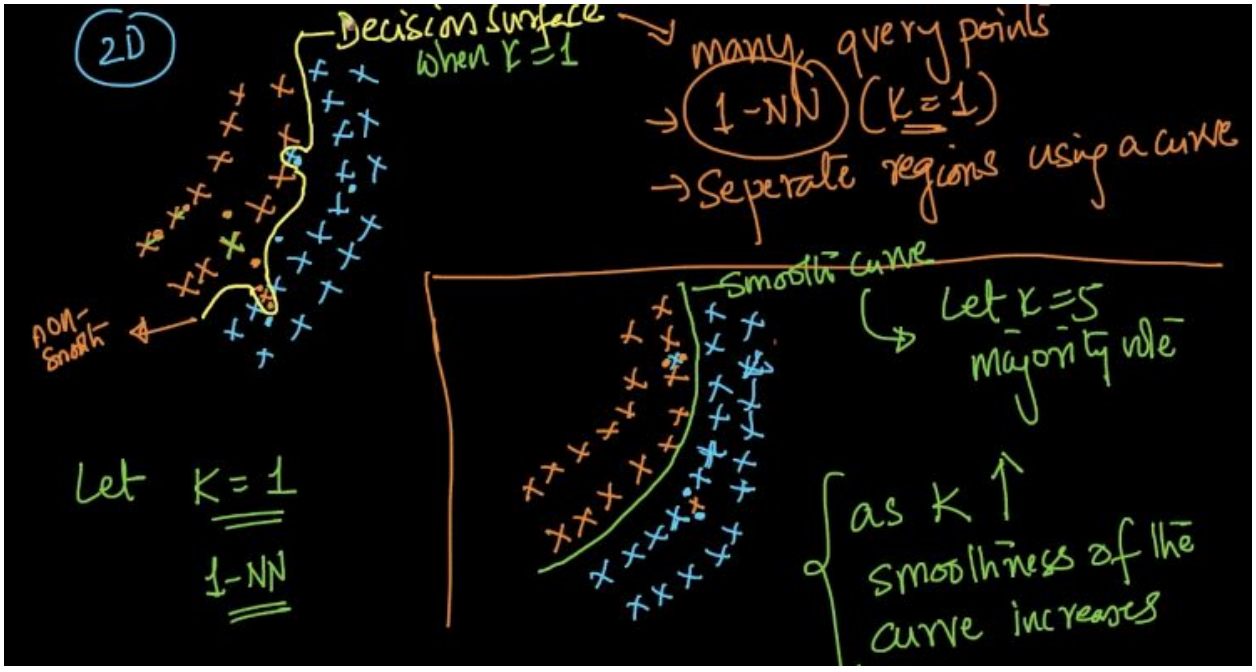


KNN Limitations



Decision surface for K-NN as K changes





In K-NN
the smoothness of the decision surfaces increases as K increases

Lets say $K = n$

$K = 1, 2, \ldots n$
max-value of $K = n$

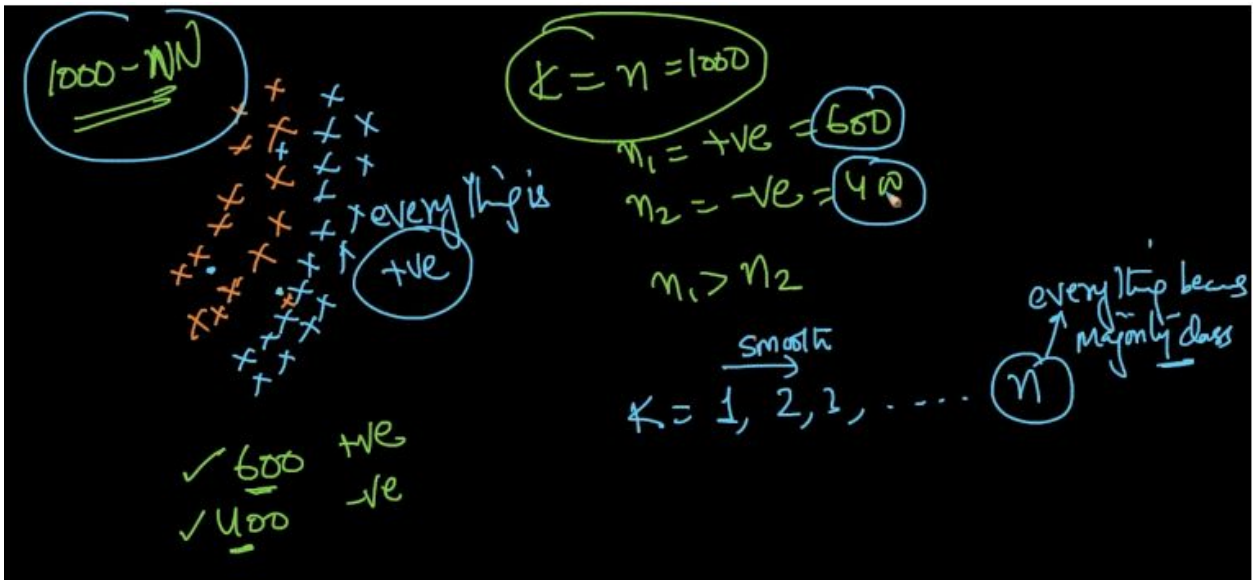
n = Total number of points (1000)

n_1 = +ve (600)

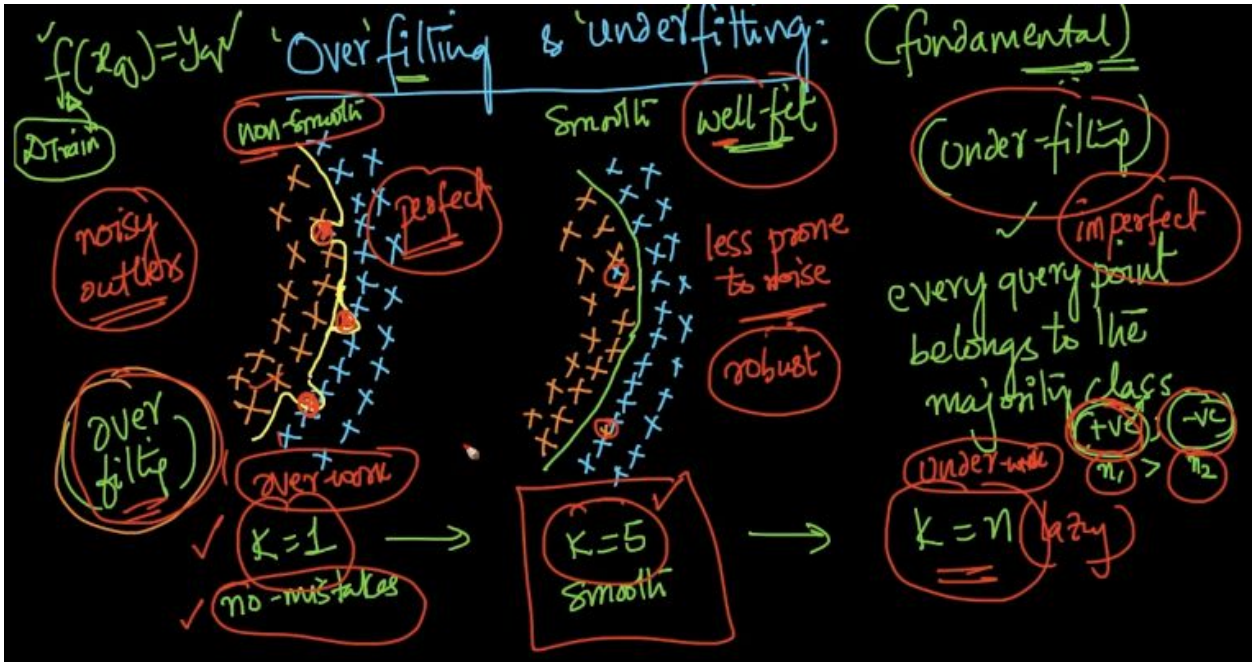
n_2 = -ve (400)

$n_1 + n_2 = n$

Lets say $n_1 > n_2$



Overfitting and Underfitting



Need for Cross validation

$K = 1, 2, 3, \ldots, n$ ← overfitting, underfitting

How to determine K?

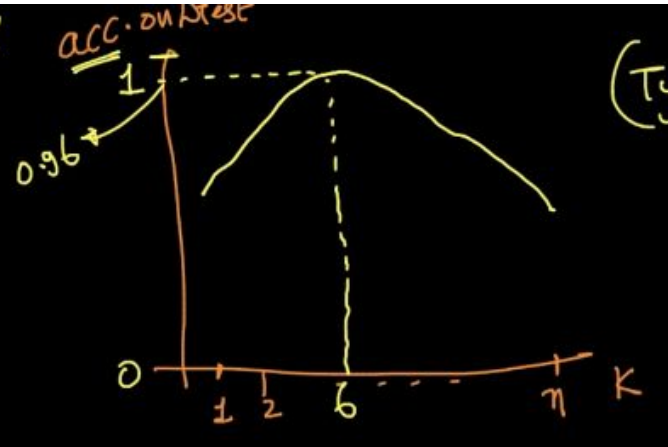
D_n —(R)— D_{train} (70%), D_{test} (30%)

acc. on D_{test} = num. of correctly classified pts / Total # pts

One idea:

	Train	acc. on D_{test}
K=1	D_{train}	0.78
K=2	"	0.82
K=3	"	0.85
⋮		

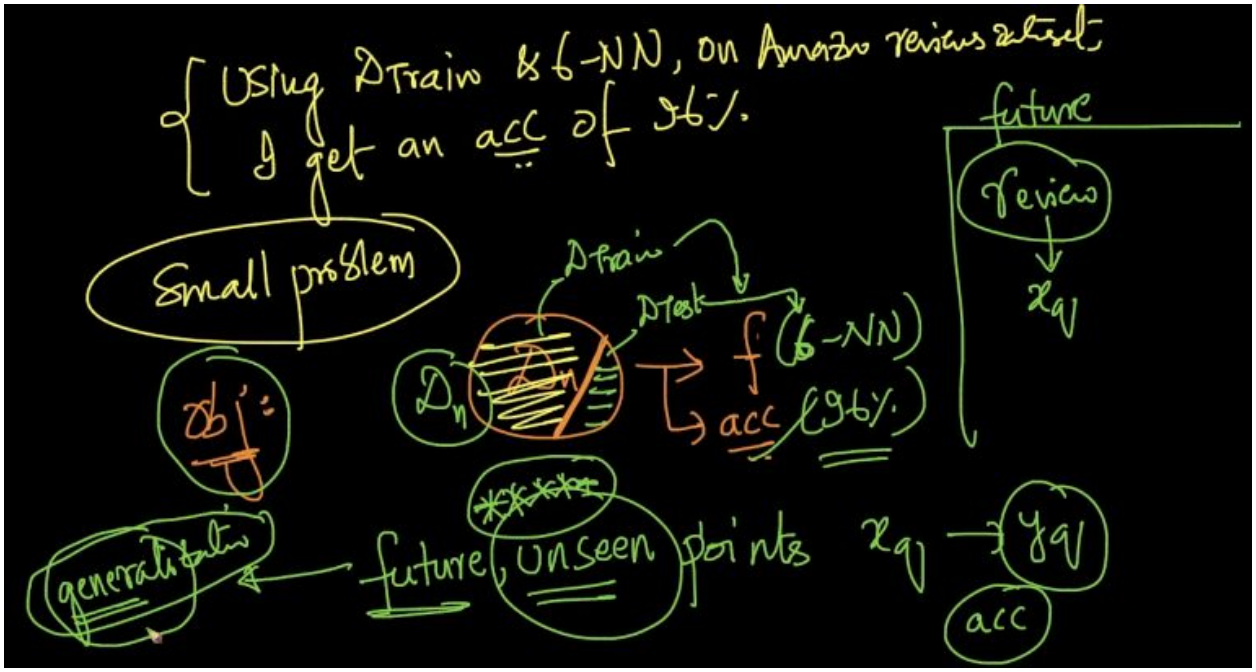
Another

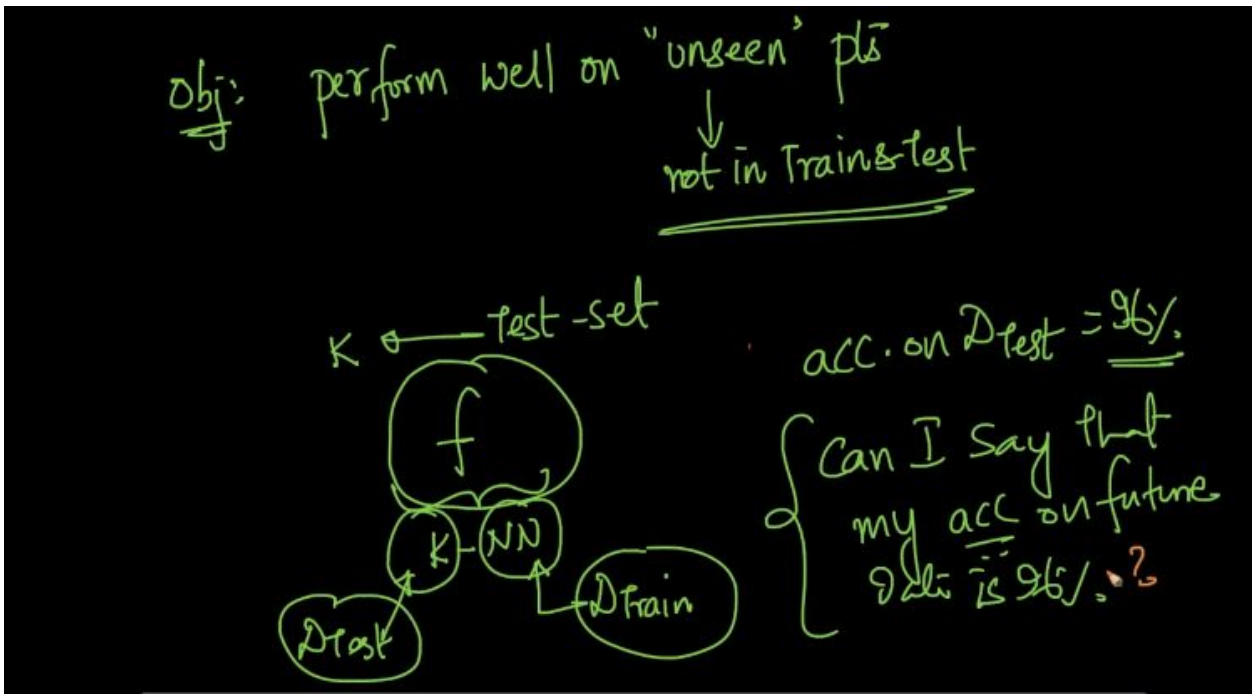


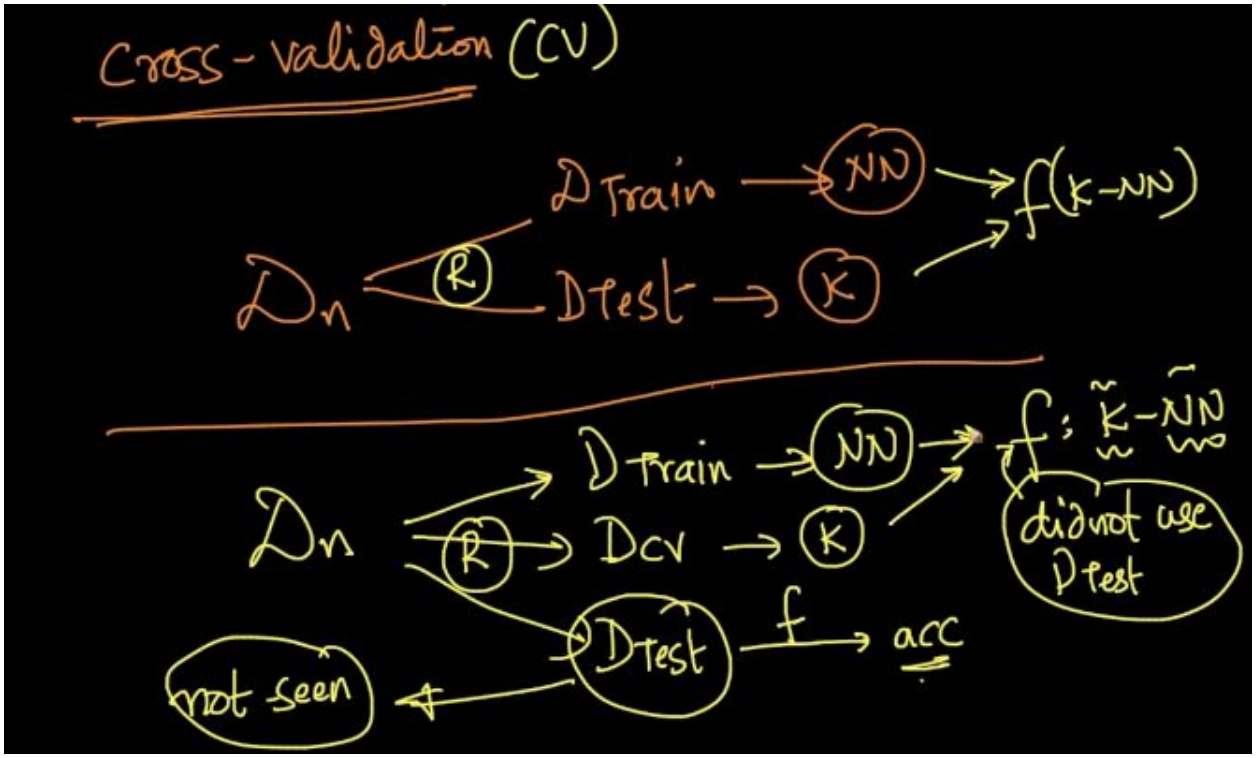
(Typically)

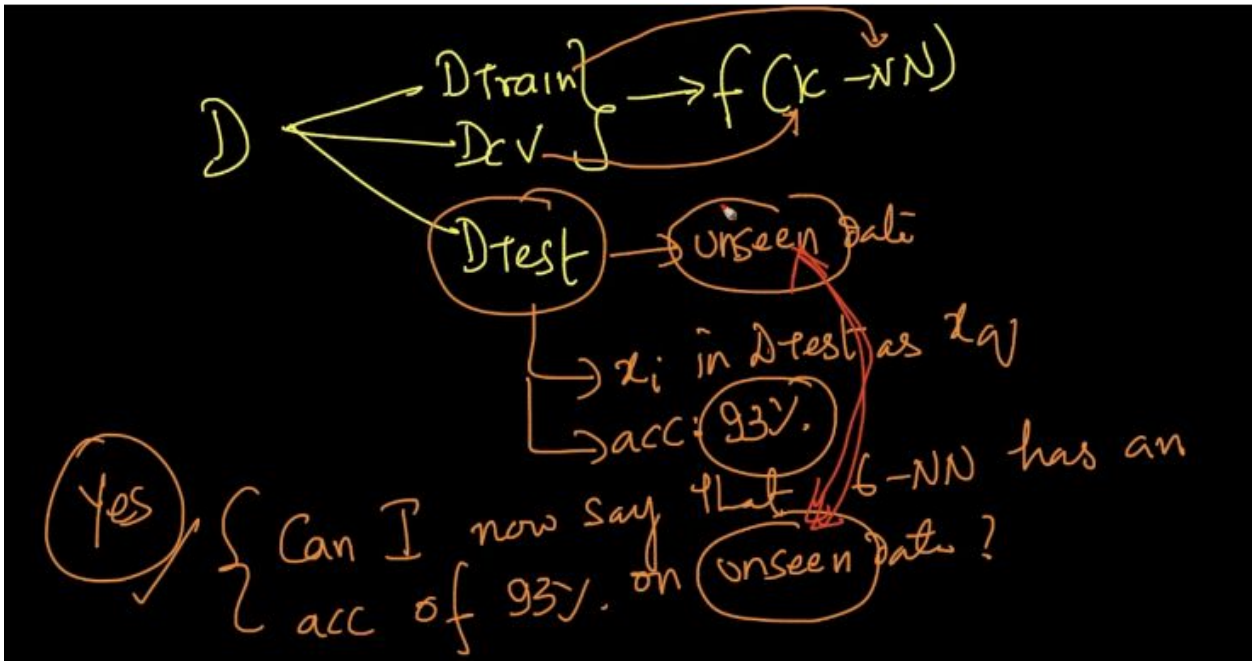
K=6 gives me the best acc on D_{test} when using D_{train} as training data

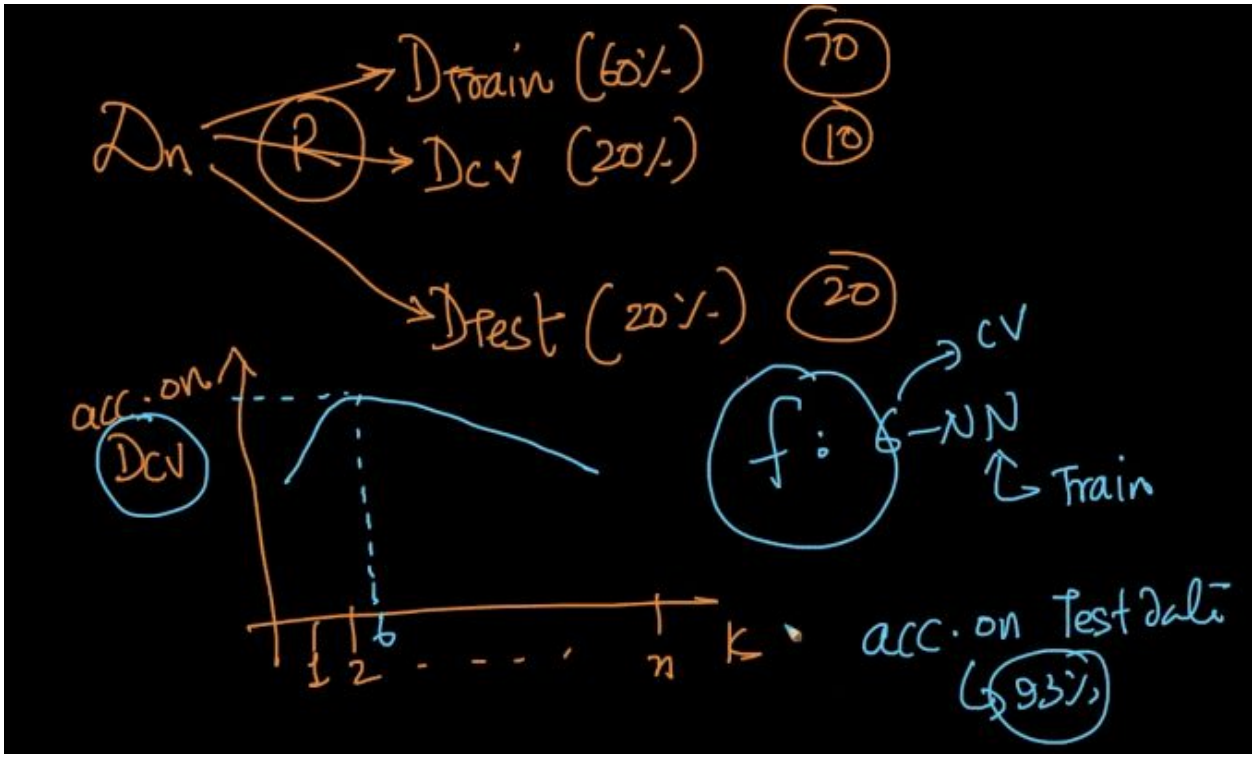
K=6 (6-NN) 96%

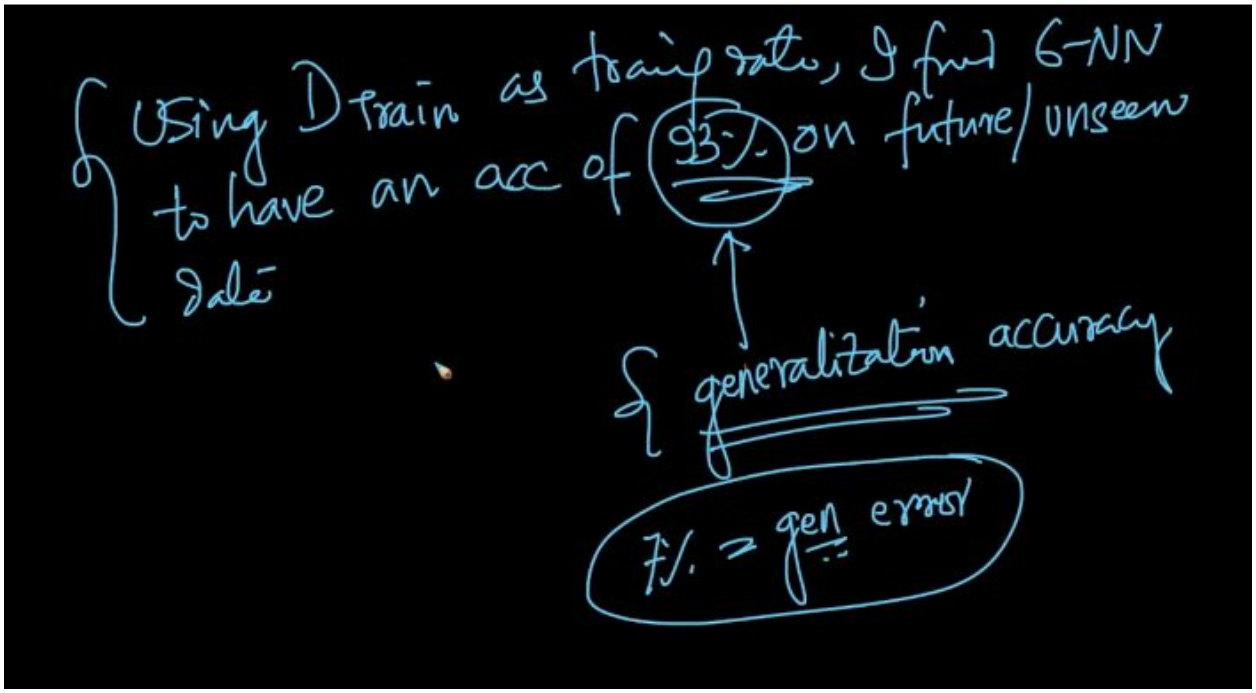

Using D_{Train} & 6-NN, on Amazon reviews dataset
I get an acc of 96%.
Small problem
obj:
generalization
D_{Train}
D_{Test}
D_n
f (6-NN)
acc (96%)
future
reviews
x_q
future, unseen points x_q
y_q
acc


obj: perform well on "unseen" pts
not in Train & test
K
test-set
f
K-NN
D_{Train}
D_{Test}
acc. on D_{test} = 96%.
Can I say that my acc on future data is 96%.?

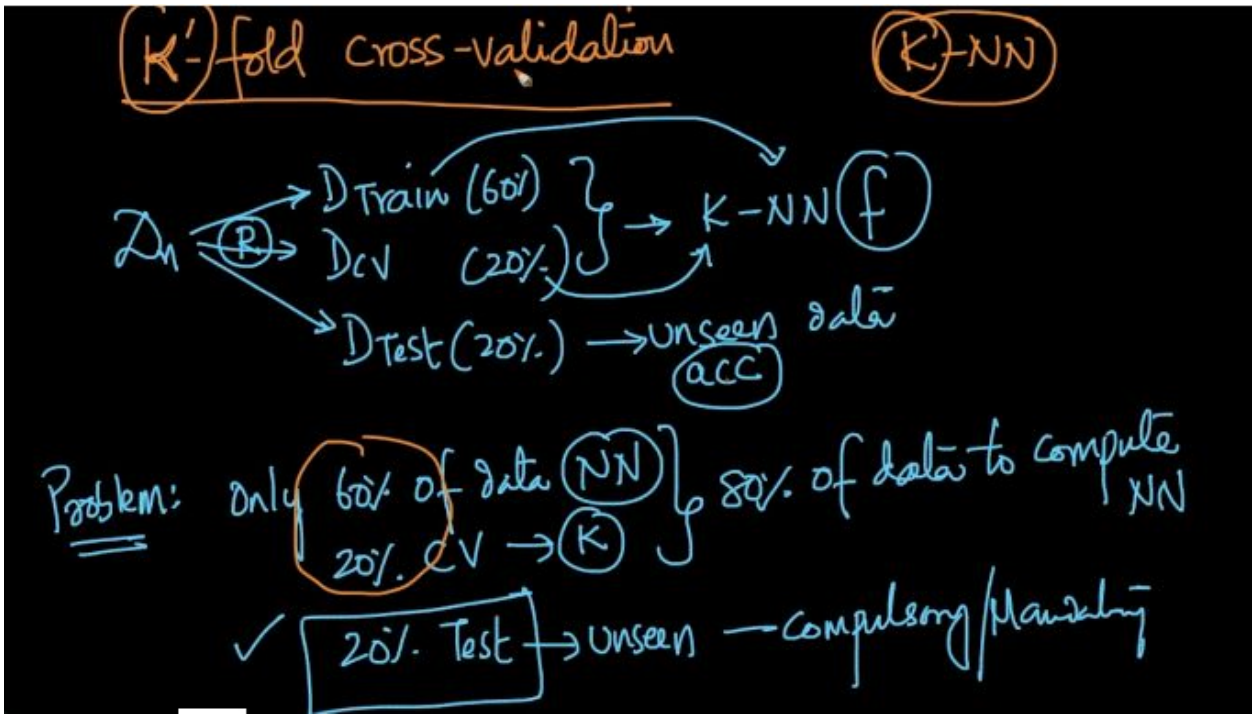

Cross-validation (CV)
D Train
NN
f(K-NN)
Dn
R
Dtest
K
D Train
NN
f; K̃-ÑN
Dn
R
Dcv
K
did not use Dtest
Dtest
f
acc
not seen


D
Dtrain
Dcv
f (K-NN)
Dtest
unseen data
xi in Dtest as xq
acc: 93%.
Yes
Can I now say that 6-NN has an acc of 93%. on unseen data ?

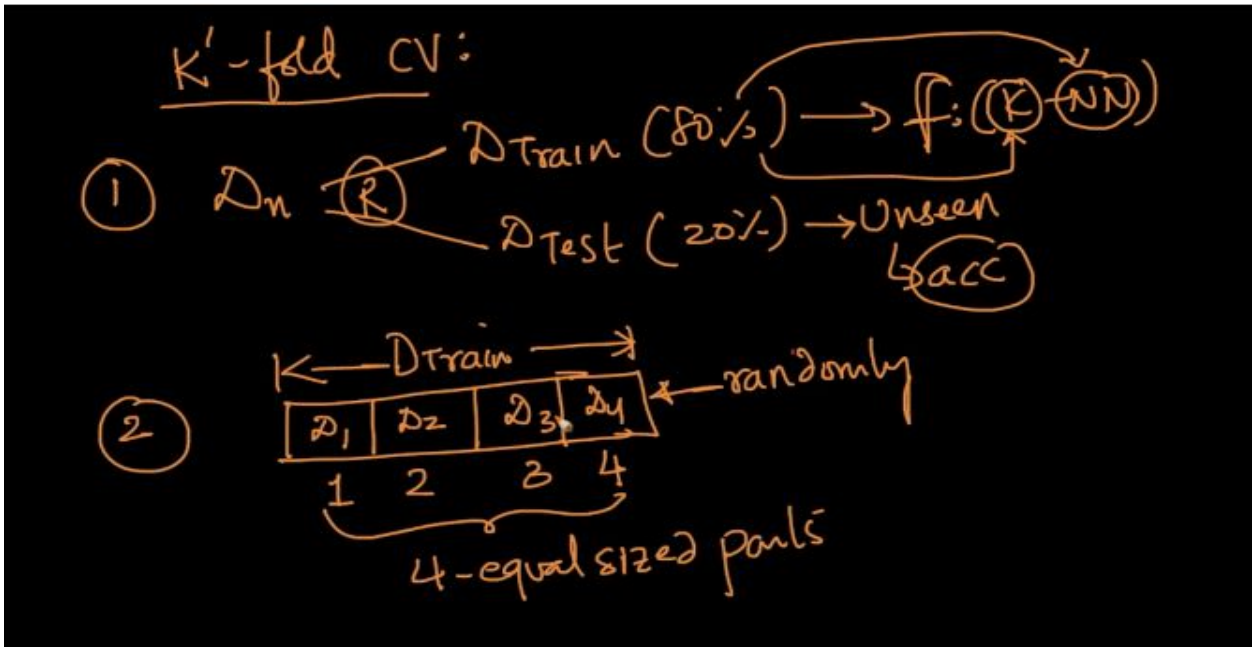

Dtrain (60%)
70
Dn
R
Dcv (20%)
10
Dtest (20%)
20
acc. on
Dcv
f:
6-NN
cv
Train
1
2
6
n
K
acc. on Test data
93%

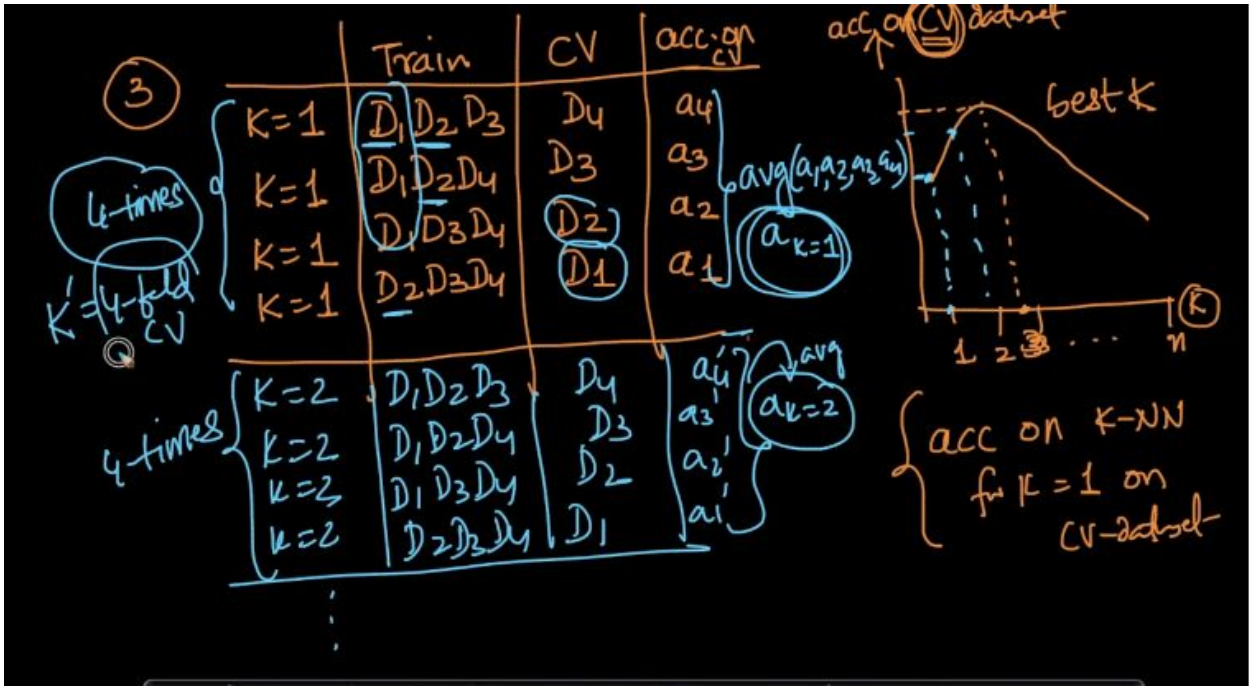

Using Dtrain as train data, I find 6-NN to have an acc of 93% on future/unseen data
generalization accuracy
7% = gen error

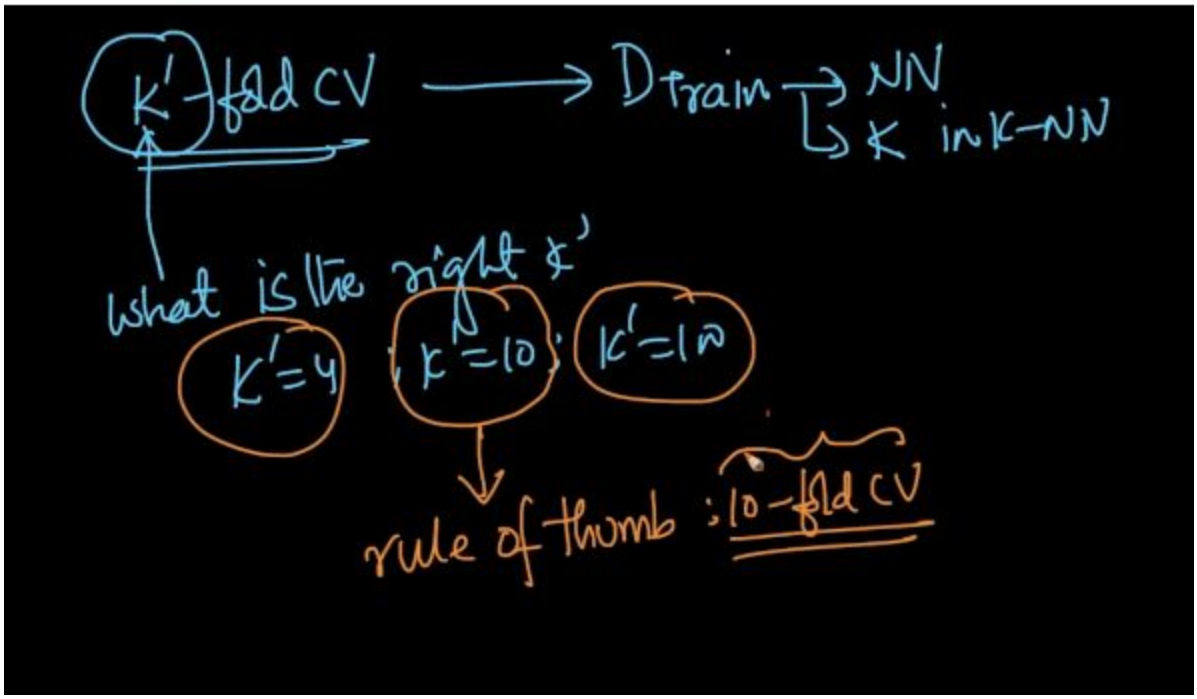
K-fold cross validation



More the training data, better is your algorithms.







Time it takes to compute the optimal /best k in K-NN increases by k' times.if we use k'-fold cross validation.

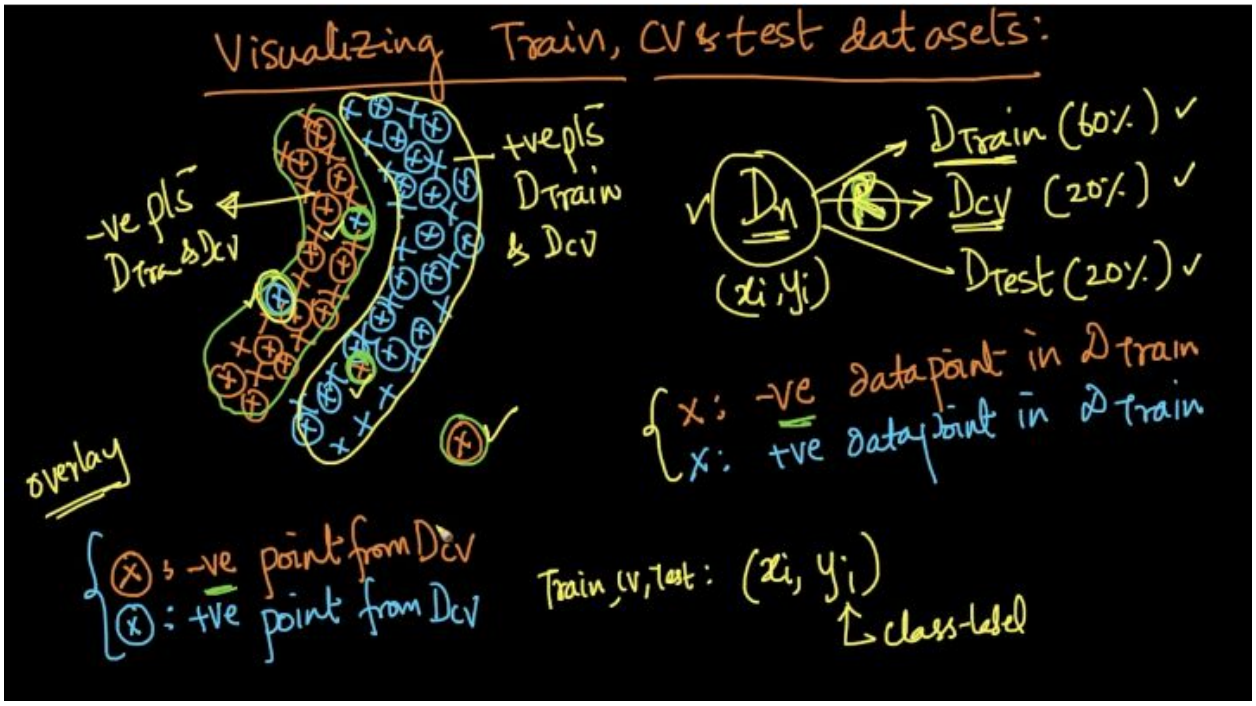
K-cross validation

Cross validation,generally the data is splitted into D(train),D(CV) and D(test) with 60%,20% and 20% of data respectively.But to achieve a good k-NN algorithm function,more data should be use to train instead of just 60%.Since we it is not right to change the content of D(test),therefore the datasets D(train) and D(CV) is mixed up to make it up of total 80% data.

Now from this 80% data we split it into 4 equal parts(D1,D2,D3,D4) and it is done randomly.And now from these parts we try to find best NN and k by taking three parts as training the data and 1 part as CV data and it is done for each and every combination of the 4 parts and also for every value of k.For example let say D1,D2 and D3 will act as training data and D4 is used as to find accuracy from the training data(D1,D2,D3) by taking value of k = 1, and let say we get accuracy from each iteration as a1,a2,a3 and a4.Now to find final accuracy for the value of k = 1,we take average of all accuracies.
And this same process is done for all k values like 2,3,4.....n.From this a plot is plotted and best k can be observed.

So in this way all parts are being used to get the NN and k.And this whole process is known as K-fold cross validation.And since in above example 4 parts has been used therefore it will be called as 4-cross validation.

Visualizing train, validation and test datasets



✓ (1) D_{train} & D_{cv} (& test) donot overlap perfectly when randomly sampled

(2) If there are many +ve pts from D_{train} in a region, then it is highly likely to find many +ve/-ve pts from D_{cv} in that region

vs

(3) If there are very few +ve/-ve pts in a region from D_{train}, then it is very unlikely to find +ve/-ve from D_{cv} in that region

noise, error, outlier

intuitively,

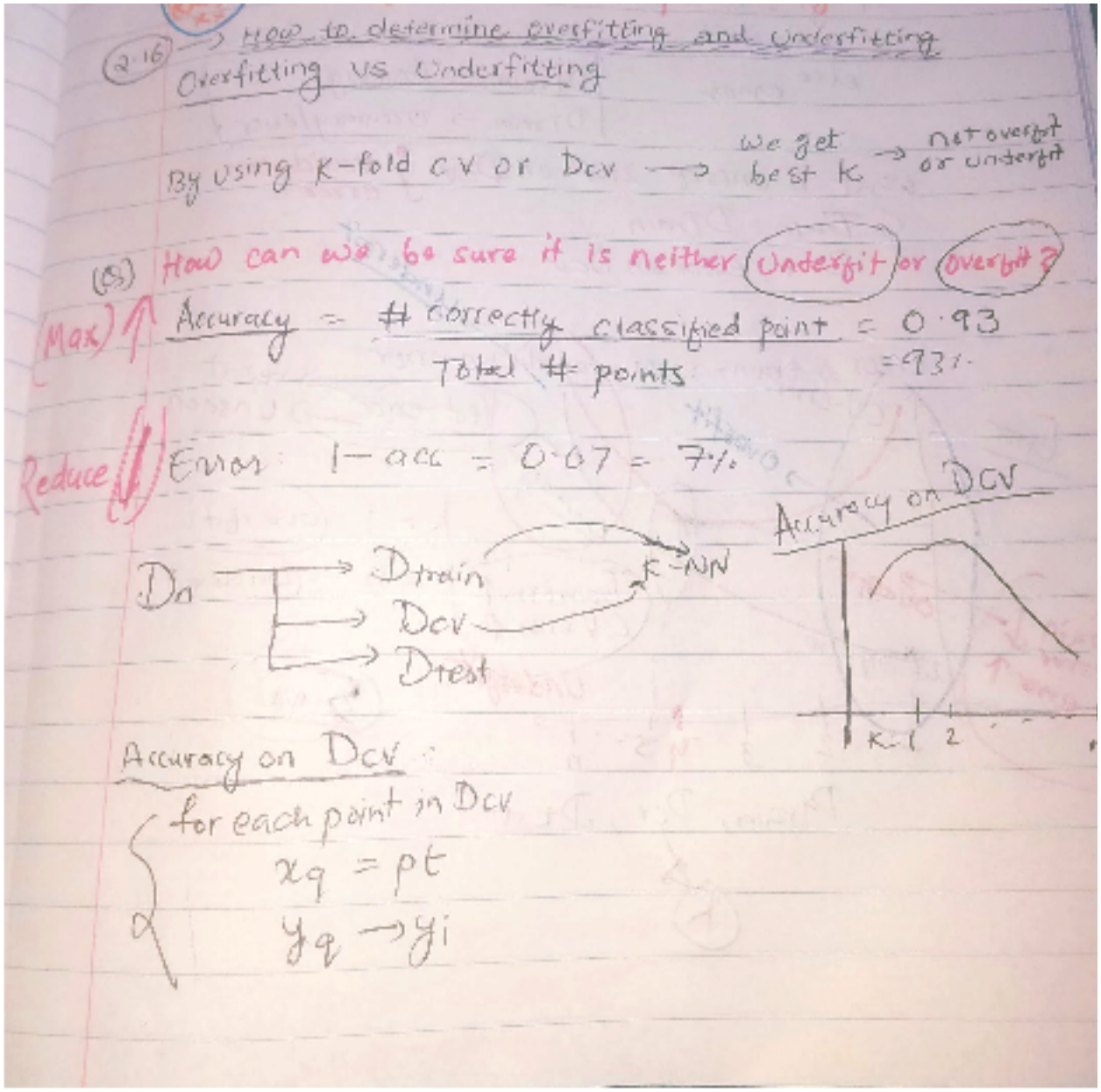
density of +ve pts from D_{train} is high

density of D_{train} -ve pts is high

→ very low density

↳ may not find any pts from D_{cv} +ve class

How to determine overfitting and underfitting?



Training error: [DTrain → NN, DCV → K, FCK-NN]

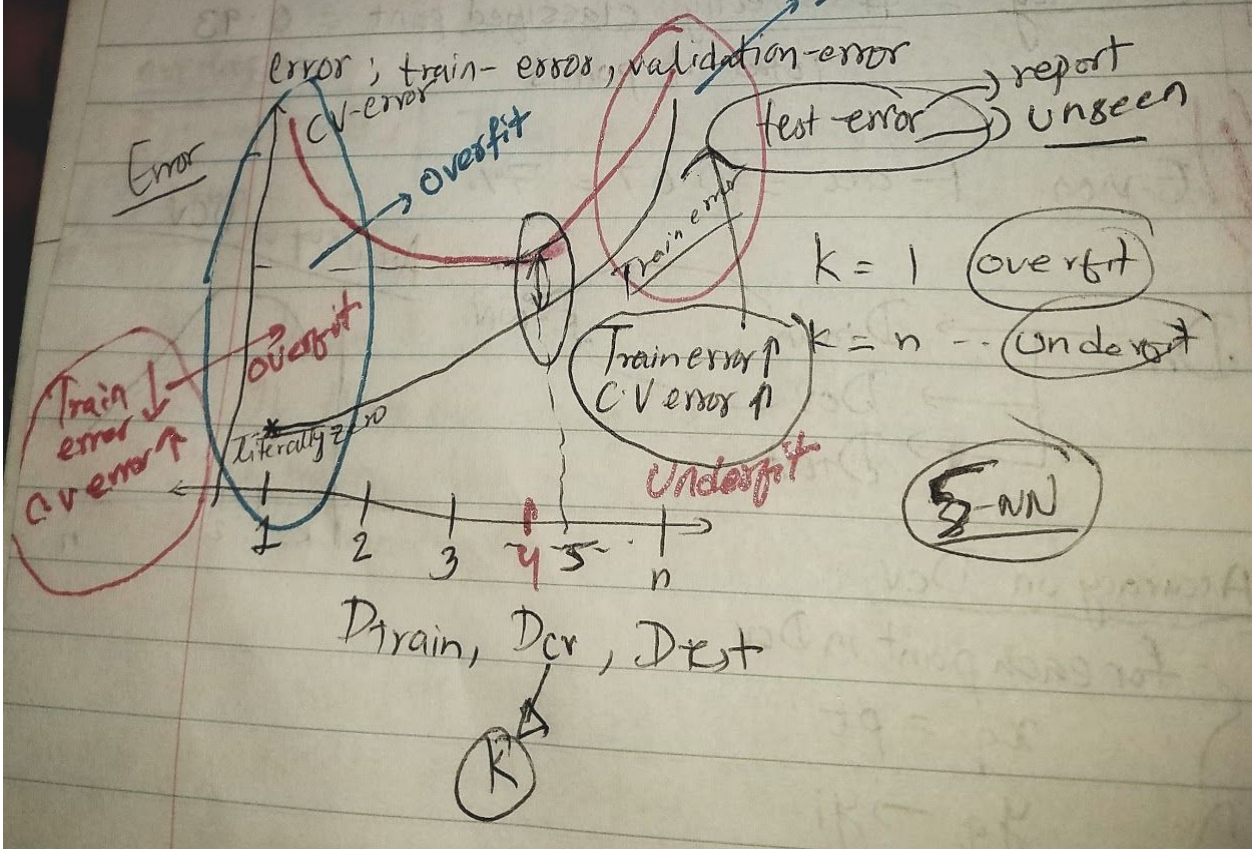
for each x_i in DTrain:
- find 2 nearest neigh to x_i from DTrain
- Majority Vot to get class label
- if y_i = class label
  accurate
  else error

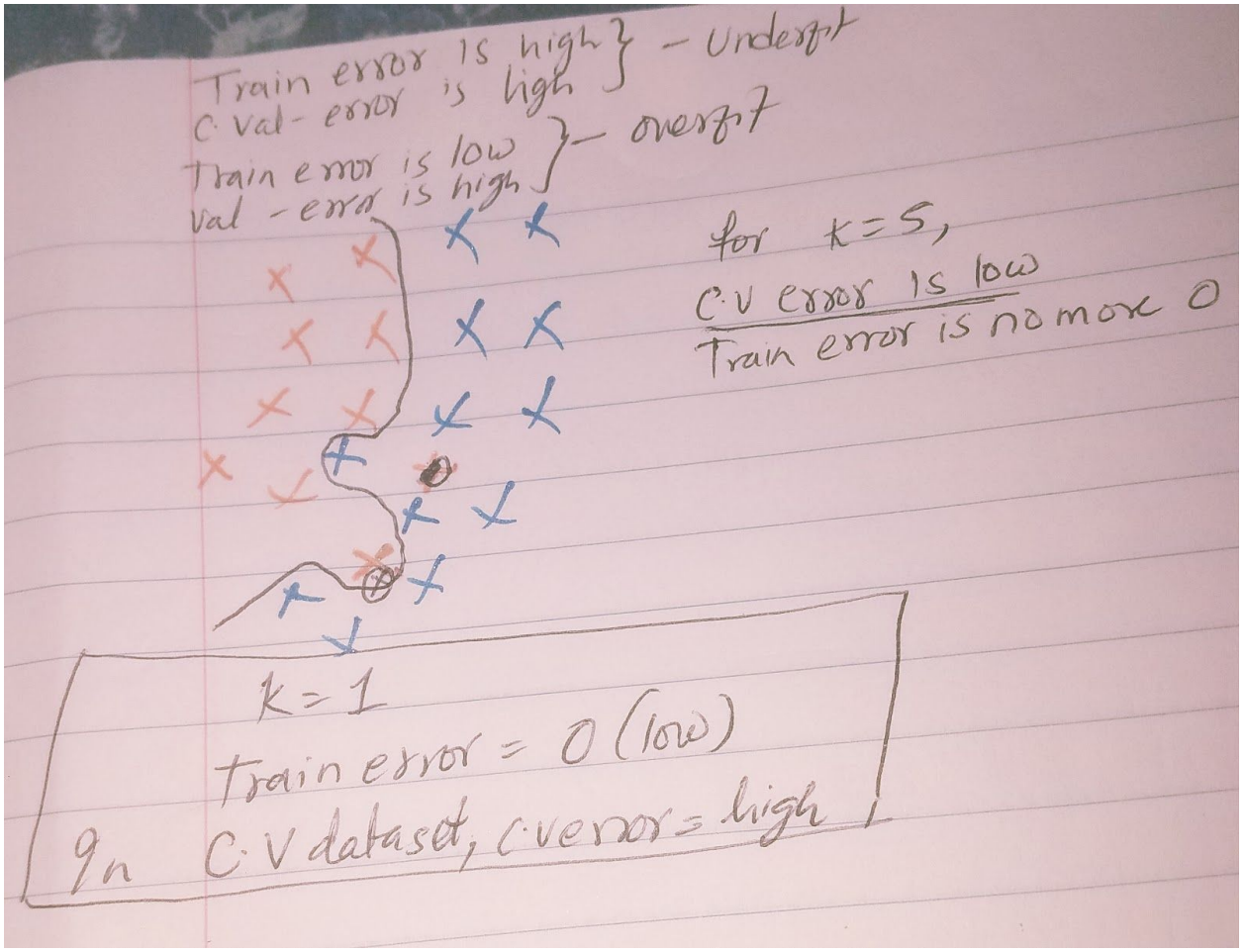
what is error (train error) if 2-NN ?

DTrain → Training
DTrain → accuracy/error

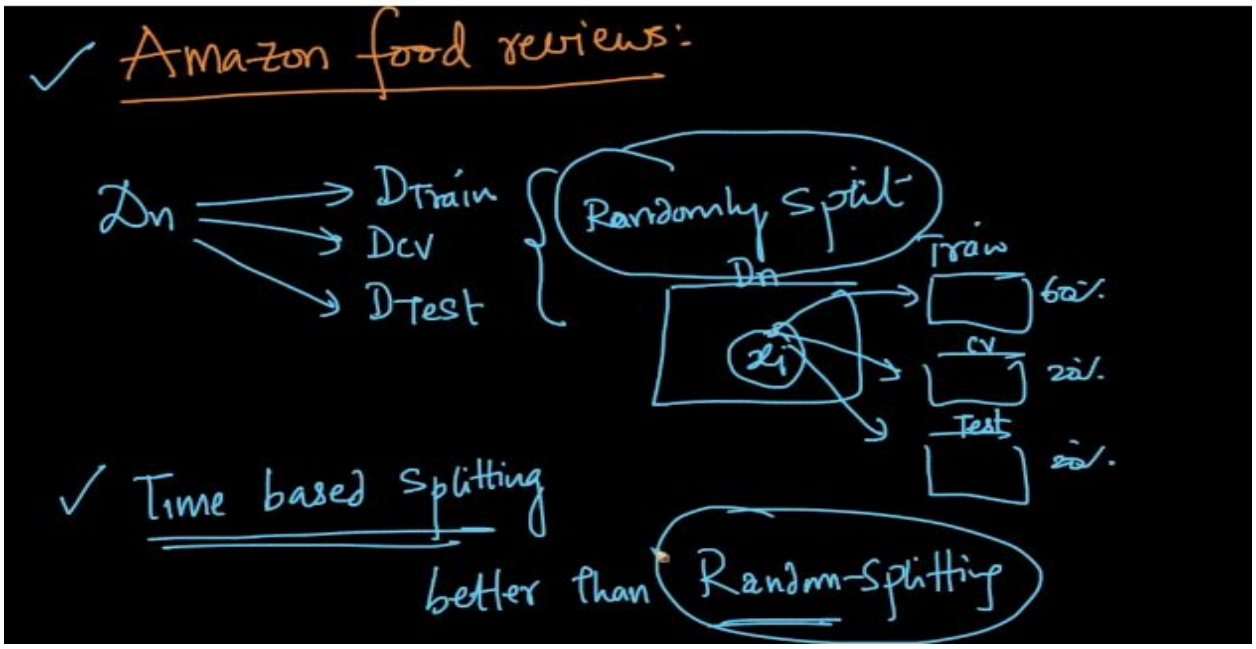
what is error of 2NN on DCV } validation error

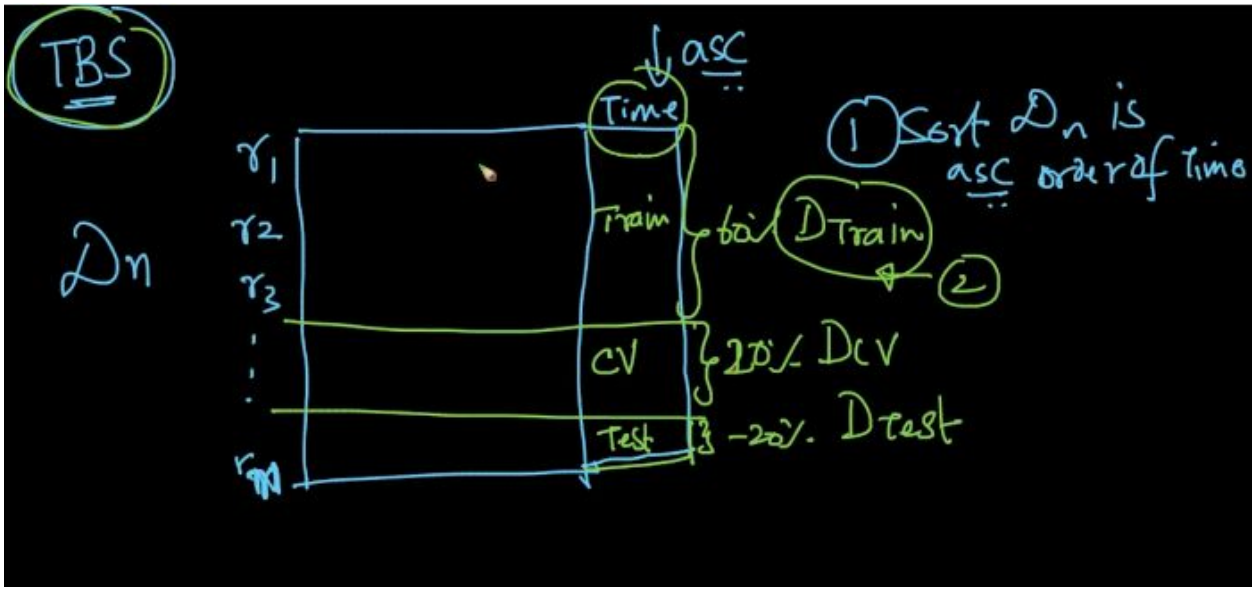
Train: DTrain
DCV: error on DCV

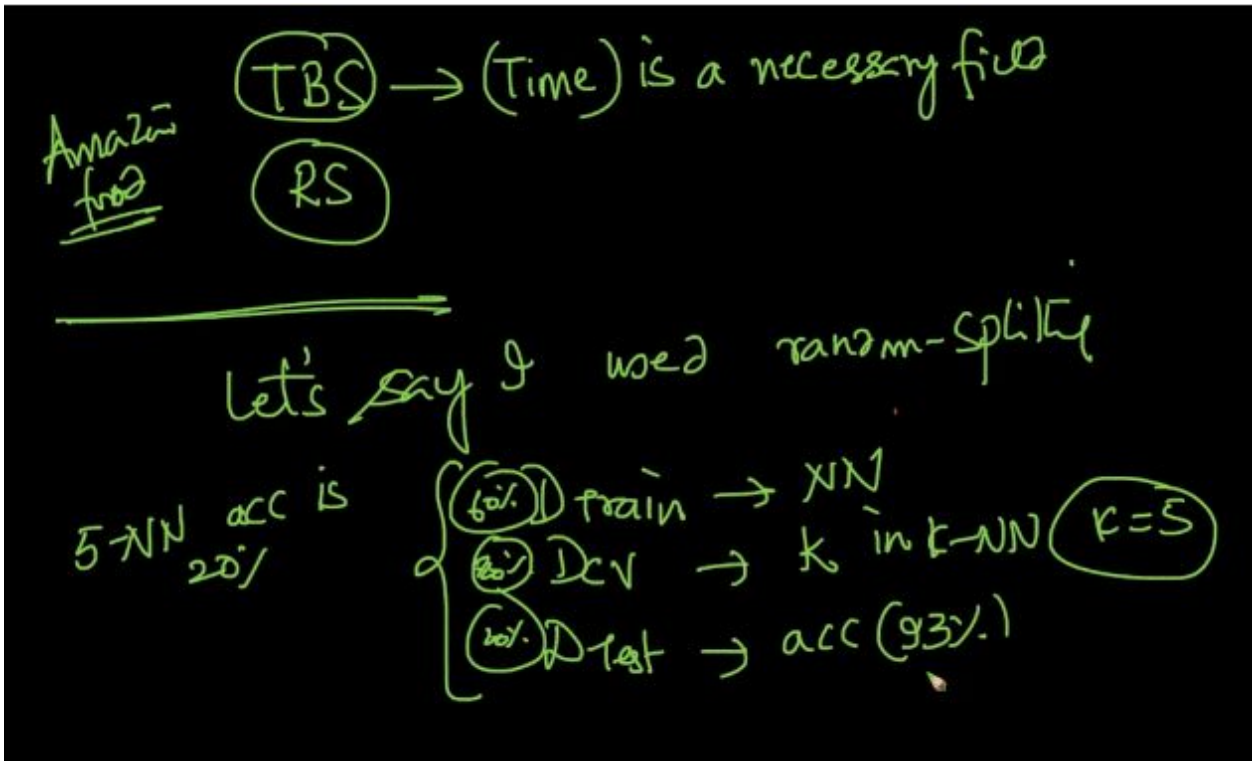


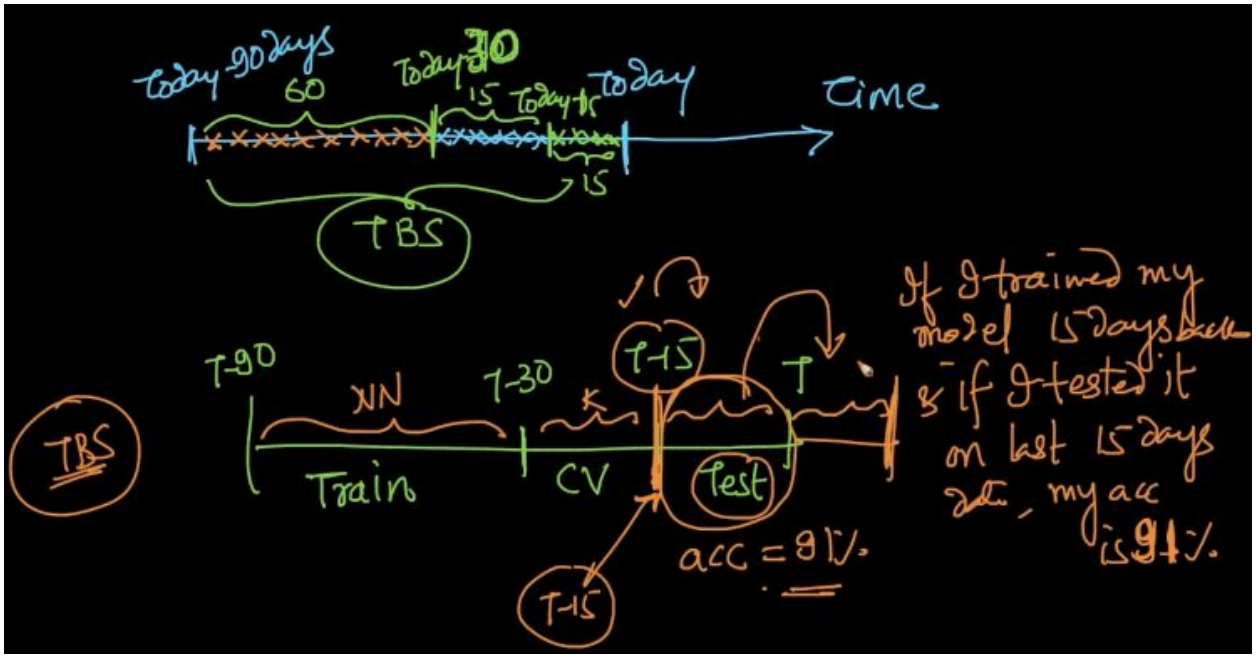


Time based splitting



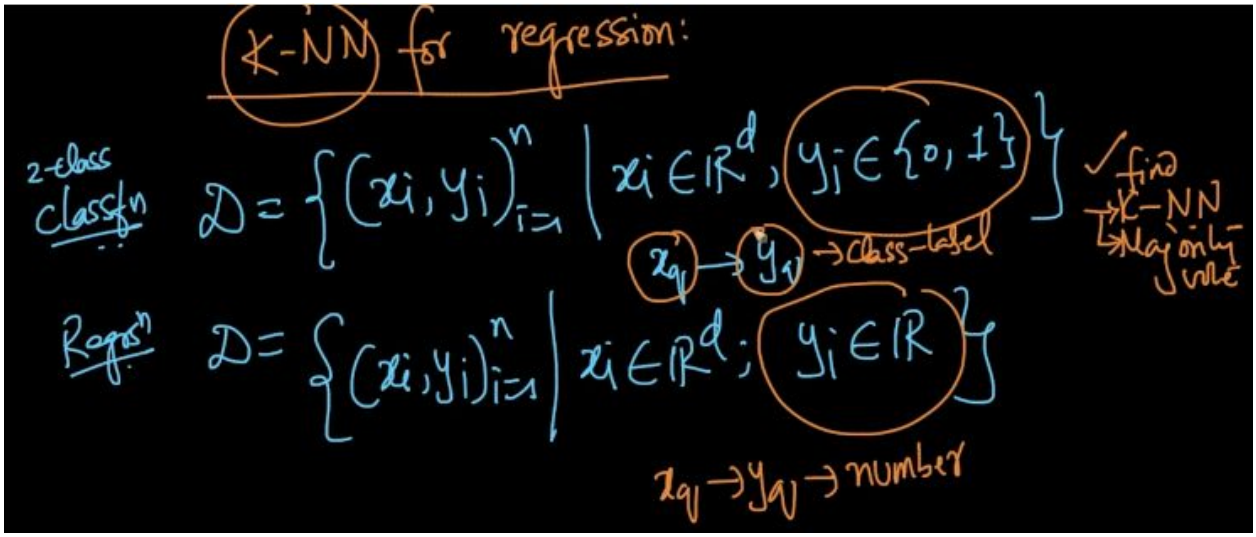

TBS
asc
Time
D_n
r_1
r_2
r_3
r_n
Train
CV
Test
60% D_{Train}
20% D_{CV}
20% D_{Test}
(1) Sort D_n is asc order of time
(2)

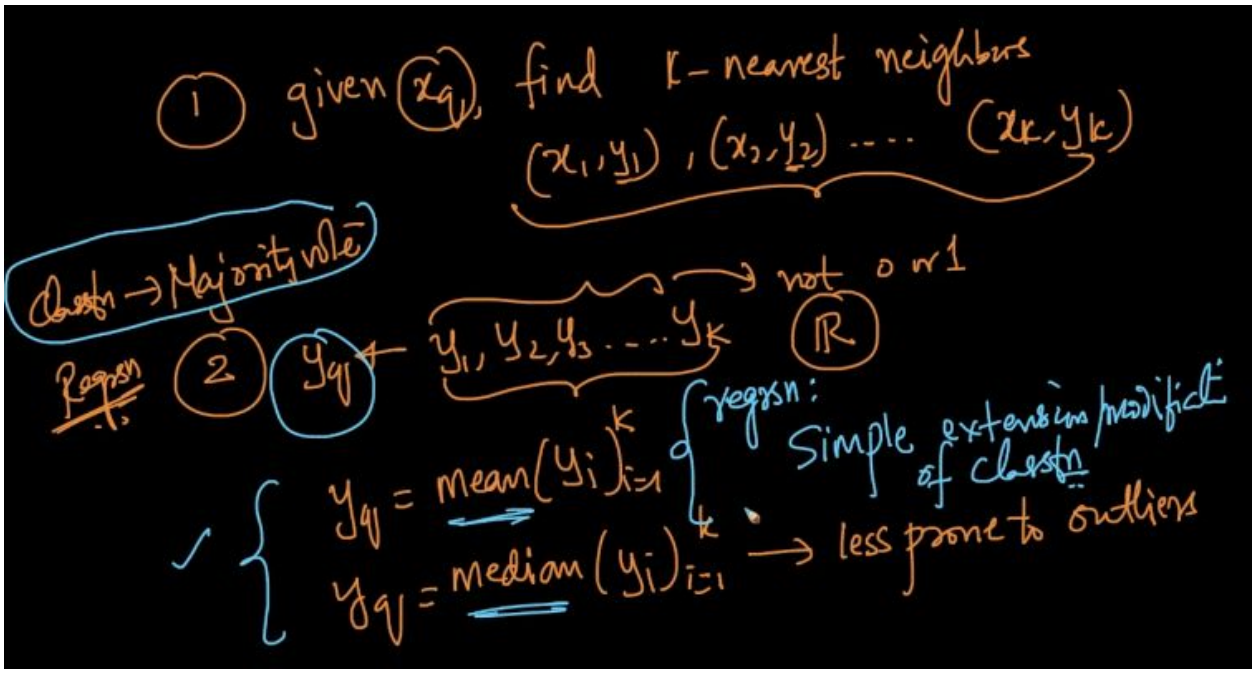

Amazing food
TBS → (Time) is a necessary field
RS
Let's say I used random-splitting
5-NN acc is 20%
60% Train → NN
20% D_{CV} → k in k-NN
20% D_{Test} → acc (93%)
k=5



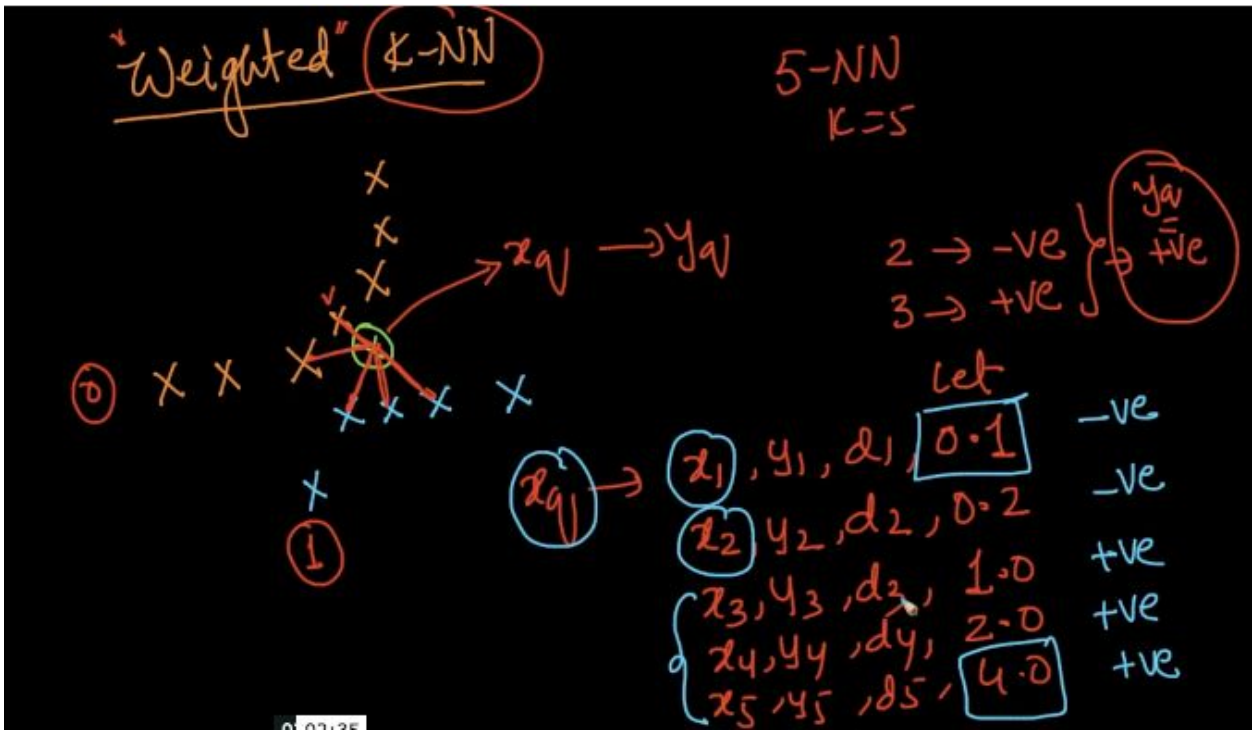
Whenever time is available and things/behavior or data changes over times then doing time based splitting is preferable instead of random splitting.

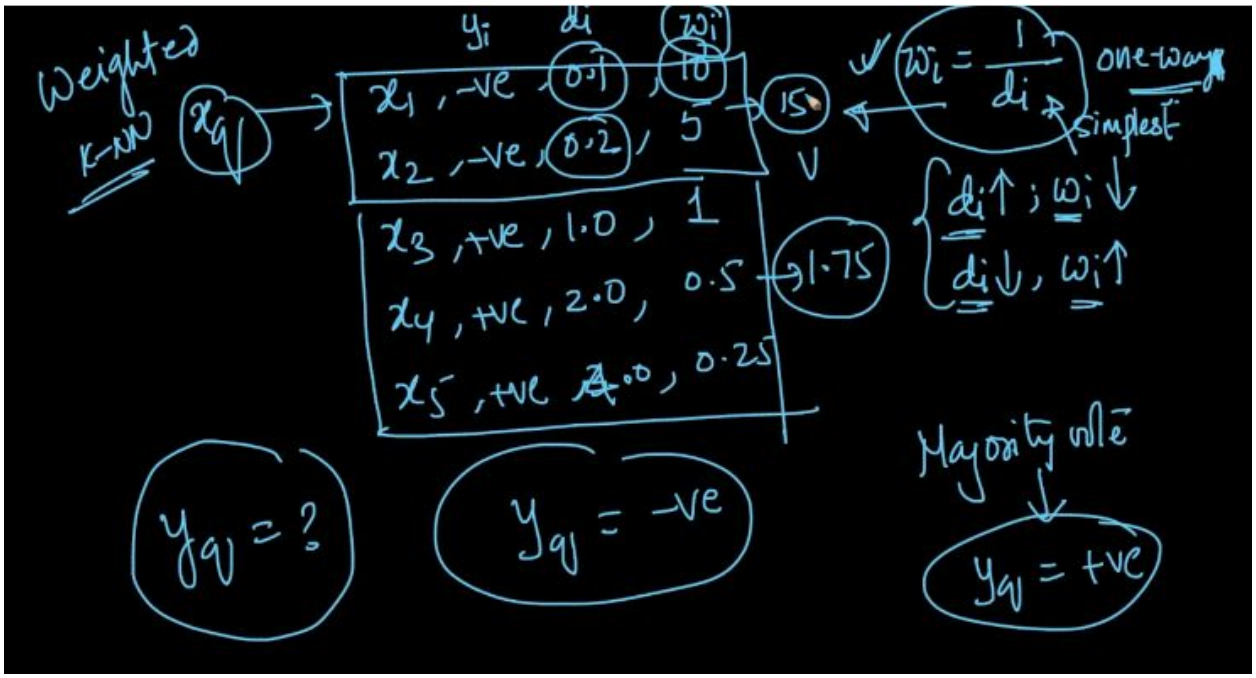
KNN For Regression





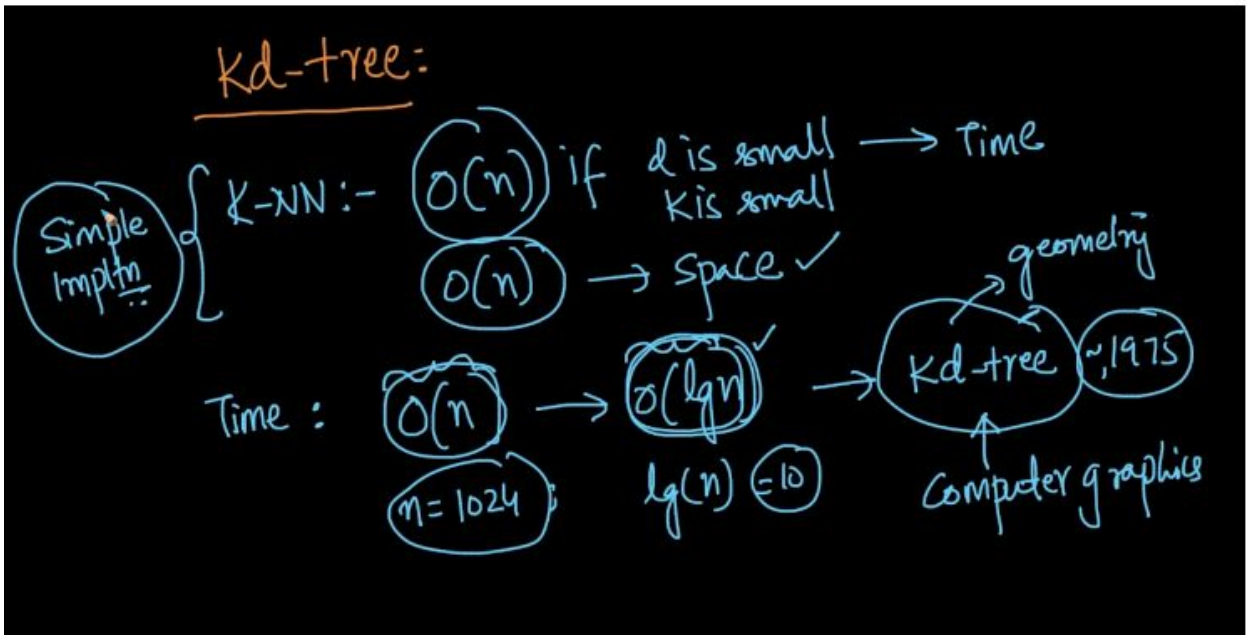
Weighted k-NN



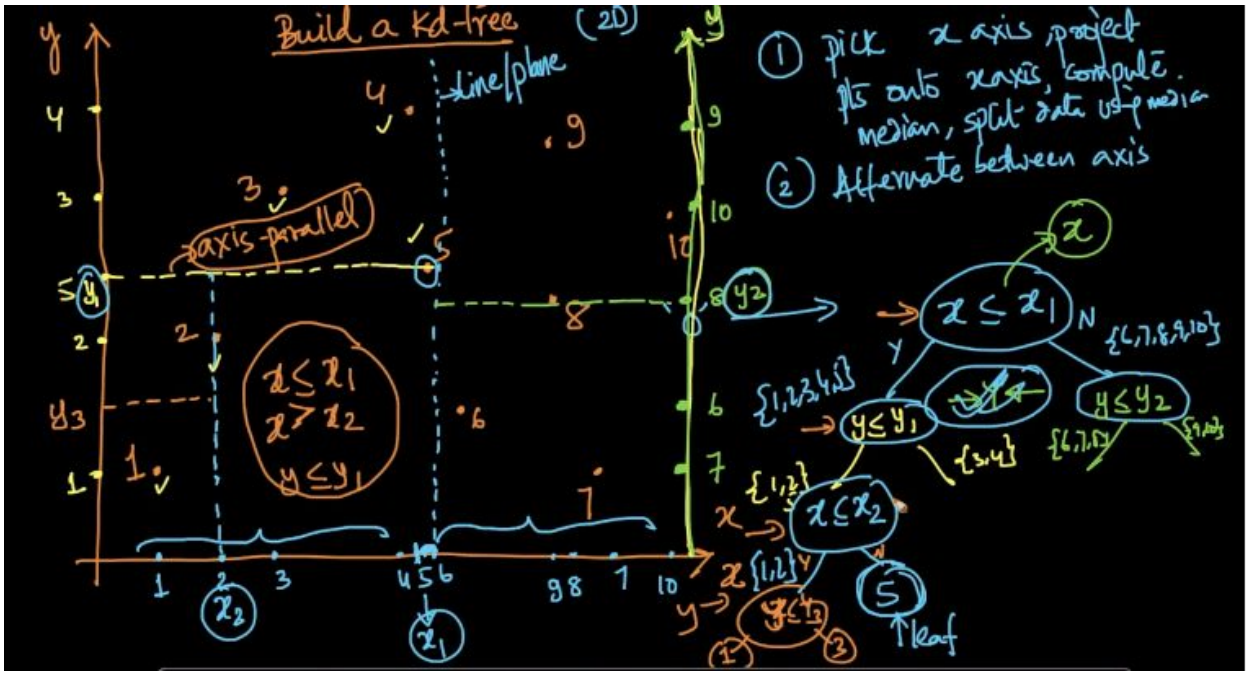


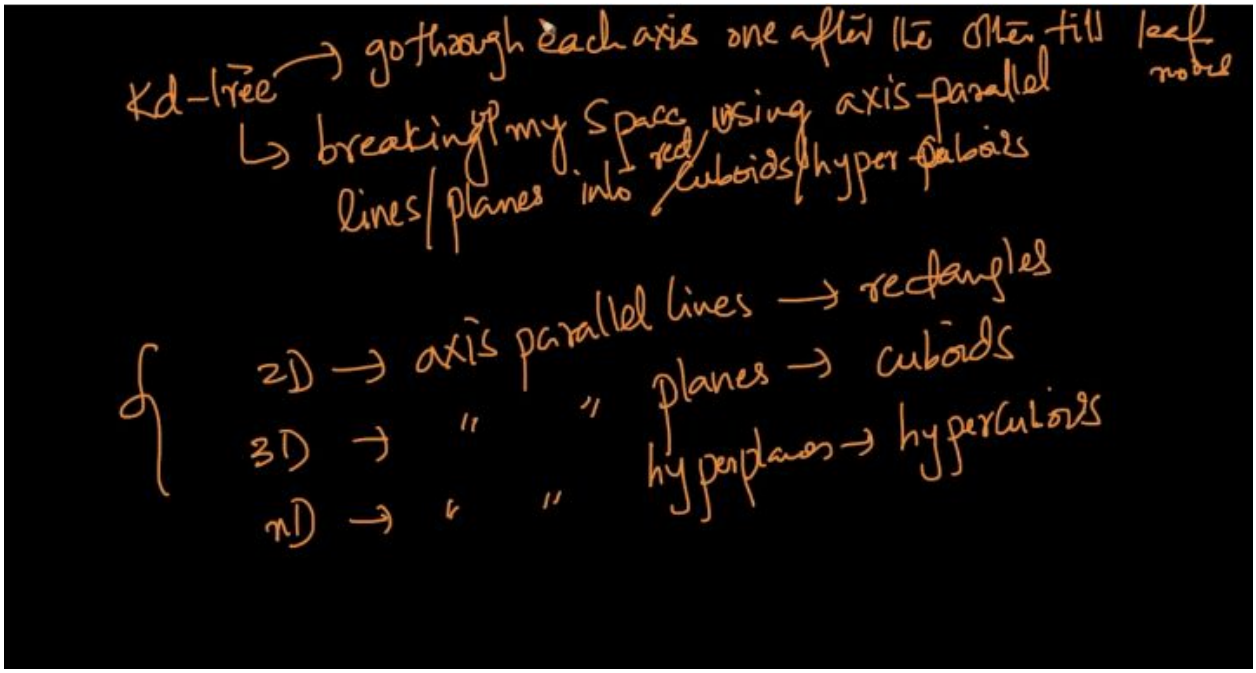
$W_i = 1/d_i$ (Simplest Weight function)

Binary Search Tree

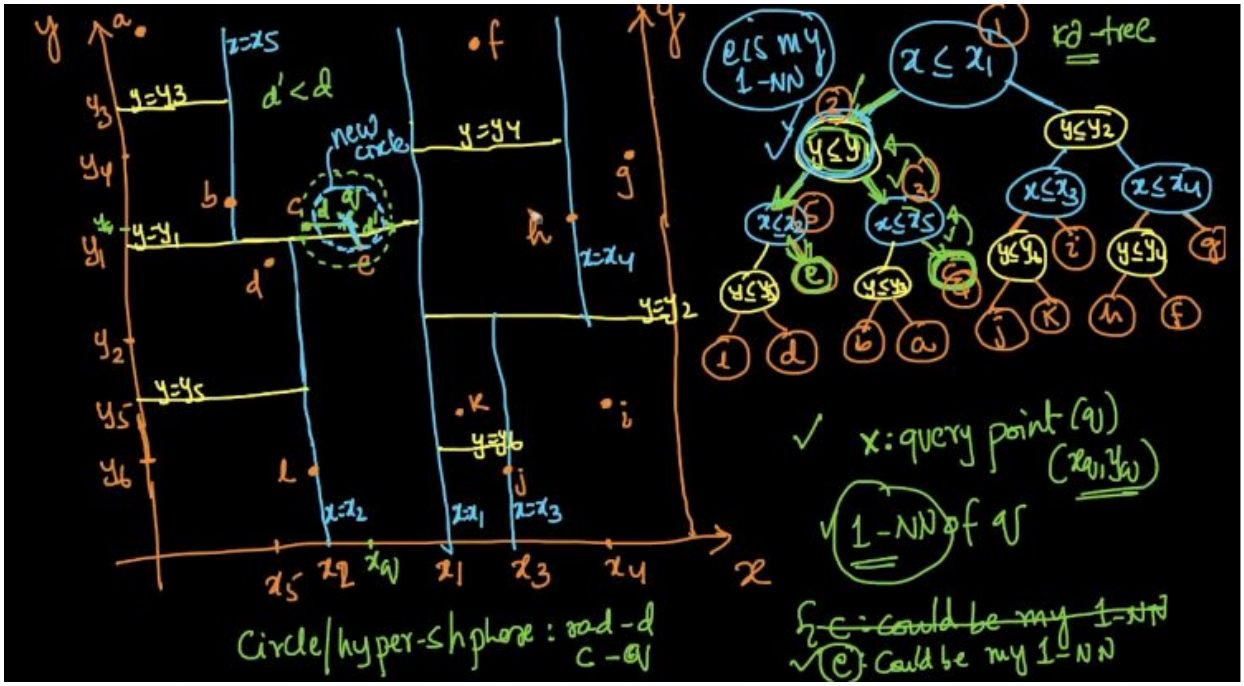


Kd-tree





Find nearest neighbours using kd-tree



Time-complx

{ # comparisions to find 1-NN

✓ best-case: $o(\lg(n))$ n: #pts

✓ worst-case: $O(n)$ $o(\lg n)$

{ perfect analysis :- very complex

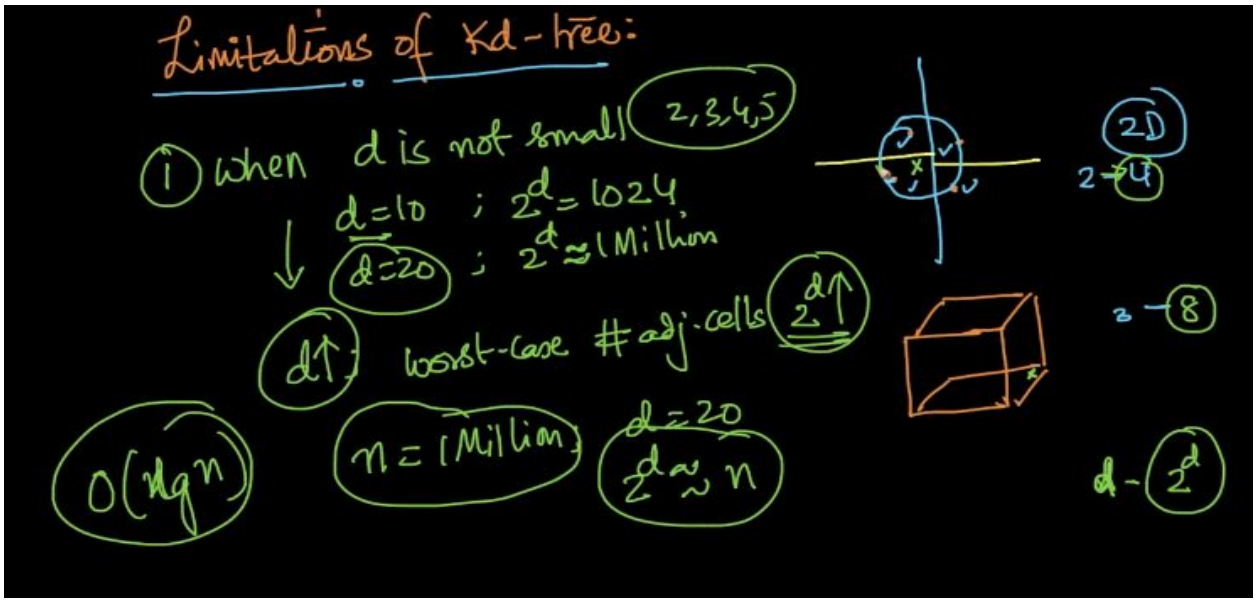
K-NN

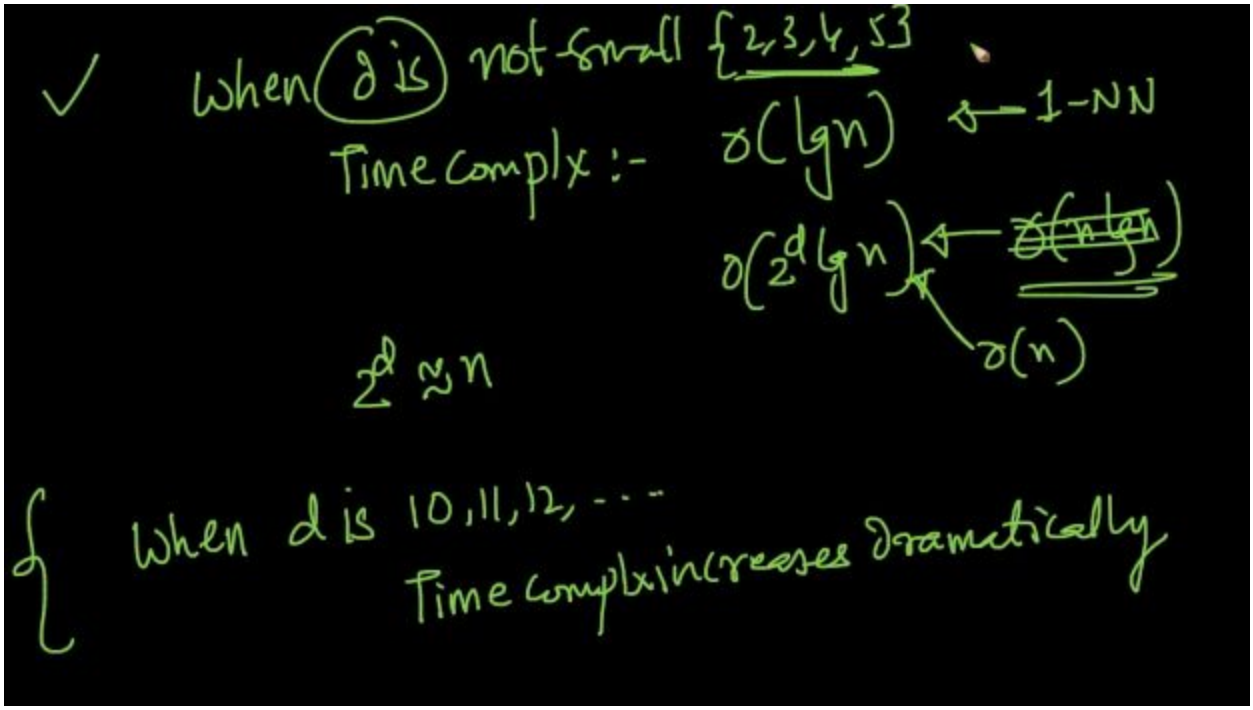
Time { best-case :- $O(k * \lg n)$ ~~~

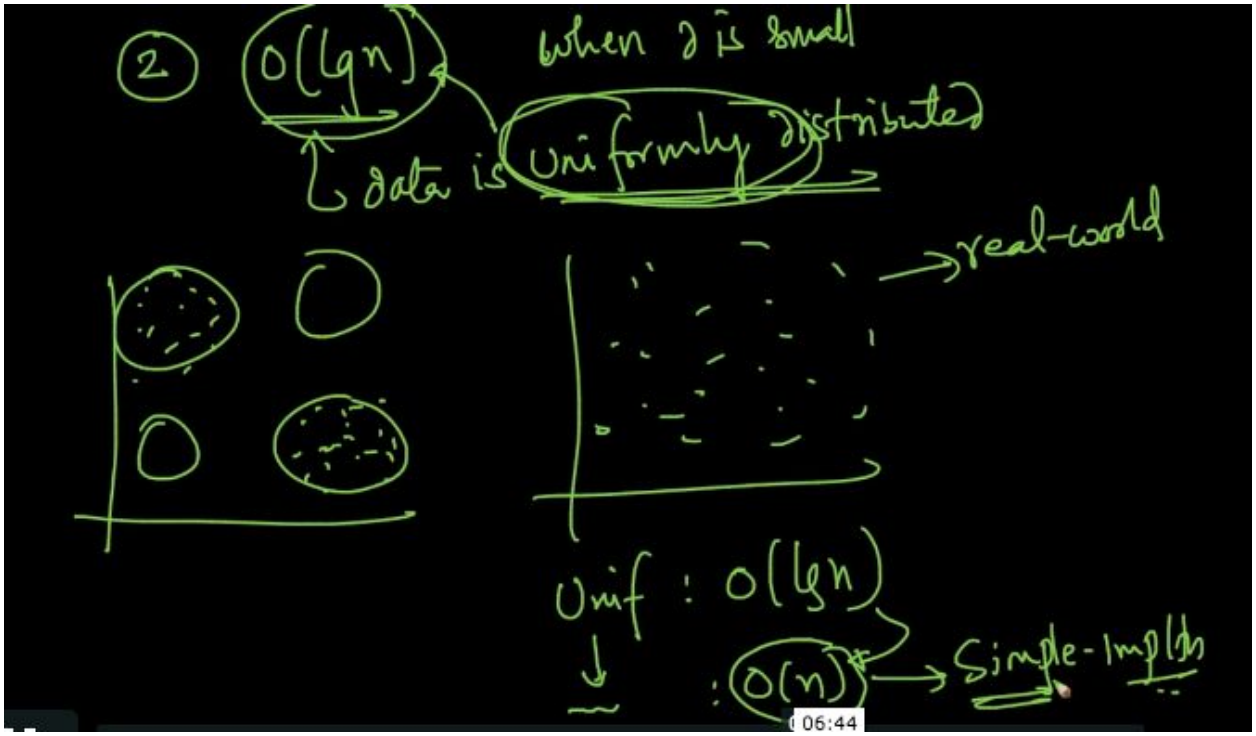
worst-case :- $O(k * n)$: if d is small

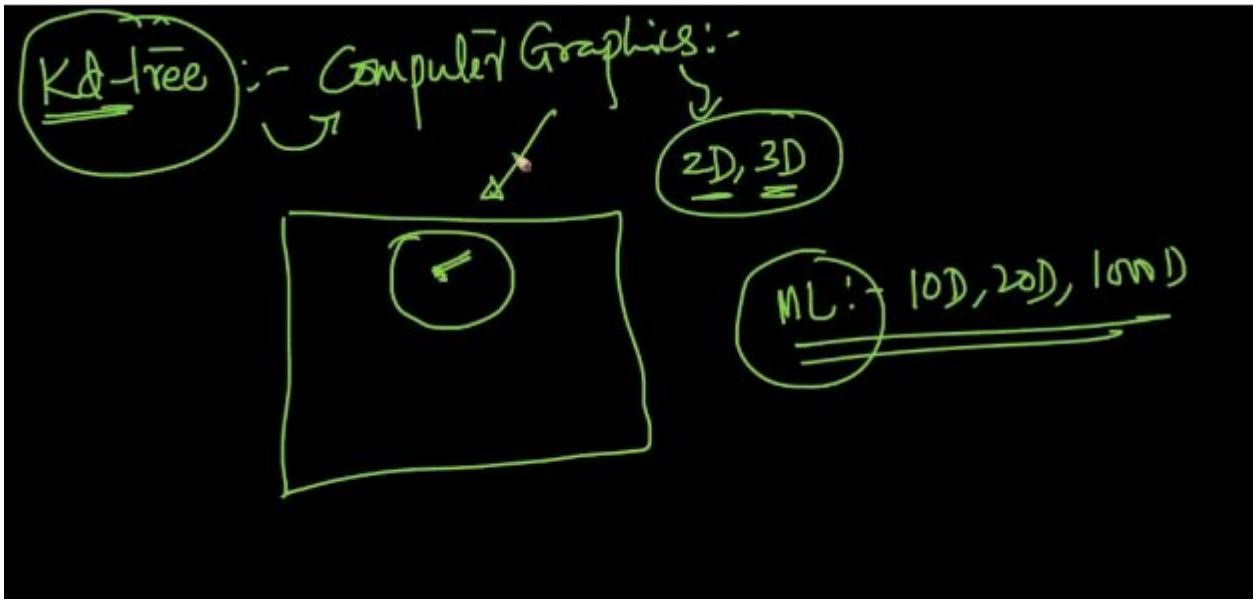
space :- $O(n)$

Limitations of k-d tree





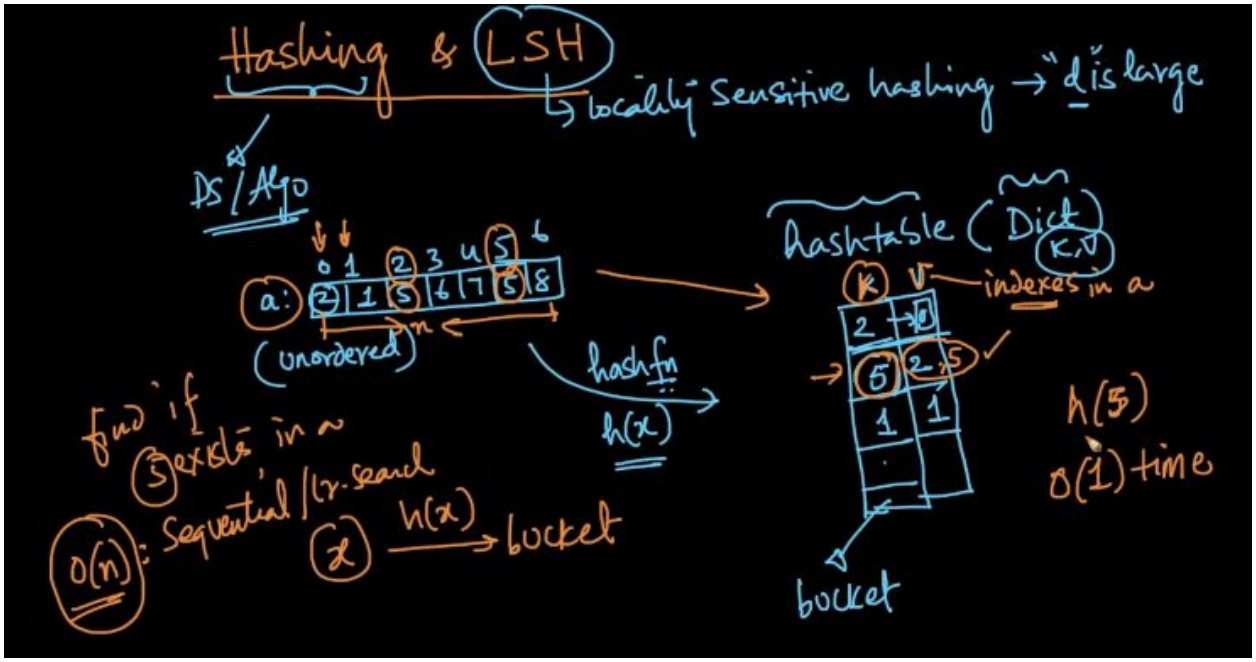


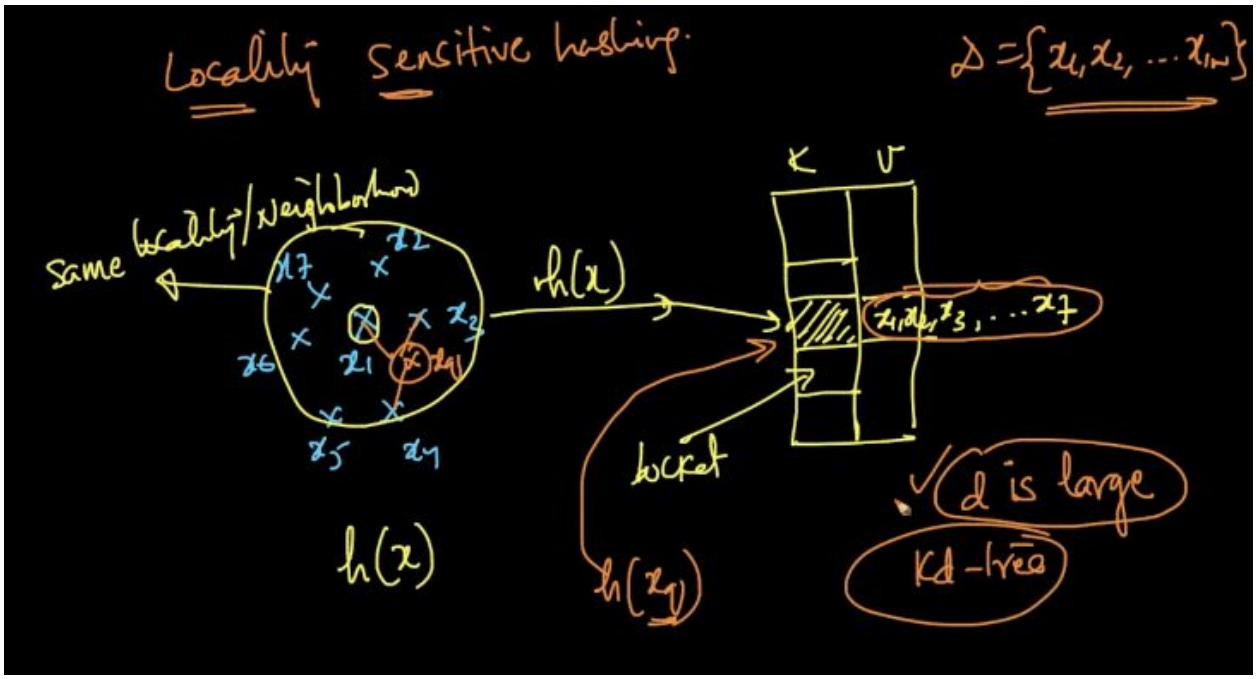


Extension : Variation in k-d tree

https://en.wikipedia.org/wiki/K-d_tree

Hashing


Hashing & LSH
↳ Locality Sensitive hashing → "d" is large
DS / Algo
a:
(unordered)
n
hash fn
h(x)
find if
(5) exists in a
: Sequential / Lr. search
O(n)
(x) h(x) → bucket
Hashtable (Dict)
K V
indexes in a
bucket
h(5)
O(1) time


Locality Sensitive hashing.
D = {x1, x2, ... xn}
Same locality / neighborhood
h(x)
K V
bucket
h(x)
h(xq)
d is large
Kd-tree