CLASSIFICATION AND Regression MODELS

K-Nearest Neighbors

18.1 How "classification" works?

Amazon fine food reviews

review → Text → Vector (Bow / tfidf / w2v)

classification:

we have 364K review $(r_i) \rightarrow (\vec{x_i}, +ve/-ve)$

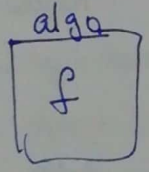
$\vec{x_i}$ → Vector representation

→ given a new review predict if the review is +ve & -ve

→ finding a function $y = f(x)$

+ve/-ve ← review text

classification algo working:



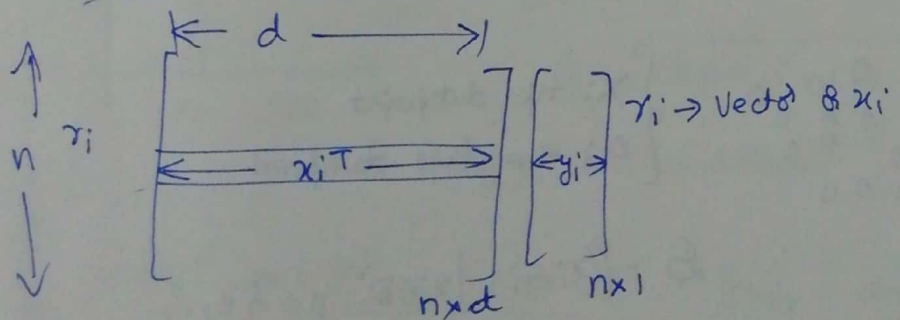
Training → Training stage.

(x_i, y_i)

$i = 1$ to 100K

Testing/evaluation $x_q \rightarrow f \rightarrow y_q$

18.2 Data matrix notation:



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

D → set, x_i → Euclidean, y → set

18.3 Classification Vs Regression (Examples)

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

Amazon food reviews

There are two classes (2 class classification) (binary classification)

MNIST :

$y_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ → 10 class / multiclass classification

what if $y_i \in \mathbb{R}$

↳ y_i is no more part of a small finite set of classes

Ex :

$i = 1 \rightarrow 10K$

x_i : <weight, age, gender, race>

y_i = height (real number)

$y_i = f(x)$

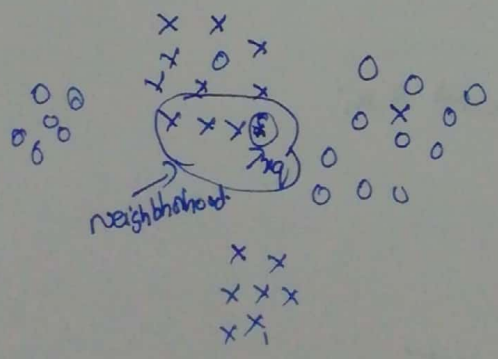
where y_i is a real number then it is called Regression algorithm.

18.4 K nearest neighbours Geometric intuition with a toy example

K-Nearest Neighbors (KNN)

2D toy dataset

Let's take a care of binary classification



x : +ve datapoint

o : -ve class datapoint

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

gear : since x_q is close to (x) we can conclude that x_q is positive

Steps

① Find K nearest points to x_q is ∅

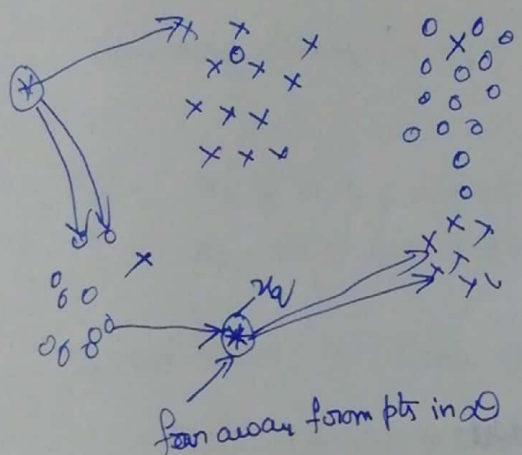
Let K=3 3 nearest points are (x_1, x_2, x_3)

② $y_1\ y_2\ y_3$

② Take all the class labels (y_1, y_2, y_3)

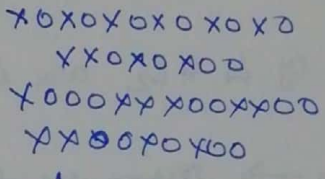
Based on the majority we will declare the class

18.5 Failure Cases of KNN



→ As the x_q is far away, then it is very good say don't know

→ There is useful information

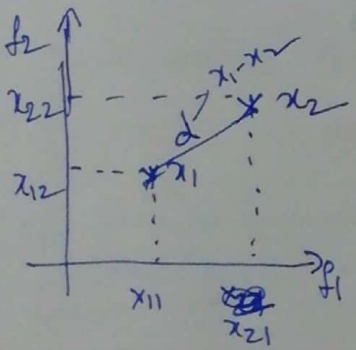


→ Jumble of +ve/-ve

→ Randomly spread

→ There is no useful information

18.6 Distance measure: Euclidean (L2), Manhattan (L1), Minkowski, Hamming.



$x_1 = (x_{11}, x_{12})$ $x_2 = (x_{21}, x_{22})$ (f_1, f_2)

d = len of shortest line from x_1 to x_2

$$d = \sqrt{(x_{21}-x_{11})^2 + (x_{22}-x_{12})^2} = \|x_1 - x_2\|$$ (length)

→ by Pythagoras theorem. Euclidean distance.

$x_i \in \mathbb{R}^d$, $x_2 \in \mathbb{R}^d$

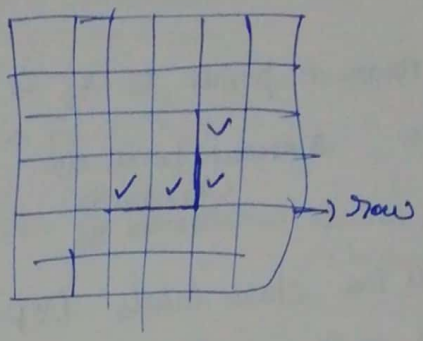
$$\|x_1 - x_2\| = \sqrt{\left(\sum_{i=1}^{d} (x_{1i} - x_{2i})^2\right)}$$

$\|x_1 - x_2\|_2$ = L_2 norm

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

Manhattan dist :-

$$\sum_{i=1}^{d} \underbrace{|x_{1i} - x_{2i}|}_{\text{abs}}$$



↳ norm of vector $(x_1 - x_2)$

$$||x_1 - x_2||_1$$

↳ norm of x_1 is

$$||x_1||_1 = \sum_{i=1}^{d} \underbrace{|x_{1i}|}_{\text{abs}}$$

Generalization of L_1 & L_2 are L_p norms

L_p - norm → Minkowski dist

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

If $P=2$ → minkowski dist ⇒ Euclidean dist

If $P=1$ → " → Manhattan distact

$$||x_1||_p = \left(\sum_{i=1}^{d} |x_{1i}|^p \right)^{1/p} \qquad P \neq 0 \\ > 0$$

→ Distances are always computed blw two points and norms are always for a vector.

Eucdist (x_1, x_2) = 22 norm of $(x_1 - x_2)$

$$= ||x_1 - x_2||_2$$

$$\text{manhattan}(x_1, x_2) = ||(x_1 - x_2)||_1$$

$$\text{mikowski}(x_1, x_2) = ||(x_1 - x_2)||_p$$

Hamming Distance

→ This is used in text processing when we have a boolean vector

x_1, x_2 → boolean vector → Binary Bocw

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

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

Hamming distance (x_1, x_2) = # location/dimen where binary vectors differ

Hamming distance $(x_1, x_2) = 4$

Ex:2 $x_1 = a\ b\ c\ a\ d\ e\ f\ g\ h\ i\ k$

$x_2 = a\ c\ b\ a\ e\ d\ e\ g\ f\ h\ i\ k$

If we have a Gene codes/seq of AGTC characters

$x_1 = AAGTCTCAG\ldots$

$x_2 = AGATCTCCA\ldots$

18.7 Cosine distance & Cosine Similarity

Similarity vs distance

→ x_1, x_2 distance increases, similarity decreases

→ " decreases " increases

$$\text{Cosdist}(x_1, x_2) = 1 - \text{Cossim}(x_1, x_2)$$

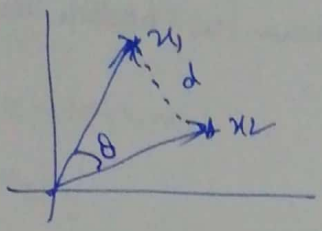
Cossimlai is b/w $[-1, 1]$

→ Cossim of $(x_1, x_2) = +1$ when they are very similar

→ " $= -1$ " very dissimilar

Cosine Similarity

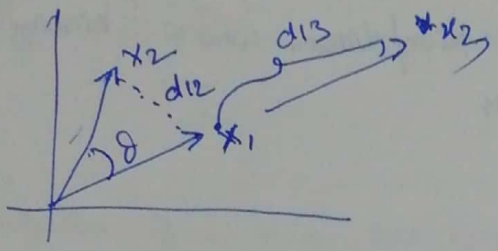
d = Euc. dist



Cos-sim = Cos θ

where θ is angle blw x_1 & x_2

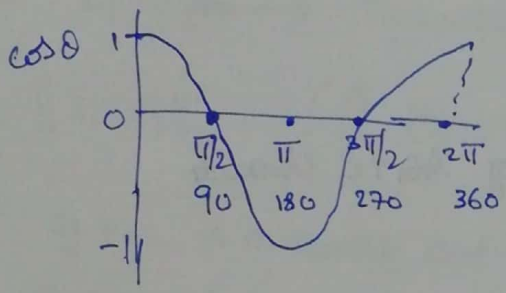
→ Difference blw Euc. dist & Cos similarity



Cos-sim (x_1, x_2) = Cos θ

Cos-sim (x_1, x_3) = 1

$\theta_{x_1, x_3} = 0°$



Cosdist (x_1, x_3) = 1 − Cos-sim (x_1, x_3)

= 1 − 1

= 0

Ex

$$\cos\theta = \frac{x_1 \cdot x_2}{\|x_1\|_2 \, \|x_2\|_2}$$

① If x_1 & x_2 are unit vectors (length = 1)

$$\cos\theta = x_1 \cdot x_2$$

Relation blw Euc dist Vs Cos-sim

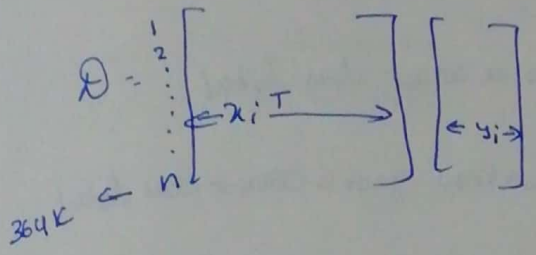
If x_1 & x_2 are unit vectors

$$\left[\text{eucdist}(x_1, x_2)\right]^2 = 2\underbrace{(1 - \overbrace{\cos(\theta)}^{\text{Cos sim}})}_{\text{Cos dist}}$$

$$= 2(\text{Cos dist}(x_1, x_2))$$

18.8 How to measure the Effectiveness of KNN

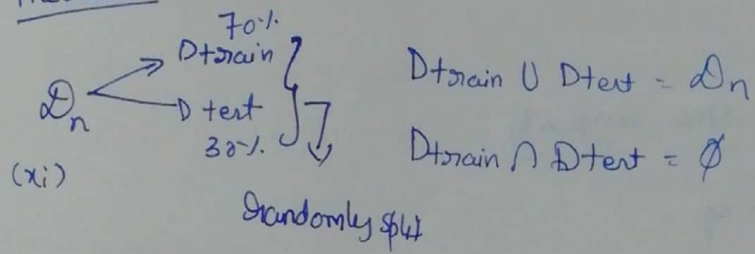
Amazon Fine food reviews

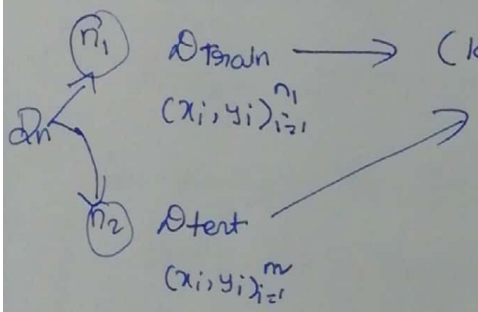


Problem : Given a new food review (x_q) what is its polarity (+ve/-ve)

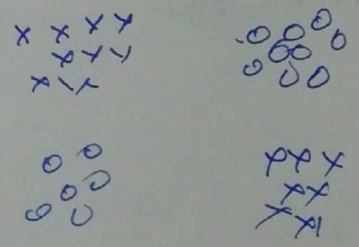
$x_q \rightarrow Text_q \rightarrow (x_q)$

measure :





what is training in D_{train} ?



Each point x_i in $D_{test} = (x_i, y_i)_{i=1}^{n_2}$

$x_q = (x_i)$

for each point in d-test x_q = pt we use D_{train} & KNN to determine.

if $y_q == y_{pt}$

cnt += 1

End

Count = # of pts for which D_{train} + knn gave a correct class label

Accuracy = $\frac{\text{Count}}{n_2}$ → # pts for which D_{train} + knn gave a correct class label

→ pts in D_{test}

$0 \leq Acc \leq 1$

Acc = 0.91 = 91% of times it is predicting correctly

18.9 Test/Evaluation time & space Complexity

we are given a query point x_q

$x_q \rightarrow y_q$

Input = D_{train}, k, $x_q \in \mathbb{R}^d$; output = y_q

knnpts = [ ]

for each x_i in D_{train}: → n pts : d-dim

- Compute $d(x_i, x_q) \rightarrow d_i$
- → Keep the smallest K distance → $(x_i, y_i, d_i) \in$ tuple. → knnpts [ ]

K is small (5 or 10)

Count_pts = 0, Cnt_neg = 0

for each x_i in knnpts

if y_i is +ve

Count_pts += 1

Else

Count_neg += 1

If Count_pts > Cnt_neg

return $y_q = 1$

Else $y_q = 0$

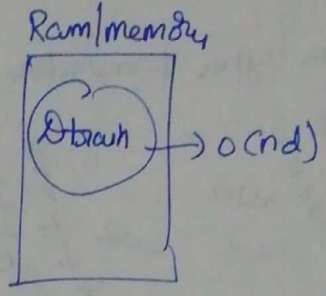
Time Complexity : $O(nd) + O(1) + O(1)$

Amazon food reviews

$n \approx 364K$

$d \approx 100K$ (BoW)

300 (tfidf)

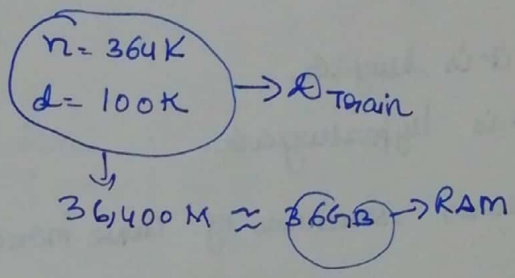


18.10 Know Limitations

Large space Complexity

Time Complexity : $O(nd)$

Space : $O(nd)$



Time Complexity

36 Billion Computation.

Review $\xrightarrow{1ms}$ +ve/-ve. (Internet application)

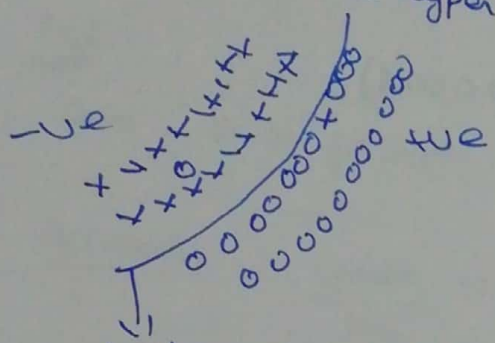
Low-latency : Time it take to predict 1q give 1q (It should be fast) Ex: Trading, Finance

→ Simple Implementation we have know → Time $O(nd)$, Space $O(nd)$

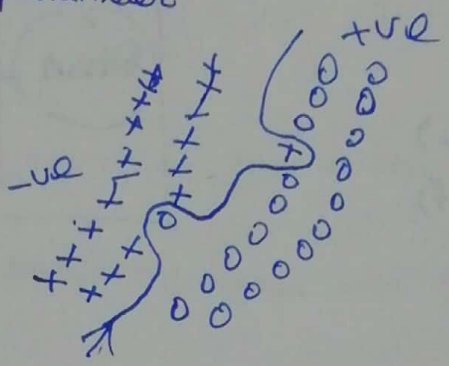
→ Ppl don't use know because of Time & Space Complexity

18.11 Decision Surface for K-NN as K changes

→ Here K is a hyper parameter



Line/Curve (2 Errors)



Curve (no Errors)

→ There curve +ve from -ve & vice versa are called decision surfaces

→ In 3D it is surface

→ In nD it is hypersurface.

→ A K increases smoothness of curve increase.

Lets say $K = n$ when $K = 1, 2 \cdots (n)$

n = Total number of points

n_1 = +ve (600)

n_2 = -ve (400)

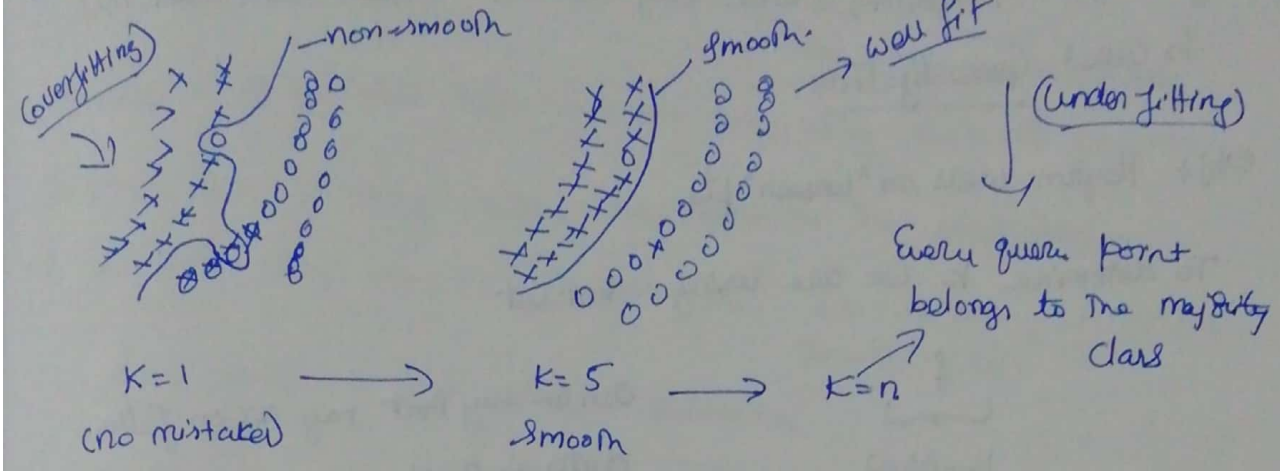
$n_1 + n_2 = n$

Let's say $n_1 > n_2$

If my K = 1000, 1000 neighbours ~~are~~ should be considered which means we have to consider all the points.

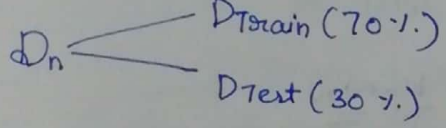
Hence here the majority class is n_1 our model will predict Every input as positive only

18.12 Overfitting & underfitting



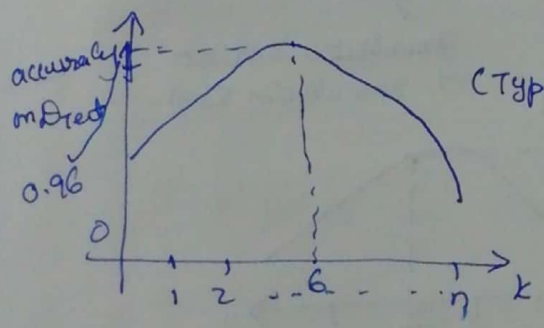
fitting because $f(x_q) = y_q$. The process of finding a function is called fitting

18.13 Need for Cross Validation



one-idea :

	Train	accuracy on Test
K=1	D_{Train}	0.78
K=2	"	0.82
K=3	"	0.85

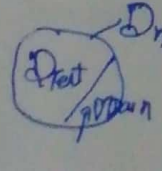


(Typically, need not be the same)

K=6, give me the best accuracy when using the D_{Train}.

K=6 → (6-nn) → 96%.

Using D_{Train} & 6-nn on Amazon reviews dataset It got an acc of 96%.

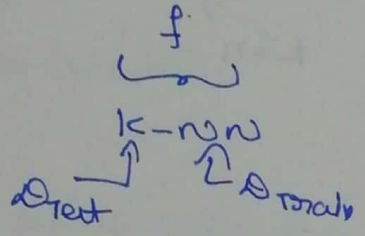


Obj : For a future unseen point x_q how accurately i can predict y_q

→ If a function/algorithm work very well on unseen data then this is called Generalization

Obj: Perform well on "unseen" pts

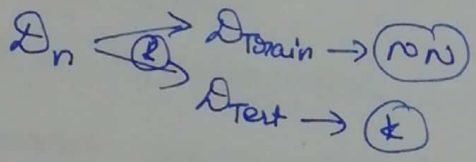
To determine k we are using Test set

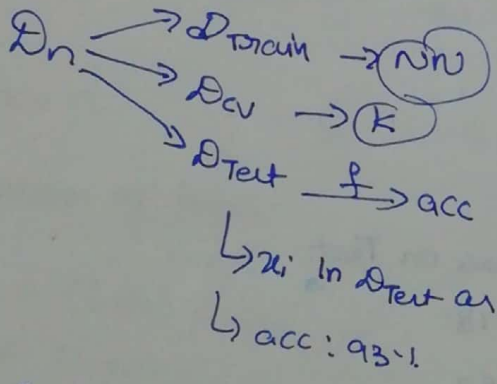


Can we say that my acc on future data is 96%.

Ans: we cannot guarantee

To overcome this problem we use Cross Validation.





} f: k-nn

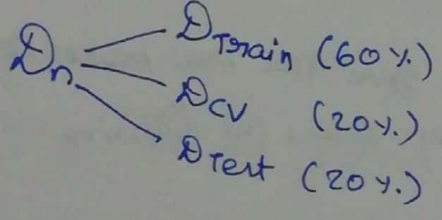
↳ did not use D_{test} to compute f

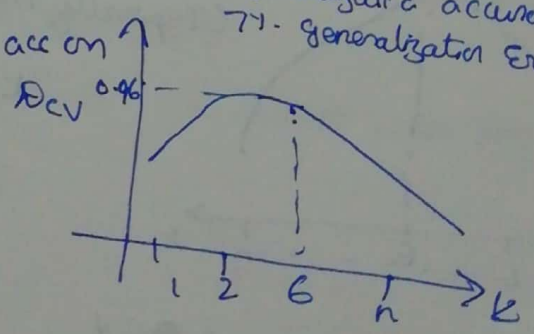
→ Can we say that 6-nn has an acc of (93%) on unseen data?

↑ generalization accuracy

% generalization Error.

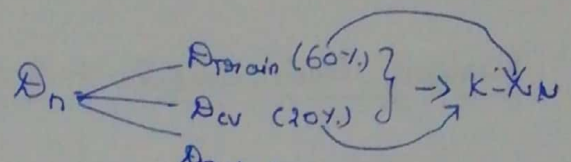
* Yes





19.14

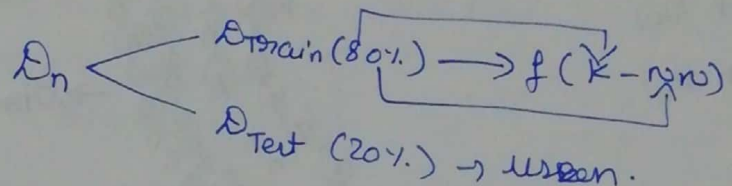
K-Fold Cross validation



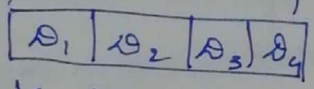
Problem : of the total data we are only using 60% to find NN

→ More the Training data, the better is the algorithm.

→ Is there any way to combine the 60% & 20%, it can be done using k-fold cross validation.
(Train) (CV)

① 

② D_{Train} 4 Equal Parts



→ This is called 4-fold cross validation.

	Train	CV	acc. on CV
4 time { k=1	$D_1 D_2 D_3$	D_4	a_4
k=1	$D_1 D_2 D_4$	D_3	a_3
k=1	$D_2 D_3 D_4$	D_2	a_3
k=1	$D_2 D_3 D_4$	D_1	a_1
4 times { k=2			
4 times { k=3			

avg(a_1, a_2, a_3, a_4) = $a_{k=1}$

③ k' fold CV → DTrain → NN
↳ k in KNN

→ what is the right k'

k'=4; (k'=10); k'=100

↓

Rule of Thumb : 10 fold CV

We are repeating multiple time, to compute the best k in KNN

Increases k' time when we are doing k' CV

18.15 Visalizing Train, Validation & test datasets

Imagine we have a big dataset D_n → D_{Train} (60%), D_{cv} (20%), D_{Test} (20%)

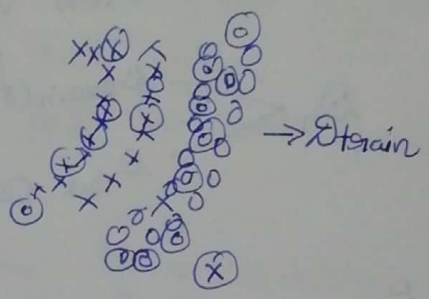
X : -ve class datapoint in D_{Train}

O : +ve class datapoint in D_{Train}

Train, CV, Test : (x_i, y_i) — x_i: Input data point; y_i: class label

(X) - ve point D_{cv}

(O) +ve point D_{cv}



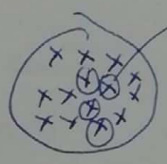
Tips

(1) D_{Train}, D_{cv} donot overlap perfectly

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

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

Intuitively



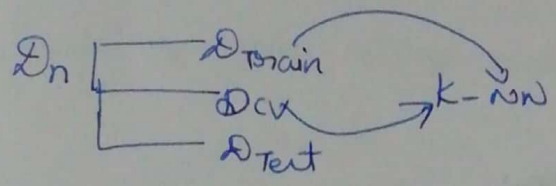
18.16 How to determine underfitting & overfitting

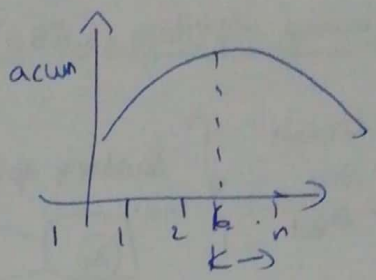
k-fold on D_{cv} → best k → neither overfit / underfit

(Q) How can we be sure about O.F & U.F

$$\text{accuracy} = \frac{\#\text{ correctly class pts}}{\text{Total } \#\text{pts}} = 0.93$$

$$\text{Error} = 1 - \text{accuracy} = 0.07$$





Training Error

what is the train Error for 2-NN?

for each x_i in D_{train}

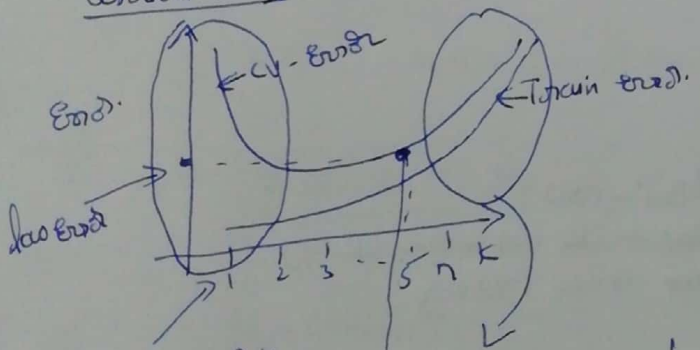
- find 2 nearest neighbour to x_i from D_{Train}
- majority vote to get the class label
- if y_i == class label
  accurate

Else
Error

what is the ~~train~~ D_{CV} Error for 2-NN?

→ Same as above, Calculate the Error & accuracy

Consider K = 5



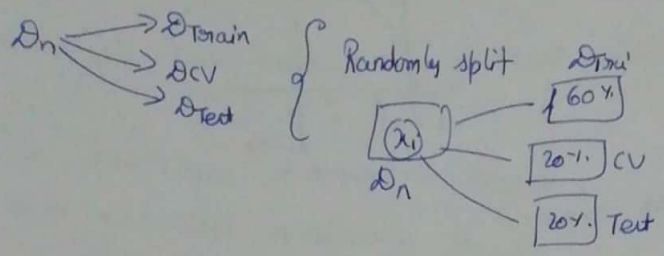
K = 1 overfit
K = n underfit

If K is in this region overfit
Train Error ↓
CV Error ↑

If K is in this region underfit
Train Error ↑
CV - Error ↑

Trade off
{Train Error & CV Error will be close}

18.17 Time based splitting (TBS)

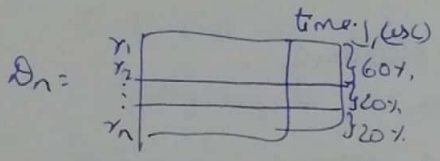


Probability of x_i to be part of
train is 60%.
CV is 20%.
Test is 20%.

For Amazon Food reviews, time based splitting is better than Random splitting

$$D = \begin{bmatrix} & f_1 & f_2 & \cdots & f_d \\ r_1 & & & & \\ r_2 & & & & \\ \vdots & & & & \\ r_n & & & & \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

In the actual dataset

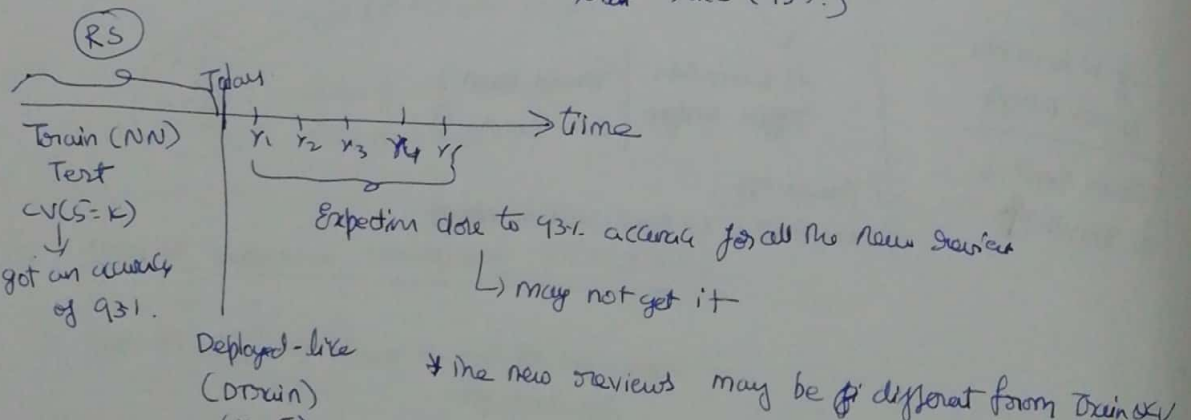


→ Sort the reviews on the time column in the asc order.

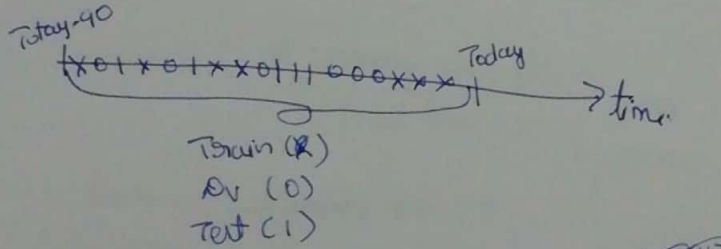
Why TBS?

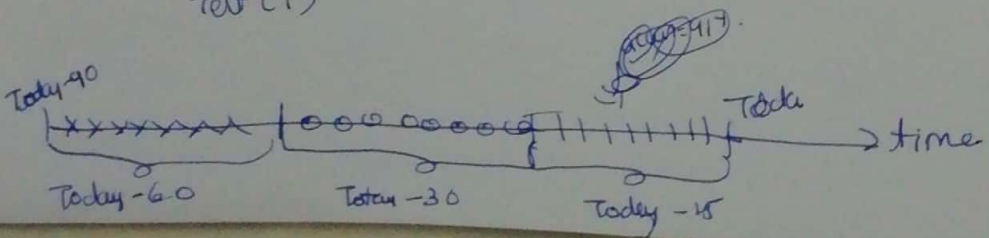
Let's say I used random splitting

D_{Tri} - NN
D_{CV} - K in K-NN (K=5)
D_{Tst} → acc (93%)



* the new reviews may be different from Train & CV
* with time my products & reviews change





If I train my model 15 days back and I tested on last 15 days accuracy is 91%.

We can predict that when we predict the next 15 days data we can expect 91% of accuracy

when ever time is available y things / behaviour or data change over time then time based splitting is preferable than random splitting

18.18 k-nn for regression

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

$x_q \to y_q \to$ class label

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

$x_q \to y_q \to$ number

① Given x_q, find K-nearest neighbours

$(x_1, y_1)\ (x_2, y_2) \ldots (x_k, y_k)$

② $y_q \leftarrow y_1, y_2, y_3 \ldots y_k$

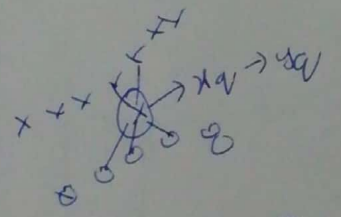
↳ all are real valued numbers.

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

$y_q = \text{median}(y_i)_{i=1}^k$ → less prone to outliers

* For some algorithm we can ~~can~~ extend/modify classification to regression.

18.19 weighted k-nn



5 nn
k=5

2 → ve
3 → +ve
→ y_q +ve

$x_q \to$ $x_1, y_1, d_1 \to 0.1$ −ve
$x_2\ y_2\ d_2 \to 0.2$ −ve
$x_3\ y_3\ d_3 \to 1.0$ +ve
$x_4\ y_4\ d_4 \to 2.0$ +ve
$x_5\ y_5\ d_5 \to 4.0$ +ve

Given my x_q →

	y_i distance	w_i	
x_1, -ve	-0.1	10	15
x_2, -ve	-0.2	5	
x_3, -ve	-1.0	1	1.75
x_4, -ve	-1.0	0.5	
x_5, -ve	-4.0	0.25	

$w_i = 1/d_i$ ← one way to find weights

$d_i \uparrow, w_i \downarrow$

$d_i \downarrow \; w_i \uparrow$

→ $w_i = \frac{1}{d_i}$ Simplest weight function

18.20 Voronai diagram

→ Related to k-nn (k=1)

→ It says

18.21 Binary Search Tree

k-nn + O(n) If d is small → Time Complexity
k is small

O(n) → Space Complexity

Time : O(n) ⟶ o(lg(n)) → Kd-tree

n = 1024 ⟶ lg(n) = 10

Applied in Computer graphics, geometry, Applied maths

Binary search tree (BST)

- Data structure
→ Binary Search

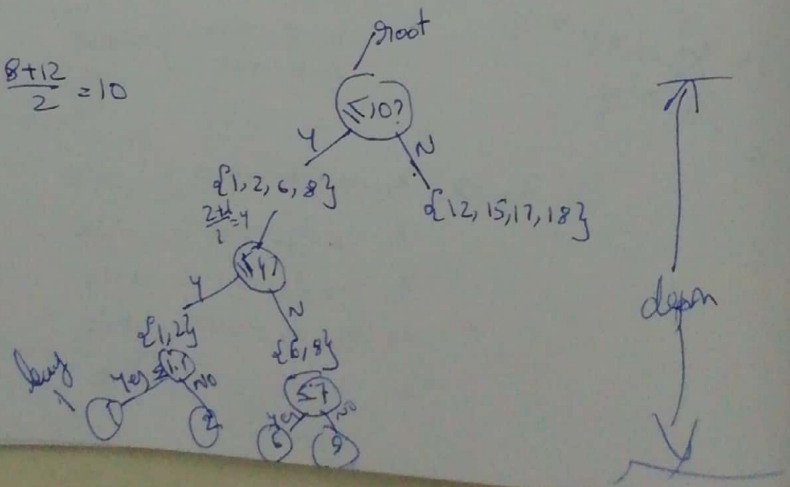
Prob: given a sorted array, find a number is present in array or not

1	2	3	4	5	6	7	8

A: [1, 3, 6, 8, 12, 15, 17, 18]

① Construct a BST

Take median : $\frac{8+12}{2} = 10$



② find an Element in an array

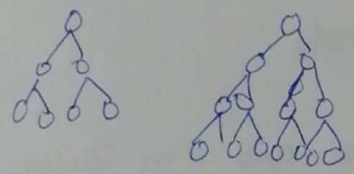
$q = 15$

To find 15, with Comparision we can find 15 exists or not

BST : Time Complex to search is $O(\log(n))$

↳ depth of tree.

2^2 $n=4$ 2^3 $n=8$ 2^4 $n=16$ 2^5 $n=32$



10:29 How to build a Kd-tree

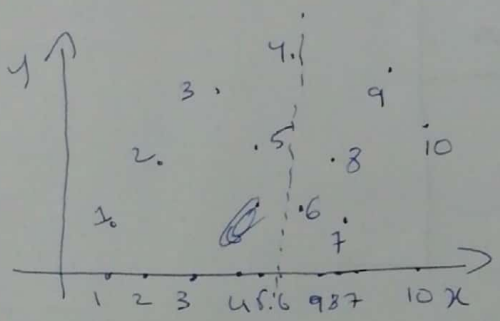
BST : given a list of sorted number → Given a tree.

scalar '1D'

Extend BST to Kd tree

↳ It will operate in 2D, 3D ... n

→ 2D tree

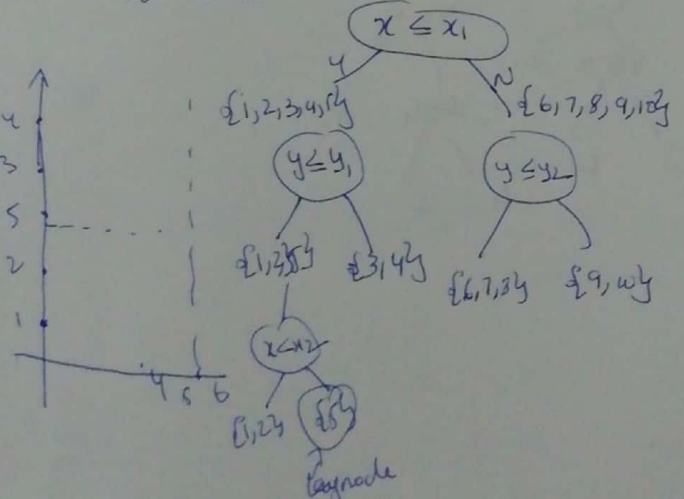


① Pick x-axis, project points on to x-axis.

② Compute the median (will be b/w 5 & 6)

↳ Half the points to the left of median. and the half to the right

③ Alternating b/w axes



18.30 Code Sample : Decision Boundary

Sklearn.neighbors.KNeighborsClassifier

class Sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None n_jobs=1, **kwargs)

n_neighbors : Int optional (default=5)

weights : Str or callable, optional (default='uniform')

- 'uniform': uniform weights. All points in the neighborhood are weighted Equally
- 'distance': weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away
- [callable]: a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights

algorithm: {'auto', 'ball_tree', 'kd_tree', 'brute'}, optional

Algorithm used to compute the nearest neighbors:

* 'ball_tree' will use BallTree
* 'kd_tree' will use KDTree
* 'brute' will use the brute-force search
* auto will attempt to decide the most appropiate algorithm based on the values passed to fit method

Note: fitting on sparse input will override the setting of this parameter, use brute force

* Dimensionality is low it will pick KDTree
* If dataset is small it will pick Brute

leaf_size : int, optional (default=30)

Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree.

The optimal value depends on the nature of the problem

p: integer, optional (default=2)

Power parameter for the Minkowski metric. When p=1, this is equivalent to using manhattan_distance (L1) & euclidean_distance (L2) for p=2. For arbitrary p, minkowski_distance (l_p) is used

metric: string or callable, default 'minkowski'

The distance metric to use for the tree. The default metric is minkowski and with p=2 is equivalent to the standard Euclidean metric.

See the documentation of the distance-metric class for a list of available metrics

metric_params: dict, optional (default= None)

Additional keyword arguments for the metric function

n_jobs: int, optional (default=1)

The number of parallel jobs to run for neighbor search.

If -1, then the number of jobs is set to no. of CPU cores.

Doesn't affect fit method.

Examples In sklearn documentation

```
X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)
print(neigh.predict([[1.1]]))
```

O/P : [0]

```
print(neigh.predict_proba([[0.9]]))
```

[[0.666667, 0.333333]]

18.31 Code Sample: Cross Validation.

```
import sklearn.cross_validation import cross_val_score.
name = ['x', 'y', 'class']  # define class names.
df = pd.read_csv('Concentriccir2.csv', header=None, name=names)
df.head()
# Create design matrix X and target vector y
X = np.array(df.iloc[:, 0:4])  # End index is Exclusive
y = np.array(df['class'])  # showing you two ways of indexing a
                           pandas df
```

Simple Cross Validation

```
# Split the dataset into train & test.
X_1, X_test, y_1, y_test = Cross_Validation.train_test_split
                (X, y, test_size=0.3, random_state=42)
```

Split the train data set into Cross validation train & Cross Validation test

```
X_tr, x_cv, y_tr, y_cv = Cross_validation.train_test_split (X_1, y_1,
                                test_size = 0.2)
for i in range (1, 30, 2):
    knn: kNeighborsClassifier (n_neighbor = i)
    knn.fit (X_tr, y_tr)
    pred = knn.pred (X_cv)  # prediction of cv train
    # Evaluate cv accuracy
    acc = accuracy_score (y_cv, pred, normalize = True) * float (100)
    print ('\n cv accuracy for k = %d is %.d %%' % (i, acc))
knn = kneighborsClassifier (1)
knn.fit (X_tr, y_tr)
pred = knn.predict (X_test)
acc = accuracy_score (y_test, pred, normalize = True) * float (100)
print ('\n Test accuracy
                                                    )
```

O/P

K = 1 100 %.

3 100 %.

5 100 %.

```
# creating odd list of k for knn.
myList = list(range(0,50))
neighbors = list(filter(lambda x: x%2 != 0, myList))

# Empty list that will hold cv scores.
cv_scores = []

for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors = k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())

# changing to misclassification error.
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print('\nThe optimal number of neighbor is %d.' %optimal_k)

plt.plot(neighbors, MSE)

for xy in zip(neighbors, np.round(MSE, 3)):
    plt.annotate('(%s,%s)' %xy, xy=xy, textcoords='data')
plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()
```

KFold

class sklearn.model_selection.KFold (n_splits, shuffle = False, random_state = None)

→ K-Folds cross validator

→ Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default)

→ Each fold is then used once as a validation while k-1 remaining folds form the training set.

n_splits : int, default = 3

Number of folds. Must be at least 2.

Shuffle : boolean, optional

Whether to shuffle data before splitting into batches.

random_state : int optional.

```
from sklearn.model_selection import KFold
X = np.array ([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array ([1, 2, 3, 4])
kf = KFold (n_splits = 2)
kf.get_n_splits (X)
// 2
print (kf)
// KFold (n_splits = 2, random_state = None, shuffle = False)
for train_index, test_index in kf.split (X):
    X_train, X_test = X [train_index], X [test_index]
    y_train, y_test = y [ ... ], y [ ... ]
```

Stratified K Fold

StratifiedKFold (n_splits=3, shuffle=False, random_state=None)

→ Stratified K-Folds Cross Validation.

→ provides train/test indices to split data in train/test sets

→ This Cross Validation object is a Variation of KFold that returns stratified folds.

→ The folds are made by preserving the percentage of samples for each class.

Notes

All the folds have size trunc(n_samples/n_splits), the last one has the complementary.

```
from sklearn.model_selection import StratifiedKFold
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4], [5, 6], [7, 8]])
y = np.array([0, 0, 1, 1])
Skf = StratifiedKFold(n_splits=2)
Skf.get_n_splits(X, y)

for train_index, test_index in Skf.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[ ,, ], y[ ,, ]
```

Repeated KFold

RepeatedKFold (n_splits=5, n_repeats=10, random_state=None)

→ Repeats K-Fold n times with different randomization in each repetition

→

```
rkf = RepeatedKFold(n_splits=2, n_repeats=2, random_state=2652124)
for train_index, test_index in rkf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[ , ], y[ , ]
```

Leave One Out Cross Validation (LOOCV)

⇒ In this approach, we reserve only one data point from the available dataset and train the model on the rest of the data

→ This process iterates for each data point.

→ This also has its own advantages & disadvantages.

* we make use of all data points, hence bias will be low
* we repeat the Cross Validation process n times (where n is the number of data points) which results in high execution time.
* This approach leads to higher variation in testing model effectiveness because we test against one data point. So, our estimations gets highly influenced by the data point. If the data point turns out to be an outlier it can lead to a higher variation.

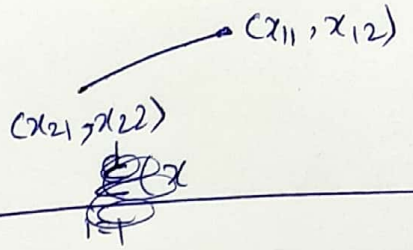
```
from sklearn.model_selection import LeaveOneOut
X = np.array([[1,2], [3,4]])
y = np.array([1,2])
loo = LeaveOneOut()
loo.get_n_splits(X)
for train_index, test_index in loo.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[ " ], y[ " ]
```

$$\text{Cosdistance}(x_1, x_2) = 1 - \text{CosSim}(x_1, x_2)$$

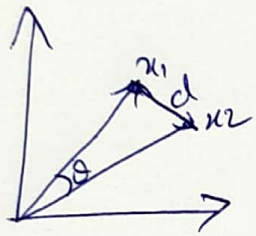
$$\text{CosSim}(x_1, x_2) = \cos(\theta_{x_1 x_2})$$

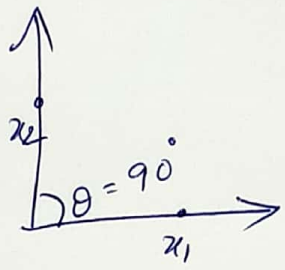
Euclidean distance

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



Example

① 

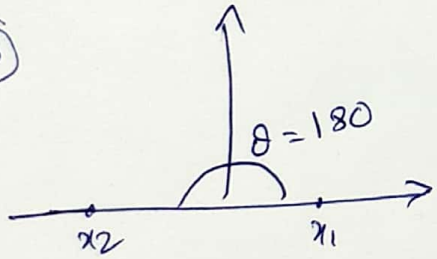
② 

$\cos(90) = 0$

$\text{CosSim}(x_1, x_2) = 0$

$\text{Cosdist}(x_1, x_2) = 1 - 0$

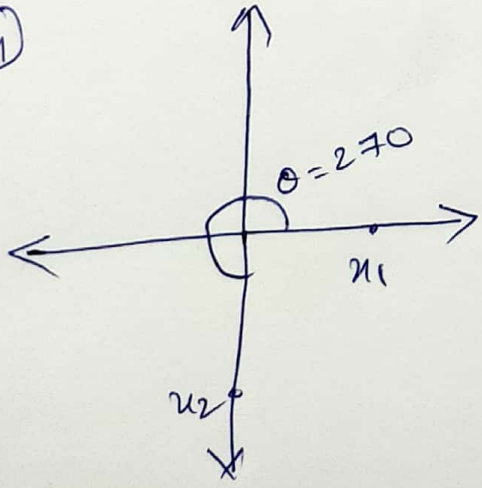
$= 1$

③ 

$\cos(180) = -1$

$\text{CosSim}(x_1, x_2) = -1$

$\text{Cosdist}(x_1, x_2) = 1 - (-1) = 1 + 1 = 2$

④ 

$\cos(270) = 0$

$\text{CosSim}(x_1, x_2) = 0$

$\text{Cosdistance}(x_1 x_2) = 1 - 0 = 1$

⑤ 

$\cos(0) = 1$

$\text{CosSim}(x_1, x_2) = 1$

$\text{Cosdist}(x_1, x_2) = 1 - 1 = 0$

Manhattan distance

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