MLE can badly overfit in high dimension.

Dirichlet distribution:

$$P(\theta|\alpha) = \frac{\Gamma\left(\sum_{i=1}^{k}\alpha_i\right)}{\prod_{i=1}^{k}\Gamma(\alpha_i)}\theta_1^{\alpha_1-1}\cdots\theta_k^{\alpha_k-1}$$

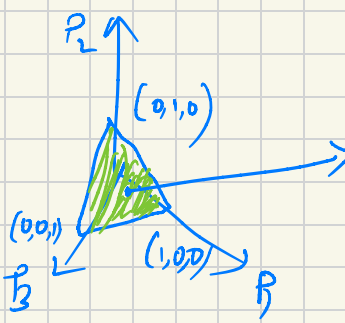
Sample from probability distribution.

$$P_1, P_2 \cdots P_K \sim Dir(\alpha_1, \alpha_2 \cdots \alpha_K)$$

$0 \le P_i \le 1$

$\sum_i P_i = 1$

Support of this distribution



this region follows the above properties. this region is called Simplex (It's a 2D simplex)

$p_2 \quad \frac{N_c}{N}$

Dim $\Rightarrow$ C

$$\mu_1 = \frac{1}{N_c} \sum_{y_i \in c} x_i$$

Dim $\Rightarrow$ CXD

$$\Sigma_c = \sum_{i: y_i = c} (x_i - \mu_c)(x_i - \mu_c)^T$$

Dim = CXDXD

$O(CD^2)$

to we need to estimate these many parameter. & its very high & if dim is large then no of parameters to estimate is large. so high chance is it will overfit if data is less.

Can we reduce the number of parameters?

YES, using naive Bayes rule.

It assumes that x_i & x_j are conditionally independent given y, for all $i \neq j$.

$$P(x_1, x_2 \cdots x_n | y) = \prod_i P(x_i | y)$$

Reducing the number of parameters to estimate

$$P(y|x_1, x_2 \cdots x_n) = \frac{P(x_1, x_2 \cdots x_n|y) P(y)}{P(x_1, x_2 \cdots x_n)}$$

$$P(x_1, x_2 \cdots x_n|y) = P(x_1|x_2, x_3 \cdots x_n, y) * P(y|x_2, x_3 \cdots x_n)$$

$$= P(x_1|x_2 \cdots x_n, y) * P(x_2|x_3 \cdots x_n, y) * P(y, x_3, x_4 \cdots x_n)$$

$$= P(x_1|x_2 \cdots x_n, y) * P(x_2|x_3 \cdots x_n, y) * \cdots P(x_n|y) * P(y)$$

Now the "naive" conditional independence assumptions comes into play: assume that each feature x_i is conditionally independent of every other feature x_j for $i \neq j$, given y.

so,

$$P(y|x_1, x_2 \cdots x_n) \propto P(y) * P(x_1|y) * P(x_2|y) \cdots P(x_n|y)$$

$$\propto P(y) * \prod_{i=1}^{n} P(x_i|y)$$

To estimate the parameter it requires 2D parameters.

$$\hat{y} = \underset{y_k}{\operatorname{argmax}} \; P(y = y_k) \cdot \prod_i P(x_i|y = y_k)$$

$$\hat{y} = \underset{y_k}{\operatorname{argmax}} \; \pi_k \prod_i \theta_{ijk}$$

MLE

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, P(D \mid \theta)$$

$$\hat{\pi}_k = \hat{P}(y = y_k) = \frac{\#D\{y = y_k\}}{|D|}$$

$$\hat{\theta}_{ijk} = \hat{P}(x_i = x_j \mid y = y_k) = \frac{\#D\{(x_i = x_j) \wedge (y = y_k)\}}{\#D\{y = y_k\}}$$

MAP (Beta → for binary classification, Dirichlet prior → when you have K-classes)

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, P(\theta \mid D) = \underset{\theta}{\operatorname{argmax}}\, \frac{P(D \mid \theta)\, P(\theta)}{P(D)}$$

$$\hat{\pi}_k = \hat{P}(y = y_k) = \frac{\#D\{y = y_k\} + (\beta_k - 1)}{|D| + \sum_m (\beta_m - 1)}$$

$$\theta_{ijk} = \hat{P}(x_i = x_j \mid y = y_k) = \frac{\#D\{x_i = x_j \wedge y = y_k\} + (\beta_k - 1)}{\#D\{y = y_k\} + \sum_m (\beta_m - 1)}$$

NBC Algorithm

$N_c = 0$, $N_{jc} = 0$;
for $i = 1 : N$ do
  $c = y_i$ // class l
  $N_c := N_c + 1$
  for $j = 1 : D$ do
    if $x_{ij} = 1$ then
      $N_{jc} := N_{jc} + 1$

$\hat{\pi}_c = \frac{N_c}{N}$

$\hat{\theta}_{jc} = \frac{N_{jc}}{N}$

Complexity:
$O(ND)$ time.

Naïve Bayes Classifier

- Features are conditionally independent given the class label.

$$p(\mathbf{x}|y=c,\boldsymbol{\theta}) = \prod_{j=1}^{D} p(x_j|y=c,\boldsymbol{\theta}_{jc})$$

v.v.f

- class-conditional density p(x|y)
  - In the case of real-valued features, we can use the Gaussian distribution: $p(\mathbf{x}|y=c,\boldsymbol{\theta}) = \prod_{j=1}^{D} \mathcal{N}(x_j|\mu_{jc},\sigma_{jc}^2)$, where μ_{jc} is the mean of feature j in objects of class c, and σ_{jc}^2 is its variance.
  - In the case of binary features, $x_j \in \{0,1\}$, we can use the Bernoulli distribution: $p(\mathbf{x}|y=c,\boldsymbol{\theta}) = \prod_{j=1}^{D} \text{Ber}(x_j|\mu_{jc})$, where μ_{jc} is the probability that feature j occurs in class c.
  - In the case of categorical features, $x_j \in \{1,\ldots,K\}$, we can model use the multinoulli distribution: $p(\mathbf{x}|y=c,\boldsymbol{\theta}) = \prod_{j=1}^{D} \text{Cat}(x_j|\boldsymbol{\mu}_{jc})$, where $\boldsymbol{\mu}_{jc}$ is a histogram over the K possible values for x_j in class c.

we know $\hat{p} = \frac{N_c}{N} \sim Beta(\alpha, \beta)$

$$P(\hat{p}|\gamma) = Beta(N_c + \alpha, N - N_c + \beta)$$

mode of Beta distribution.

$$\text{mode} = \frac{N_c + \alpha - 1}{N + \alpha + \beta - 2}$$

when $\alpha = 1$
$\beta = 1$
It's same as MLE which is uniform distribution

$$\text{mode} = \frac{N_c}{N} = \text{MLE}$$

when $\alpha = 2$ $\beta = 2$

$$\text{mode} = \frac{N_c + 1}{N + 2}$$

same as MAP.

So, its like Laplace smoothing.

Input:

Training dataset T,

F= (f_1, f_2, f_3,.., f_n) // value of the predictor variable in testing dataset.

Output:

A class of testing dataset.

Step:

1. Read the training dataset T;
2. Calculate the mean and standard deviation of the predictor variables in each class;
3. Repeat

   Calculate the probability of f_i using the gauss density equation in each class;

   Until the probability of all predictor variables (f_1, f_2, f_3,.., f_n) has been calculated.
4. Calculate the likelihood for each class;
5. Get the greatest likelihood;

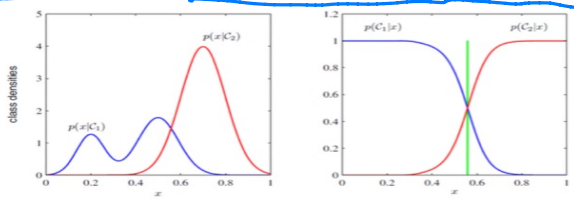
** How Generative model is useful for detecting outlier

Discriminative vs Generative

- **Generative Models :** model the joint distribution p(x, Ck) directly and then normalize to obtain the posterior probabilities.

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}$$

- Most demanding because it involves finding the joint distribution over both x and Ck. For many applications, x will have high dimensionality, and consequently we may need a large training set. if we only wish to make classification decisions, then it can be wasteful of computational resources but can be useful for detecting **outliers or novel classes**.



2

Let x be your data and y be some sort of label or decision variable that this data can be mapped to. Discriminative models learn (some approximation of) $p(y|x)$. Generative models, depending on who you ask, learn $p(x)$, $p(x|y)$ or $p(x, y)$. These three are of course related. We can get the joint from the conditional as $p(x, y) = p(x|y)p(y)$, and we can get the marginal from the joint as $p(x) = \int p(x, y)dy$. In any case, the point is that they learn something about the probability of the data.

Now let's think about what an outlier is. An outlier is an observation in your data that is unlikely. That is, $p(x)$ is small (or $p(x|y)$ is small, if you are able to condition on an observed value of y). So you need some way of calculating $p(x)$ or $p(x|y)$. A generative model allows you to do that, through the math I described above. In words, a generative model was trained to accurately gauge the probability of your training data (assigning higher probabilities to more common values), and so given a new piece of data a generative model can tell you how probable that value is (under the model).

For a discriminative model, this is much harder, as we have just learned $p(y|x)$. How are you going to get from that to $p(x)$ or $p(x|y)$? To get the latter, you need:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

(This is basically Bayes' rule in reverse.) Note that this depends on $p(x)$, which you don't know. Rearranging the same formula, we find:

$$p(x) = \frac{p(y|x)}{p(x|y)p(y)} = \frac{p(y|x)}{p(x, y)}$$

Which again isn't helpful as it requires us to fill in terms on the r.h.s. that we don't know because they're not part of the model. In other words, because you never trained your discriminative model to tell you the probabilities of your data (only the probabilities of the labels *given* the data), you can't get it to do so at test time either.

For a generative model, you plug in either x or the combination of x and y, and the resulting probability, if it's small, will tell you that that particular observation is an outlier.

Share Cite Improve this answer Follow

answered Jan 17, 2022 at 15:02

☐ Parametric Supervised Learning:

A parametric algorithm has a fixed number of parameters.

A parametric algorithm is computationally faster, but makes stronger assumptions about the data. the algo may work well if the assumptions turn out to be correct, but it may perform badly if the assumptions are wrong.

A learning model that summarises data with a set of parameters of fixed size (predefined mapped function) (independent of the numbers of training examples).

No matter how much data you throw at a parametric model, it won't change its mind about how many parameters it needs.

Ex of parametric algorithm is Linear Regression, Linear SVM, Perceptron, Logistic Regression.

Pro - need few data points to learn parameters.

Cons - strong distributional assumptions, not satisfied in practice.

Parametric models:

It assume some finite set of parameters w. given the parameters, future predictions, x, are independent of the observed data, D. $P(x|w,D) = P(x|w)$

So, w capture everything there is to know about the data.

So, the complexity of the model is bounded even if the amount of data is unbounded. This makes them not very flexible.

Non parametric models:

It assumes that the data distribution can't be defined in terms of such a finite set of parameters. But they can often be defined by assuming an infinite dimensional w. usually we think of w as a function.

The amount of information that w can capture about the data D can grow as the amount of data grows. This makes them more flexible.

Ex of non parametric algorithm is KNN, DT, ANN, SVM with Gaussian kernels.

A model is called parameteric if it has finite and fixed number of parameters. Otherwise, it is called as non-parameteric.

For example, let us say that $x_1, x_2, \ldots, x_n$ are real valued samples from some hidden distribution. Now, I can model densities in two different ways:

1. **Parameteric Model:** I can choose a parameteric distribution to model the density, for example, Gaussian. I can compute mean as $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and variance as $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2$. Thus, using these parameters we can model the density as a Gaussian distribution i.e. $f(x) \propto \exp\left(-\frac{(x-\hat{\mu})^2}{2\hat{\sigma}^2}\right)$. Note that, no matter how many data points are given, we just have to estimate two parameters mean and variance for this model. Thus, the number of parameters for this model are finite and fixed and independent of the number of samples available to us.
2. **Non Parameteric Model:** We can also choose a non-parameteric model like Kernel Density Estimator.

   Kernel Density Estimator defines density as: $f(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)$ where h is the bandwidth parameter. Note that, the samples are directly part of the model which means they are the parameters of the model (different samples will give rise to different models). The number of parameters of this model grow linearly with the number of samples and theoretically, they could be infinite in number.

Non Parametric Density Estimation Methods in Machine Learning

Read | Discuss

Non-parametric methods: Similar inputs have similar outputs. These are also called instance-based or memory-based learning algorithms. There are 4 Non – parametric density estimation methods:

1. Histogram Estimator
2. Naive Estimator
3. Kernel Density Estimator (KDE)
4. KNN estimator (K – Nearest Neighbor Estimator)

Histogram Estimator

It is the oldest and the most popular method used to estimate the density, where the input space is divided into equal-sized intervals called **bins**. Given the training set $X = \{x^t\}_{t=1}^N$ an origin x_0 and the bin width h, the histogram density estimator function is:

$$\hat{p}(x) = \frac{\#\{x^t \text{ in the same bin as } x\}}{Nh}$$

Histogram estimator

The density of a sample is dependent on the number of training samples present in that bin. In constructing the histogram of densities we choose the origin and the bin width, the position of origin affects the estimation near the boundaries.

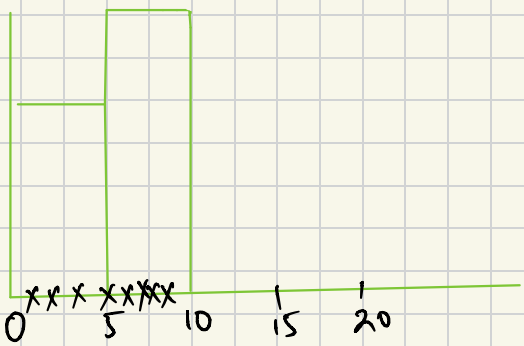
Ex of Histogram Estimator

Let's say values are

[2, 3, 5, 7, 9, 4, 6, 8]

origin = 0

bin-width = 5



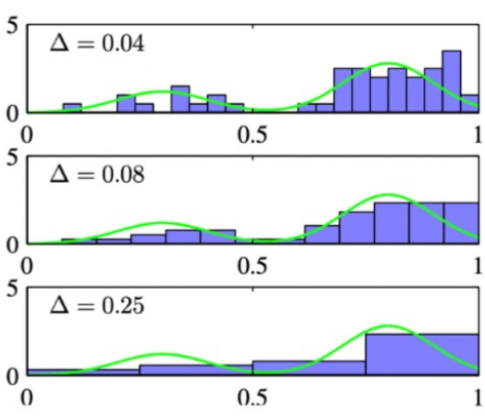
$p(x_i) = \frac{\#\text{ of observations belong to same bin } i}{N \Delta i}$

Δi = bin-width

if Δ too small density model is too spiky.

if Δ too large density model is too smooth.

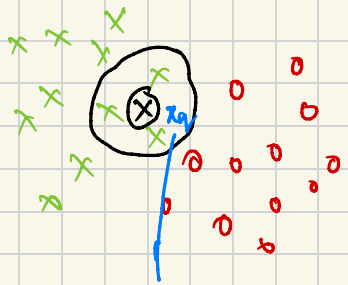
** model complexity is decided by its bin size.



[] KNN [Classification and Regression] :

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

D =



Steps:

1. find K-nearest points to x_q in D. Let K = 3 here.

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

$\downarrow \quad \downarrow \quad \downarrow$

$y_1 \quad y_2 \quad y_3$

2. $\{y_1, y_2, y_3\} \rightarrow$ majority vote.

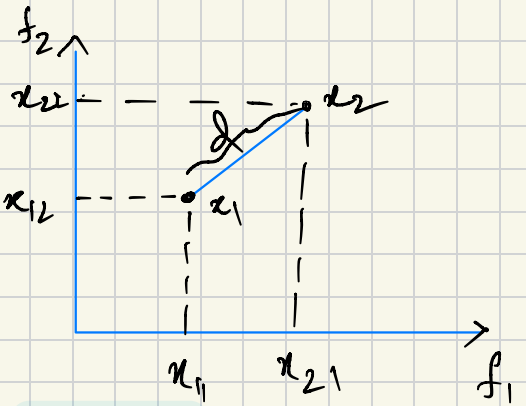
$\{+, +, +\}$ y_q = '+'

$\{+, +, -\}$ y_q = '+'

K should be odd no so that no drawn happens.

Distance measures:

① Euclidean Distance:



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

$$= \|x_1 - x_2\| \quad [\text{length of } x_1, x_2]$$

$x_1 \in R^d$

$x_2 \in R^d$

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

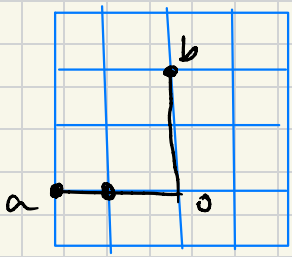
↓ L2 norm of a vector.

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

② Manhatten Distance :

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

(absolute value)



$||x_1 - x_2||_1$ L1 norm of a vector.

dist(a,b)
= dist(a,o) + dist(o,b).

Generalization to L1 & L2 norm —

③ Lp norm / Minkowski dist :

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

if p = 2 then Minkowski dist = Euclidean dist

if p = 1 then Minkowski dist = Manhatten dist.

④ Hamming Distance:

XOR between two boolean vectors.

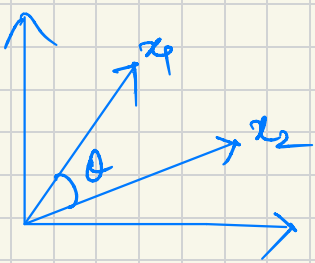
applicable to string also,

x_1 = a b c a d e f h i k

x_2 = d c b a d e g h i k

hamming dist $(x_1, x_2) = 4$.

⑤ Cosine-similarity:-



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

if x_1, x_2 is unit vector

then $\cos(\theta) = x_1 \cdot x_2$

⑥ Edit Distance:-

- The minimum edit distance between two strings.
- 3 operations allowed (Insert, Delete, substitute)
- Needed to transform one into the other.

Ex:

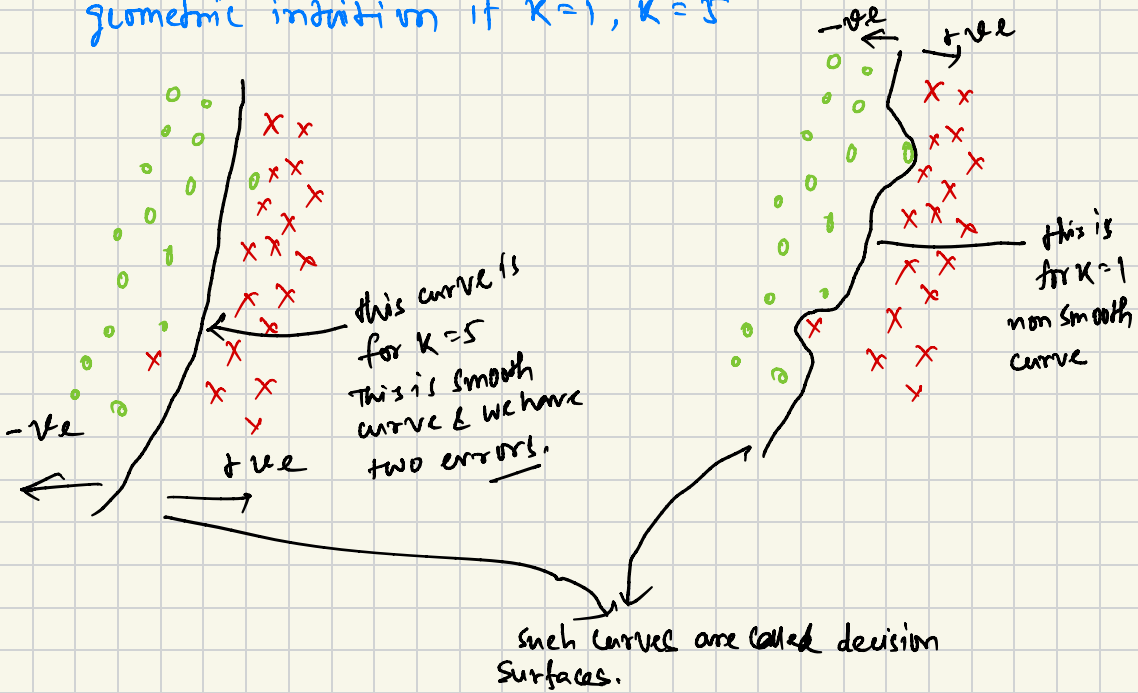
```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N

d s s   i s
```

So, Distance between these two is 5.

how to decide K ?

K in KNN is hyperparameter.

geometric intuition if K=1, K=5



if K is too large, let K=n

then if $n_1 > n_2$ [$n_1 + n_2 = n$]

In this case, any query point you take, it will be true. because you have to consider all n points & among n_1 & n_2, $n_1 > n_2$. so new point will be assigned to n_1 side.

Overfit
- It tries to cover noisy points also
- non-smooth
- no mistakes

K=1

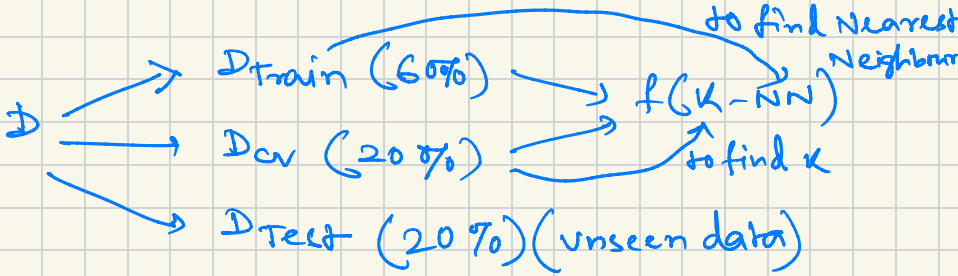
underfit
- Imperfect
- It cares about majority points. every points belongs to majority class.
- Imperfect

K=n

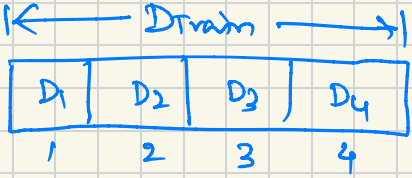
wellfit
- less prone to noise
- robust
- K=5 smooth.

** How to determine K ?

(1) We will devide the whole dataset into 3 parts.



(2)

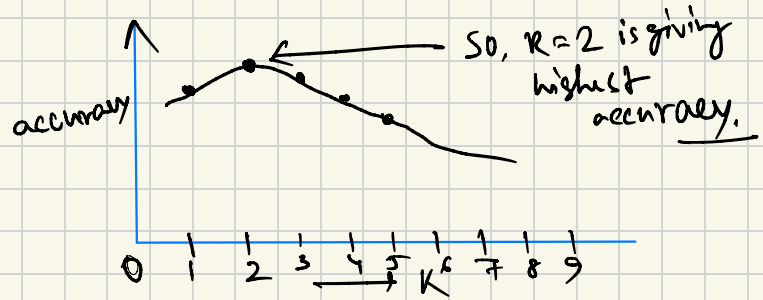


→ randomly divide into 4 parts with equal size. (4-Fold)

(3)

	D_{Train}	D_{CV}	Accuracy	
K=1	$D_1 D_2 D_3$	D_4	a_1	avg(a_1, a_2, a_3, a_4)
K=1	$D_1 D_2 D_4$	D_3	a_2	
K=1	$D_1 D_3 D_4$	D_2	a_3	
K=1	$D_2 D_3 D_4$	D_1	a_4	
K=2			a_1	avg(a_1, a_2, a_3, a_4)
K=2			a_2	
K=2			a_3	
K=2			a_4	

Considering 4-fold cross-validation we need to plot average accuracy.



Test/Evaluation time & space complexity:

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

output: y_q

K is generally small, its 5 or 10.

KNNPts = [ ]

for each x_i in D_{Train}: → n points, d-dim each, so, time complexity = O(nd)

- Compute $d(x_i, x_q) \rightarrow d_i$
- Keep smallest K distance (x_i, y_i, d_i) → considering K is very small it will be O(1)

O(nd)

Count_pts = 0 ; Count_neg = 0

for each x_i in KNNpts:

- if y_i is true:
  - Count_pts + 1
- else Count_neg + 1

O(1)

- if Count_pts > Count_neg:
  - return y_q = 1 (+ve)
- else:
  - return y_q = 0 (-ve)

O(1)

Time Complexity = O(nd)

Space Complexity = O(nd) to keep training dataset.

** Limitations of KNN:

It takes huge time & Space. to reduce it we have some techniques

→ KD-tree

→ LSH (Locality sensitive Hashing)

weighted KNN:

Instead of majority votes, sometimes we can go with shortest distance.

let $K = 5$

for given x_q

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

these points are closest to x_q, but if we consider majority votes, it will be assigned to +ve. Do we need to consider distance also? why only majority votes? to eliminate this we can go with weighted KNN.

one way to assign weights is

$w_i = \frac{1}{d_i}$

$d_i \uparrow \; w_i \downarrow$

$d_i \downarrow \; w_i \uparrow$

Sum = 15 $(1 0.5)$

Sum = 1.75 $(1 + \frac{1}{2} + \frac{1}{4})$

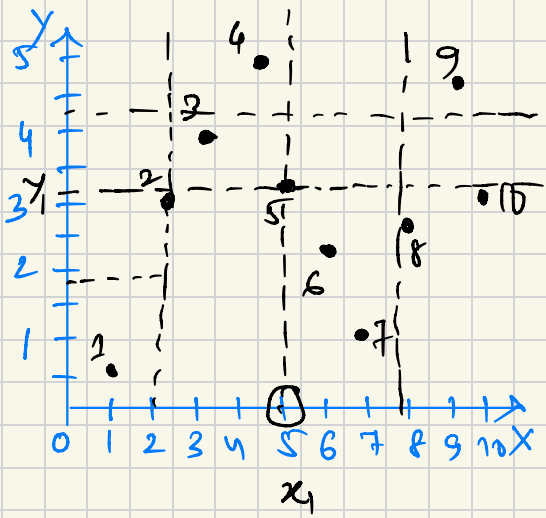
Here, $15 >> 1.75$ so, it will be assigned to −ve class.

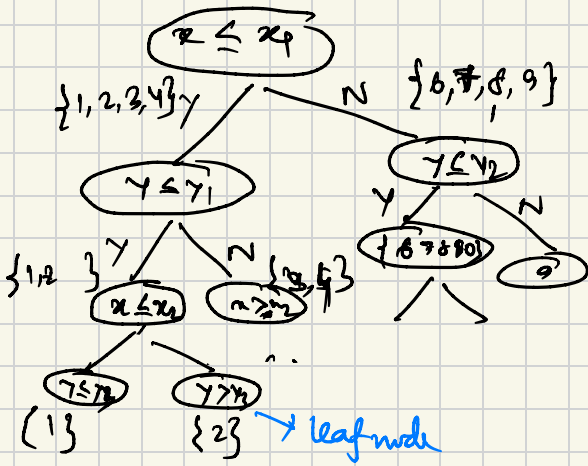
How to implement kd-tree using BST?

BST is → 1D [scalar]

↳ Extending BST to K-dimensional tree is called kd tree.

step 1: pick x-axis, project points into x-axis, compute median, split data into median.





You alternate each axis one after another till leaf nodes.

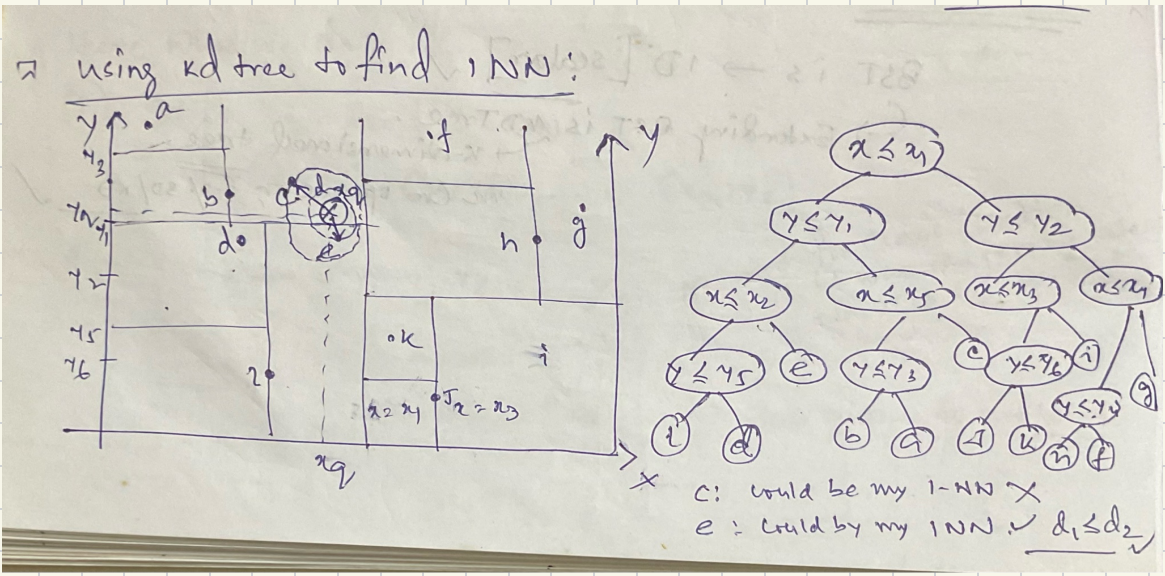
So, kd-tree breaks the space using axis parallel lines/planes into rectangle (2D) / cuboids (3D) / hypercuboid (nD).

In 2D → axis parallel lines → rectangles

In 3D → axis parallel planes → cuboids.

In nD → axis parallel hyperplane → hypercuboid.

using KD-tree to find 1NN:



# of comparison to find 1-NN

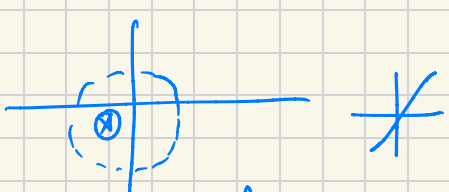
best case: $O(\log n)$

worst case: $O(n)$

if it is K-NN, the best case $O(K * \log n)$

worst case $O(K * n)$.

Limitations of KD-tree:



When d is not small the time complexity $O(2^d \log n)$

for 2D if need 4 regions to cheer