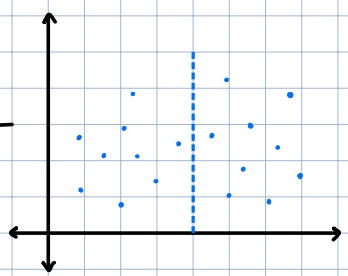
Ball Trees / Decision Trees

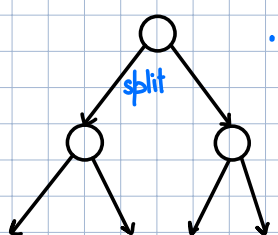
Taken from: https://www.youtube.com/watch?v=E1_WCdUAtyE&t=2140s

Imagine a data set like



Split the data in ½ in some dimension. (good choice - take the dim that has max spread)

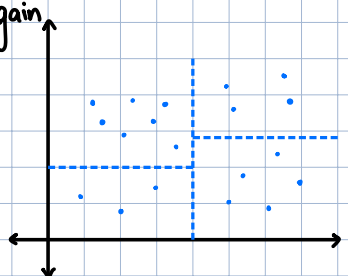
Idea is to keep doing this → we are building an inverted tree.



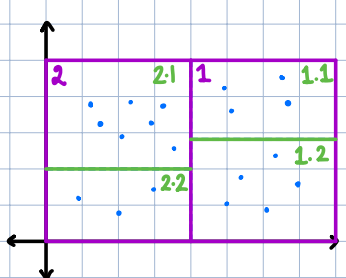
- Initially → one node for entire data set

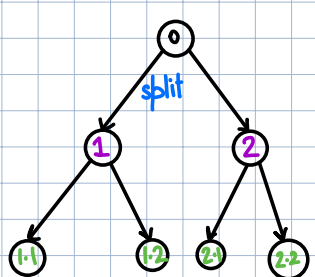
Keep calling the fn. again. We stop when only m ($m \in \mathbb{N}$) are left.

split the data set again

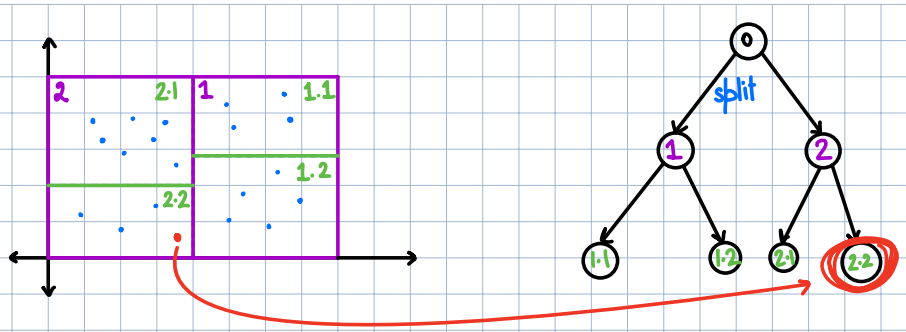


Now





Eg. Mark everything on the right of the data set to be bigger than the median of the dataset, and smaller on the left.

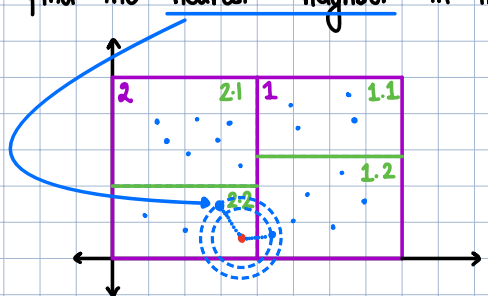


Let • in 2.2 be a test point.

Assume we want to find the nearest neighbor. First thing is that we will descend the inverted tree :-

- Look at the top node. → which is the first half split.
- Is the data point left / right ? Left
- Now, the second split look at inner split of left of first split.
- Are we above or below the split (or threshold) ? → below
- We end up in box 2.2

Now, we find the nearest neighbor in that set.



We store that distance and draw a circle around it. what about others?
Go one split up.
Does the circle intersect the last split ? No → Rule out that split.
Go one more up.
Does the circle intersect any other split ? YES → Cannot Rule it out
Go in it.
Does the circle intersect 1.1 split ? No → Rule it out
Does the circle intersect 1.2 split ? YES →

We do nearest neighbor search, and we find another point from 1.2 to be closer. Note the distance and draw a ball around the test point.

This method is called KD trees.

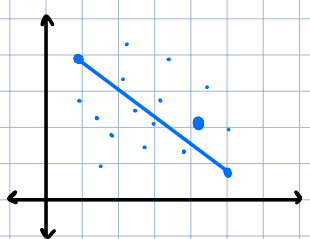
KD trees are effective whenever the dimension of the data set is not too much. (or low).

As KD trees are axis aligned and cannot take a different shape. So the distribution might not be correctly mapped, leading to poor performance.

That is when Ball trees come in. Balls trees Algorithm might be the best solution. It's performance depends upon training data, the dimensionality and the structure of the data.

If the data set is high dimensional, or intrinsically small that is embedded in a higher dim, Ball trees are preffered.

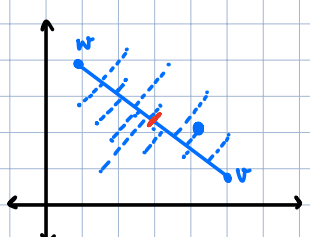
Ball Trees



This is a vectorised operation; very fast

- Pick a random point.
- Find a pt that is furthest away.
- Find a pt that is furthest away from above (second) point.
- Connect those two.

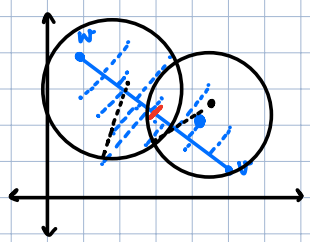
Now we have a direction and we want to project everything onto that direction.

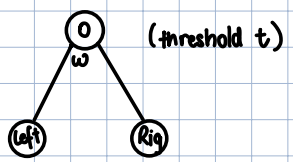


w is some vector. Similarly u.
So we can just $w^T x$ of every single pt and we will find the projections on the direction.

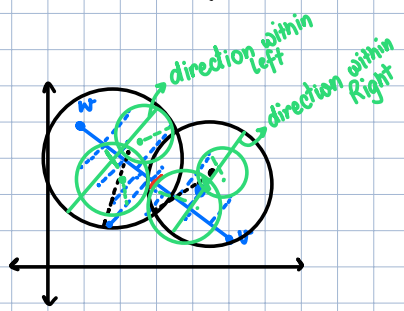
And then, we can split the direction by the median.

Now calculate the mean point of the data pts on the left and on the right.



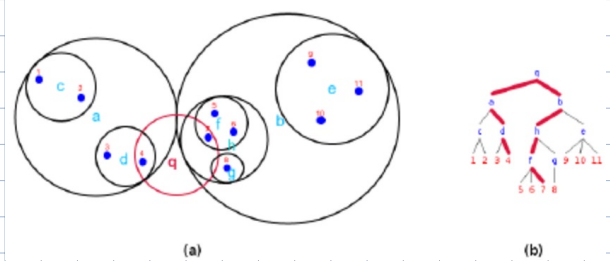


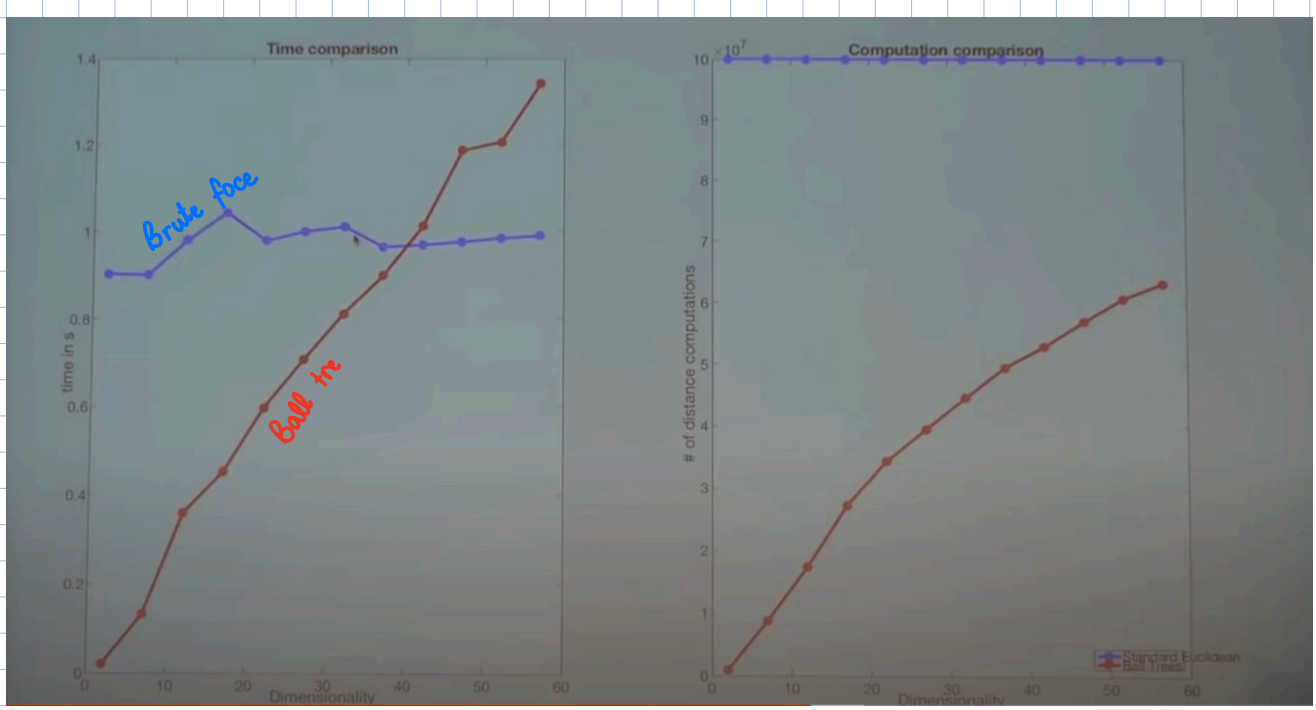
Then calculate the furthest pt from the mean on the left and draw a circle of that radius. Do same on right.



We will get a tree of these balls. We will do the same process to eliminate the balls (instead of blocks) for a test point.

Balls can intersect. If there is a data pt in the intersected area, then, that pt goes to an arbitrary node.





The reason for the difference :

1.) Cache Performance
2.) Parallelism

Decision Tree Algorithm

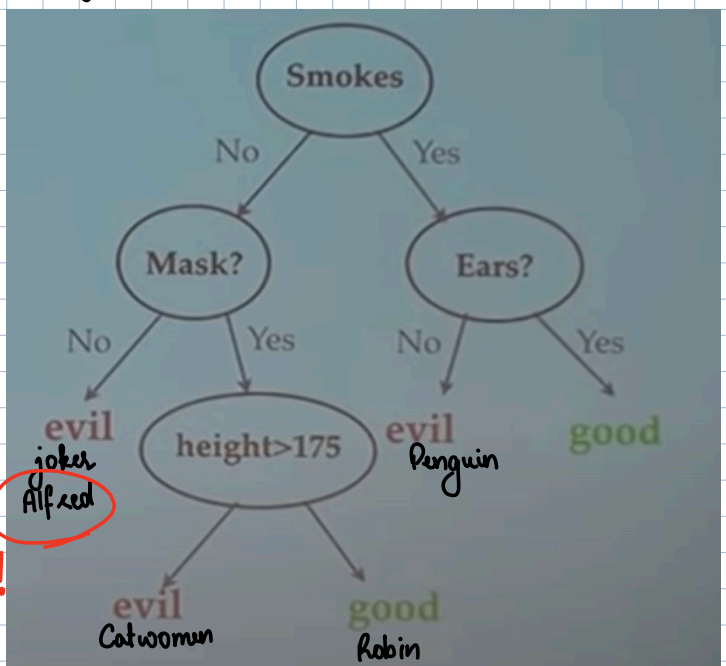
Let's build a decision tree on the following data to check if our date is good / evil ?

Problem: Is your date good or evil?

	mask	cape	tie	ears	smokes	height	class
Batman	y	y	n	y	n	180	good
Robin	y	y	n	n	n	176	good
Alfred	n	n	y	n	n	185	good
Penguin	n	n	y	n	y	140	evil
Catwoman	y	n	n	y	n	170	evil
Joker	n	n	n	n	n	179	evil
Batgirl	y	y	n	y	n	165	?
Riddler	y	n	n	n	n	182	?
Your date	n	y	y	y	y	181	?

Is the following a good choice for decision tree algorithm?

Using the below tree, we get



How to correct it? Make use of tie column.

We can repeat the decision tree enough times, then we are guranteed to have a perfect decision tree algorithm.

But this is not an optimal thing to do. Why? Overfitting!

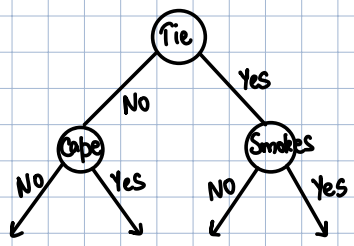
Small tree → Higher error on the training set (High Bias Secnario)

Too deep tree → Appropriate error on training set, but high error on test set

How to find a decision tree that is most correct?

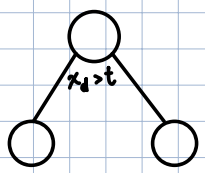
This is a NP hard problem.

Best tree for this problem ?



Consider a Dataset → having n data points.

To build decision trees, we want to create a data structure like



In K-D trees, we split the data set in half, to partition the data to get balanced tree. With decision trees, the object is different. We want to have leaves that are pure, such that at every single point, a leaf has, has the same label. The reason being, if a test point goes in there, there's a very high probability it has a predicted label.

We need impurity fns.

A function that measures how pure a set is in terms of for example label.

We want high puriety → every data pt has the same label.

1.) Gini Impurity :

Let $S = \{(x_1, y_1), \ldots\ldots, (x_n, y_n)\}$, a set in a certain leaf.

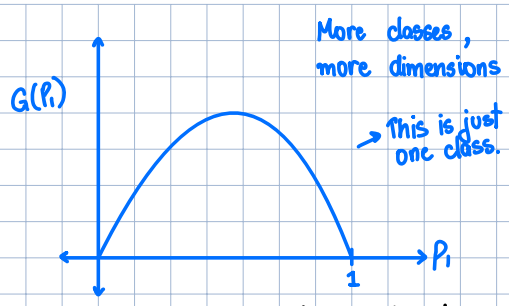
Let K be the number of labels.

Prob of getting a certain label = $\frac{|S_k|}{|S|}$ = $\frac{\#\text{ of points with that label}}{\#\text{ total points in the set}}$

where $S_K = \{(x,y) \in S \mid y = K\}$

So, $S = S_1 \cup S_2 \cup \ldots\ldots \cup S_K$

Impurity, $G(S) = \sum_{k=1}^{K} p_k (1 - p_k)$



We want the impurity to be low.

2.) Entropy (Information):

We don't want → Worst case scenario, we get a tree who's all k classes have same prob., s.t $q_1 = q_2 = \ldots\ldots = q_K = \frac{1}{K}$

K-L Divergence: Tells us how different two distributions are.

It is defined as: $KL(p \| q) = \sum_{k=1}^{K} p_k \log \frac{p_k}{q_k}$

Simple case: when $p_K = q_K$, then $\log(1) = 0$, thus $KL(p\|q) = 0$
Hence, all leaves are the same.

We want $KL(p \| q) = \sum_{k=1}^{K} p_k \log \frac{p_k}{q_k}$ to be as high as possible.

$$KL(p \| 1/K) = \sum_{k=1}^{K} p_k \log \frac{p_k}{1/K}$$

$$= \sum_{k=1}^{K} p_k \log p_k - \underbrace{\sum_{k=1}^{K} p_k}_{=1} \underbrace{\log(K)}_{\text{constant}}$$

$$= \sum_{k=1}^{K} p_k \log p_k - \log(K)$$

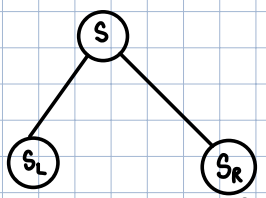
Since we want to maximize the above fn, which is equivalent to maximise (this)

$$= \max_{p} \sum_{k=1}^{K} p_k \log p_k$$

$$= \min_{p} - \sum_{k=1}^{K} p_k \log p_k$$

$$= \min_{p} H(s) \rightarrow \text{Entropy over a leaf}$$

Entropy over a tree :-

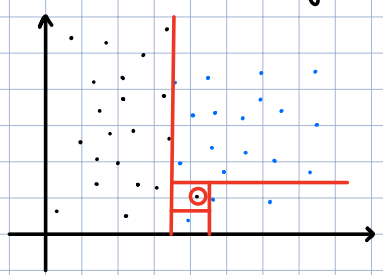


$$P_L = \frac{\#\text{ in } S_L}{\#\text{ total}} = \frac{|S_L|}{|S|} \qquad P_R = \frac{|S_R|}{|S|}$$

$$H(S) = P_L\, H(S) + P_R\, H(S)$$

How to find the most optimal split ? → This is NP hard problem

Trick → We try out every single split in the dataset and choose the one which has min entropy, H(S).



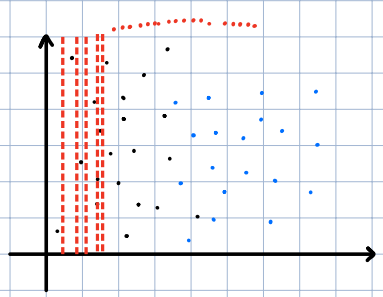
This dataset, as a whole, has a large entropy.

If the dataset is split into half, then entropy on the left is minimized. (all black dots)

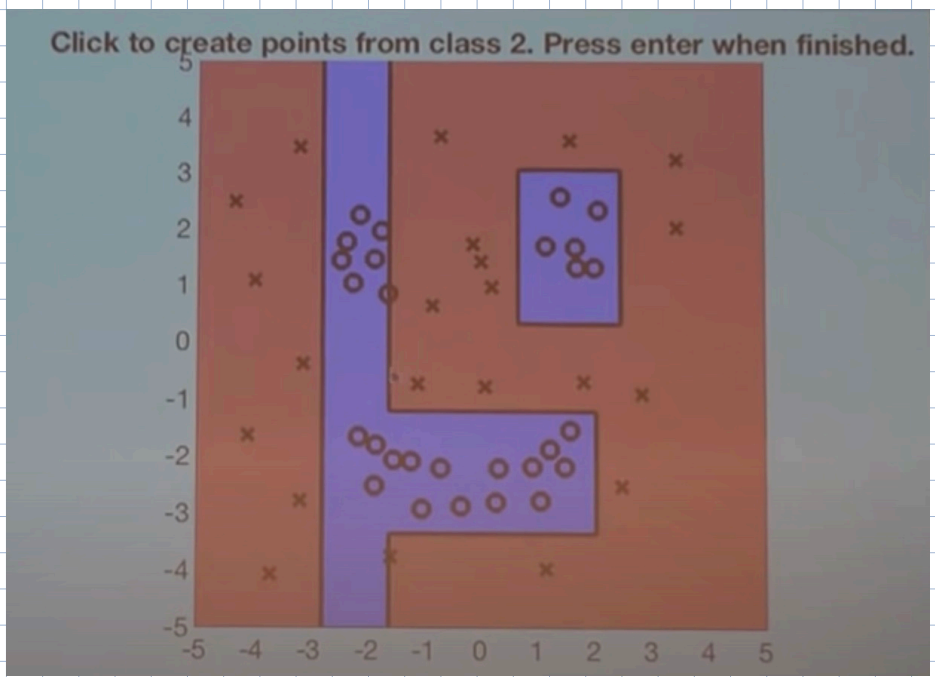
On right, there is one black data point. Create splits to isolate it.

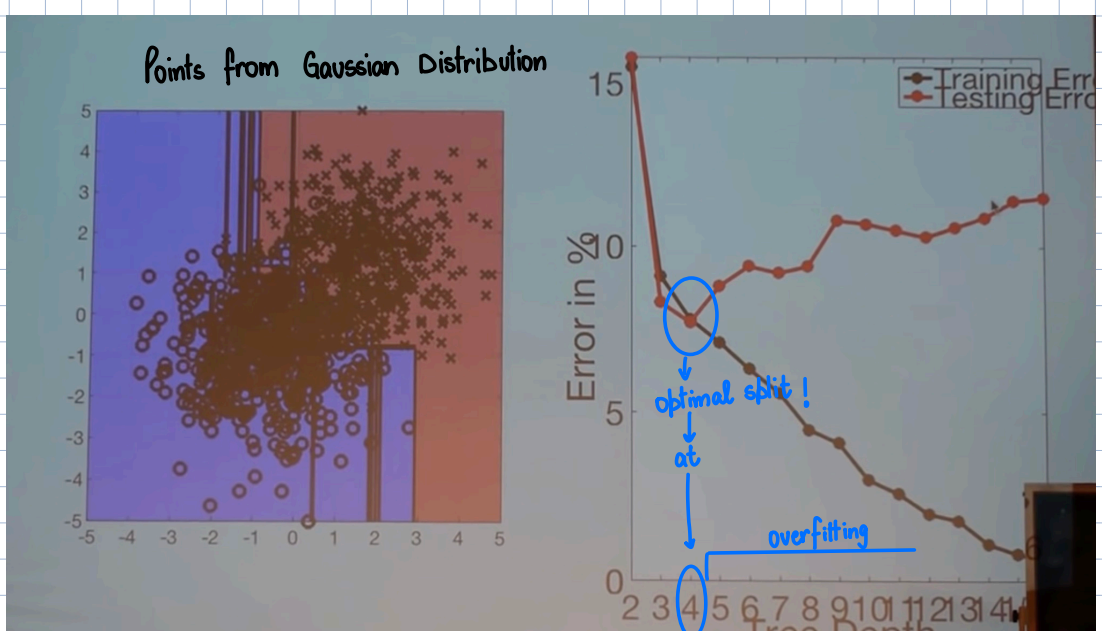
Qᵤᵢ. How do we create these splits ?

Ans. Try out every single split in the dataset and compute the entropy of two leaves. and pick the one that is the lowest.



We can do it in every single dimension.


Click to create points from class 2. Press enter when finished.


Points from Gaussian Distribution
Training Err
Testing Erro
Error in %
optimal split !
at
overfitting