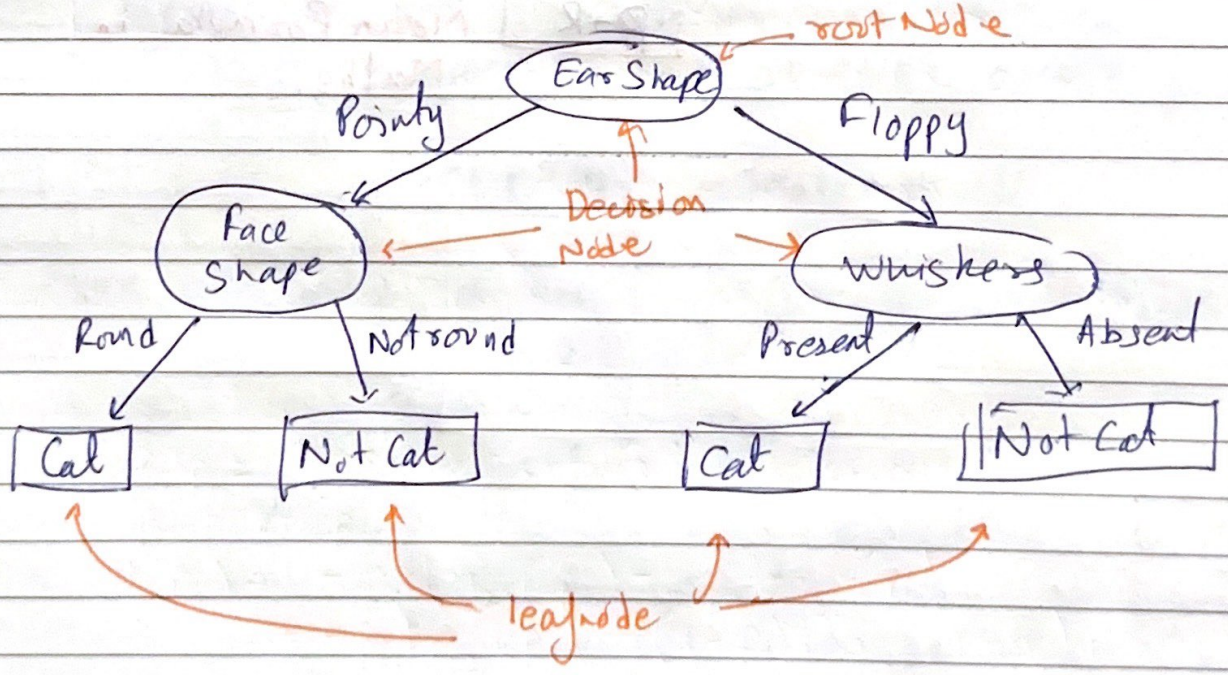
Decision Trees

Eg. We have data set to check for wether an animal is a cat or no

x_1	x_2	x_3	y
Ear shape	Face Shape	Whiskers	Cat
Pointy	Round	Present	1
Floppy	Not Round	Present	1
Floppy	Round	Absent	0
Pointy	Not Round	Present	0
⋮	⋮	⋮	⋮

We can represent the trained model like



Decision Tree Building Process

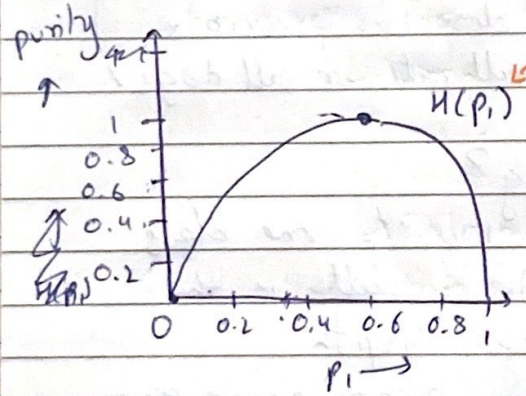
Let us use the same examples of first example.

- Decision 1: How to choose what feature to split on at each node?
  - Maximize purity (get a set that is as close as possible to all cats or all dogs)
- Decision 2: When do we stop splitting?
  - When a node is 100% one class
  - When splitting a node results in the tree exceeding maximum depth.
  - When improvements in purity score are below threshold (minimum improvement) (to reduce risk of overfitting)
  - When number of examples in a node is below a threshold. (when we have small sets) (to reduce risk of overfitting)

Entropy as a Measure of Impurity.

If a set is all one class, eg. cat, or not cat, eg dog the set is called pure.

let p_1 = fraction of examples that are at
= 3/6.



$\therefore H(p_1) = 1$

impurity is highest when fraction is 50-50.
least when its 0 or 1

if $p_1 = 5/6$, $H(p_1) = 0.65$
if $p_1 = 6/6$, $H(p_1) = 0$
if $p_1 = 0$, $H(p_1) = 0$
if p_1 2/6, $H(p_1) = 0.92$

$\therefore$ if p_1 = ratio of set (ratio of 1 class vs total examples)
$p_0 = 1 - p_1$

$$\therefore H(p_1) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)$$
$$= -p_1 \log_2(p_1) - (1-p_1) \log_2(1-p_1)$$

Entropy

Note: $0 \log(0) = 0$

Information Gain
Choosing Split

Information gain is the reduction of entropy

Information gain =

$$H(p_1^{root}) - \left(w^{left} H(p_1^{left}) + w^{right} H(p_1^{right})\right)$$

where w = number of examples in the subset divided by total examples

Ex.

$p_1 = 5/10 = 0.5$ $H(0.5) = 1$

Ear Shape

Pointy → $p_1 = 4/5 = 0.8$, $H(0.8) = 0.72$

Floppy → $p_1 = 1/5 = 0.2$, $H(0.2) = 0.72$

$$H(0.5) - \left(\frac{5}{10} H(0.8) + \frac{5}{10} H(0.2)\right)$$

$$= 0.28$$

$p_1 = 5/10$ $H(0.5) = 1$

Whiskers

Present → $p_1 = 3/4 = 0.75$, $H(0.75) = 0.81$

Absent → $p_1 = 2/6 = 0.33$, $H(0.33) = 0.92$

$$H(0.5) - \left(\frac{4}{10} H(0.75) + \frac{6}{10} H(0.33)\right)$$

$$= 0.12$$

Information gain

bigger information gain → better split

Putting Entropy & Information Gain together

- Start with all examples at the root node
- Calculate information gain for all possible features, and pick one with the highest information gain
- Split dataset according to selected features, and create left & right branches of the tree
- Keep repeating until stopping criteria is met
  - When a node is 100% one class
  - When splitting a node will result in the tree exceeding a max depth
  - Information gain is less than threshold
  - When number of examples in a node is below a threshold.

One-Hot Encoding

Used when a feature has more than 2 ~~va~~ classes

When a class has more than 2 possibilities for a feature, we divide it into 3 features that have 2 classes → True or False.

∴ If a categorical feature can take on k values, create k features (0 or 1 valued)

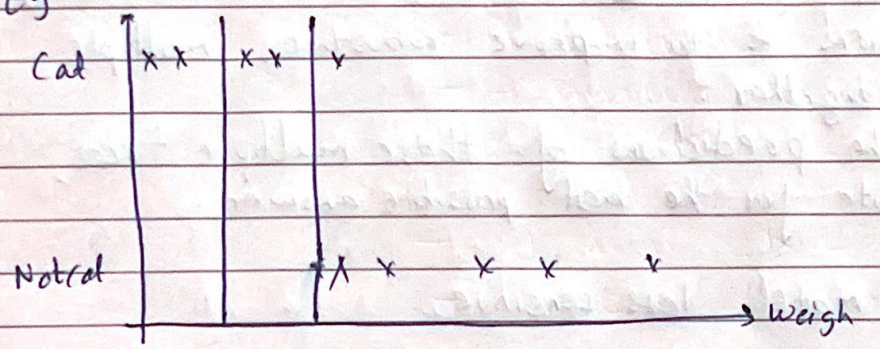
This can also used to train NN & DL models

Continuous features

What to do if a features ~~can~~ has no descrete classes.

We plot the data on a graph and draw boundaries to split the set into two and use the information gain formula to find the best threshold.

Eg



$$H(0.5) - \left(\frac{2}{10} H\left(\frac{2}{2}\right) + \frac{8}{10} H\left(\frac{3}{8}\right)\right) = 0.24$$

$$H(0.5) - \left(\frac{4}{10} H\left(\frac{4}{4}\right) + \frac{6}{10} H\left(\frac{1}{6}\right)\right) = 0.61$$

~~Regression Tree~~

- ~~If we use the same data set for~~

Tree Ensemble

using multiple decision trees.

If we have only 1 tree, introduction of new data can lead to wrong prediction or completely new tree.

∴ To help in decisions & to improve robustness, multiple trees are used together.

And using the predictions of these multiple trees, we can vote for the best possible answer.

This makes the model less sensitive.

Sampling with Replacement

- We take a group of data and select n examples at random always replacing what we took with the new choice.

After this we get random sets of data which we can train Decision trees on & create an ensemble.

Eg. bag with 4 tokens R B Y G

Taking one, reading it, placing it back in a bag and repeat 4 times.

Random Forest Algorithm

Used to create multiple trees.

- Given a training set of size 'm'
- For b = 1 to B:
  - Use sampling with replacement to create a new training set of size m.
  - Train a decision tree on the new data set

To make the algorithm much better, we need to ~~use~~ randomize the feature choice

At each node, when choosing a feature to split, if n features are available, pick a random subset of k < n features and allow the program to choose from that subset of features.

XGBoost

The implementation of Decision tree ensemble which are quick, accurate and easy to build.
They use the concept of Boosted Tree.

- Given training set of size m
- For b=1 to B:
  - Use sampling with replacement to create new training set of size m
    But instead of picking all examples with equal (1/m) probability make it more likely to pick ~~to pick~~ misclassified examples from previously trained trees.
  - Train tree on the new data set

↑ Boosted Tree Algo ↑

XGBoost (eXtreem Gradiant Boosting)
- open source implementation of boosted tree
- fast efficient implementation
- Good choice of default splitting criteria & criteria for when to stop splitting
- Built in regularization to prevent overfitting

Eg.
```
from xgboost import XGBClassifier
from xgboost import XGBRegressor
model1 = XGBClassifier()
model2 = XGBRegressor()
model1.fit(X, y)
model1.predict(X_test)
```

Decision Trees vs Neural Networks

Decision Trees & Tree ensemble

- Works well on tabular (structured) data
- Not recommended for unstructured data (images, audio, text)
- Fast ~~training time~~
- Small decision trees may be human interpretable

Neural Networks

- Works well on all types of data, including structured and unstructured data.
- May be slower than a decision tree.
- Works with transfer learning (allows us to use pretrained models to reduce training time)
- When building a system of multiple models working together, it might be easier to string together multiple Neural Networks.