Tree Ensembles

- Single decision trees can be highly sensitive to small changes in the data, leading to different results with minor alterations.
- To make the algorithm more robust, multiple decision trees can be built, creating a tree ensemble.
- Changing a single training example can cause the algorithm to choose a different feature to split on, resulting in a completely different tree.
- More accurate predictions can be achieved by training a collection of different decision trees, or a tree ensemble.
- Each tree in the ensemble can make a prediction for a new test example, and the final prediction is determined by majority vote.

Sampling with replacement

Sampling with replacement in the context of decision trees is a technique used to create multiple different training sets from the original training set.

▼ Here's how it works:

1. Imagine all your training examples are in a large bag. Each training example is like a token in this bag.
2. You then draw a training example from this bag, record it, and then put it back into the bag. This is the "replacement" part - every time you draw an example, it goes back into the bag, making it possible to be drawn again.
3. You repeat this process as many times as there are examples in your original training set. This means if your original training set had 100 examples, you would draw 100 examples with replacement to create a new training set.
4. Because you're drawing with replacement, your new training set might have multiple copies of some examples and might be missing some other examples entirely.

5. You then use this new training set to build a decision tree.
6. Steps 2-5 are repeated to create multiple decision trees, each built from a different training set created by sampling with replacement. This collection of trees is your tree ensemble or random forest.

Random forest algorithm

The Random Forest algorithm is a powerful ensemble method that uses multiple decision trees to make predictions. Here's how it works:

1. Given a training set of size M, you perform the following steps B times (where B is the number of trees you want in your forest, typically around 100):

   a. Use sampling with replacement to create a new training set of size M. This means you put all your training examples in a "virtual bag" and draw examples with replacement until you have a new set of M examples.

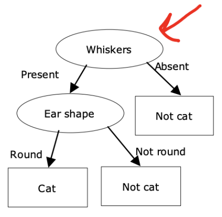
   b. Train a decision tree on this new training set. The tree will be different each time because the training set is different each time.

2. Once you have built your ensemble of B trees, you use them to make predictions. Each tree in the ensemble "votes" on the correct prediction, and the final prediction is determined by majority vote.

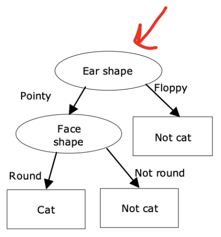
Generating a tree sample

```
Given 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 dataset
```

Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Pointy	Round	Present	Yes
Pointy	Not Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Present	Yes
Pointy	Not Round	Absent	No
Pointy	Not Round	Absent	No
Pointy	Not Round	Present	Yes



Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	Yes
Pointy	Round	Absent	Yes
Floppy	Not Round	Absent	No
Floppy	Not Round	Absent	No
Pointy	Round	Absent	Yes
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Pointy	Not Round	Absent	No
Pointy	Round	Present	Yes



...

Bagged decision tree

One modification to this algorithm can make it work even better:

Randomizing the feature choice

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

$K = \sqrt{n}$

Random forest algorithm

This further randomises the trees and makes them more different from each other, which can lead to more accurate predictions when they are combined. **This modification turns the algorithm into a Random Forest.**

XGBoost algorithm

The XGBoost algorithm, which stands for Extreme Gradient Boosting, is a popular and efficient implementation of the gradient boosting framework. Here's how it works:

1. Given a training set of size M, you repeat B times (where B is the number of trees you want in your ensemble):

   a. Instead of using sampling with replacement to create a new training set, XGBoost assigns different weights to different training examples. This makes it more efficient than using a sampling with replacement procedure.

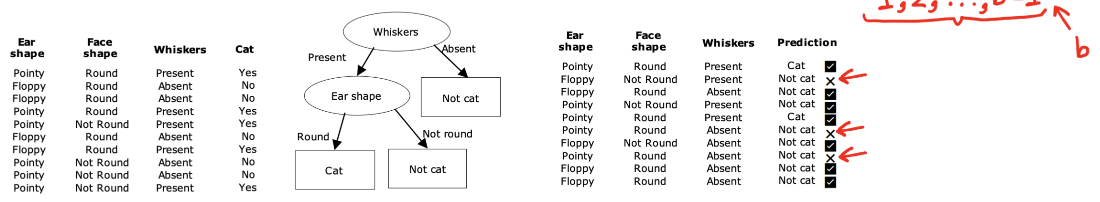
   b. The weights are adjusted such that examples that were misclassified by the previously trained trees are given more weight. This means that the algorithm focuses more on the examples it's not yet doing well on, a process similar to deliberate practice.

c. Train a decision tree on this weighted training set. The tree will be different each time because the weights are different each time.

2. Once you have built your ensemble of B trees, you use them to make predictions. Each tree in the ensemble "votes" on the correct prediction, and the final prediction is determined by majority vote.

Boosted trees intuition

```
Given training set of size $m$
For $b$ = 1 to $\underline{B}$:
    Use sampling with replacement to create a new training set of size $m$
        But instead of picking from all examples with equal (1/m) probability, make it
        more likely to pick examples that the previously trained trees misclassify
    Train a decision tree on the new dataset
```



Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Pointy	Round	Present	Yes
Pointy	Not Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Present	Yes
Pointy	Not Round	Absent	No
Pointy	Not Round	Absent	No
Pointy	Not Round	Present	Yes

Ear shape	Face shape	Whiskers	Prediction	
Pointy	Round	Present	Cat	✓
Floppy	Not Round	Present	Not cat	✗ ←
Floppy	Round	Absent	Not cat	✓
Pointy	Not Round	Present	Not cat	✓
Pointy	Round	Present	Cat	✓
Pointy	Round	Absent	Not cat	✗ ←
Floppy	Not Round	Absent	Not cat	✓
Pointy	Round	Absent	Not cat	✗ ←
Floppy	Round	Absent	Not cat	✓
Floppy	Round	Absent	Not cat	✓

XGBoost also includes built-in regularisation to prevent overfitting, and it has a good choice of the default splitting criteria and criteria for when to stop splitting.

Using XGBoost

Classification

```
→from xgboost import XGBClassifier

→model = XGBClassifier()

→model.fit(X_train, y_train)
→y_pred = model.predict(X_test)
```

Regression

```
from xgboost import XGBRegressor

model = XGBRegressor()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Decision Trees vs Neural Networks

Basis for Categorisation	Decision Tree	Neural Network
Type of Data	Works well with structured or tabular data.	More suitable for unstructured data such as images, audio, video, and text.
Training Speed	Fast to train, allowing for quick iterations.	Can take a longer time to train, especially for large networks.
Interpretability	Small trees can be easily interpreted and visualized. Interpretability decreases as the tree becomes larger or when using an ensemble of trees.	Generally harder to interpret due to their complex structure.
Transfer Learning and Complex Systems	Not typically used for transfer learning. Can be more difficult to string together multiple models.	More suitable for transfer learning. Easier to string together and train multiple models.
Computational Budget	Single decision tree might be more suitable for very constrained computational budgets.	Can be computationally expensive, especially for large networks.