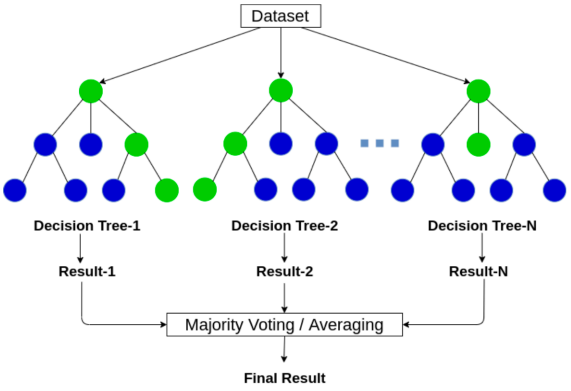
Tree ensembles

- A single decision tree can be sensitive to small changes in data, leading to different predictions.
- To improve robustness, multiple decision trees are built, forming a tree ensemble.

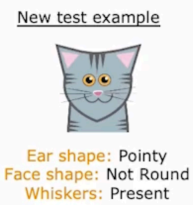


Using multiple Decision Trees

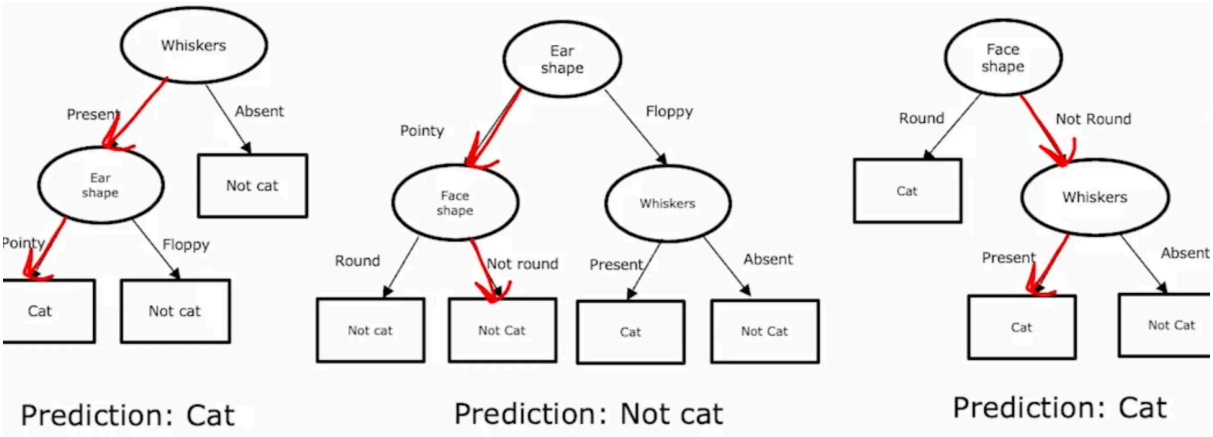
Trees are highly sensitive to small changes of the data

Voting Mechanism

- Each tree in the ensemble makes predictions on **new examples**, and the final prediction is based on majority voting.
  - **1st** Decision Tree predicts **Cat**
  - **2nd** Decision Tree predicts **Not Cat**
  - **3rd** Decision Tree predicts **Cat**



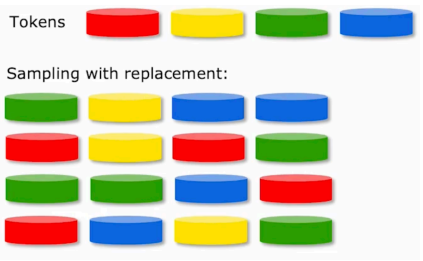
⇒ Accept: **CAT**



- This approach reduces the impact of any single tree's prediction, enhancing overall accuracy.

Sampling with replacement

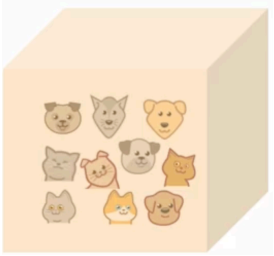
- Demonstration using colored tokens illustrates the process: tokens are drawn from a bag, noted, replaced, and drawn again.
- This method allows for the same token to be selected multiple times, leading to varied outcomes in each sampling round.



By allowing duplicates, you can create multiple training sets that are slightly different from each other, which helps in training models that generalize better to new data.

Application in Tree Ensembles

- In constructing tree ensembles, multiple random training sets are created from an original dataset.



	Ear shape	Face shape	Whiskers	Cat
	Pointy	Round	Present	1
	Floppy	Not round	Absent	0
	Pointy	Round	Absent	1
	Pointy	Not round	Present	0
	Floppy	Not round	Absent	0
	Pointy	Round	Absent	1
	Pointy	Round	Present	1

- Each new training set may contain duplicates and not all original examples, which is acceptable and necessary for diversity in the ensemble.

Random Forest algorithm

Generating a Tree sample

Given training set of size m

For $b = 1$ to B:

1. Use Sampling with Replacement to create a new training set of size m
2. Train a decision tree on the new dataset

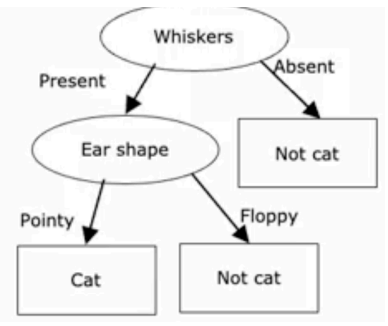
Example:

- $b = 1$:

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

Decision Tree after training:

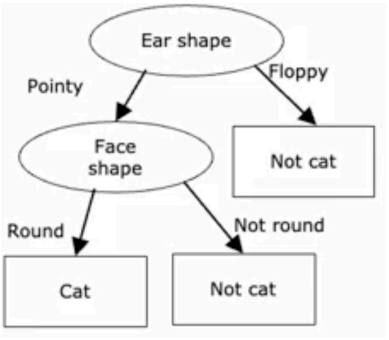


- $b = 2$:

Dataset:

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

Decision Tree after training:



Setting the number of trees, denoted as **B**, in the random forest algorithm is important for balancing performance and computational efficiency.

- **Typical Range**: A common choice for **B** is around **100 trees**. Values can range from **64 to 128** or more, depending on the specific problem and dataset.
- **Performance Impact**: Increasing **B** generally improves the model's performance, as more trees can lead to better averaging of predictions. However, after a certain point, the performance gains diminish.
- **Diminishing Returns**: Beyond about **100 trees**, adding more trees may not significantly enhance accuracy but can slow down computation. Therefore, it's often unnecessary to use very high values like **1000 trees**.

This specific instance creation of tree ensemble is sometimes also called a **Bagged Decision Tree**.

Randomizing the feature choice

- To further enhance the algorithm, a random subset of features is selected at each node for splitting, rather than using all available features.

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.

- When n is large, the typical choice for k is $k = \sqrt{n}$
- The random forest algorithm (just using trick in generating sample) averages the predictions from multiple trees, making it less sensitive to small changes in the training data.
- It is generally more effective and robust than using a single decision tree, providing better performance in various applications.

Example:

Random Forest = Bagging + Random Feature Selection

Random Forest adds a twist to Bagging:

- At **each split** in the tree, only a **random subset of features** is considered.
    - Let's say:
        - You have 3 features: Ear shape, Face shape, Whiskers.
        - At each split, the algorithm randomly selects **2 of 3** features.

For example:

- At one node, it might pick Ear shape and Whiskers.
- Another node might get Face shape and Whiskers.

Input:

- Training set of size m with d features
- Number of trees B
- Max features k per split

Steps:

1. Initialize an empty list: ensemble ← $\{\}$
2. For $b = 1$ to B do:
    a. newTrainingData ← SampleWithReplacement(TrainingSet, m)
    b. newDecisionTree ← BuildDecisionTreeWithRandomFeatures(newTrainingData, k)
    c. ensemble.append(newDecisionTree)
3. Return ensemble

XGBoost

XGBoost stands for eXtreme Gradient Boosting and is known for its speed and efficiency.

Boosted Trees intuition

Given training set of size m

For $b = 1$ to B:

1. Use Sampling with Replacement to create a new training set of size m
    a. But instead of picking from all examples with equal ($\frac{1}{m}$) probability, make it more likely to pick misclassified examples from previously trained trees
2. Train a decision tree on the new dataset

Example

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 ✓

The (X) examples (misclassified) is prior to be picked in the next trees.

- The algorithm improves decision trees by focusing on misclassified examples from previous iterations.
- This approach is similar to deliberate practice in learning, where attention is given to areas needing improvement.

Focus on Misclassified Examples: For each new tree (let's call it tree number **b**), the algorithm looks at the previous trees (from tree 1 to tree **b-1**) and identifies which examples they misclassified.

XBoost (eXtreme Gradient Boosting)

- Open source implementation of boosted trees
- Fast efficient implementation
- Good choice of default splitting criteria and criteria for when to stop splitting
- Built in regularization to prevent overfitting
- Highly competitive algorithm for machine learning competitions (eg: Kaggle competitions)

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)
```

When to use Decision Tree

Decision Tree vs Neural Network

Decision Tree and Tree ensembles

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

Neural Networks

- Works well on all types of data, including tabular (structured) and unstructured data
- May be slower than a decision tree
- Works with transfer learning
- When building a system of multiple models working together, it might be easier to string together multiple neural networks
    - It relates to that even when you string together multiple neural networks you can train them all together using gradient descent. Whereas for decision trees you can only train one decision tree at a time.

Example

Random Forest

Now let's try the Random Forest algorithm also, using the Scikit-learn implementation.

- All of the hyperparameters found in the decision tree model will also exist in this algorithm, since a random forest is an ensemble of many Decision Trees.

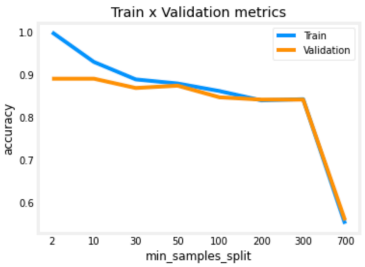
- One additional hyperparameter for Random Forest is called `n_estimators` which is the number of Decision Trees that make up the Random Forest.

Remember that for a Random Forest, we randomly choose a subset of the features AND randomly choose a subset of the training examples to train each individual tree.

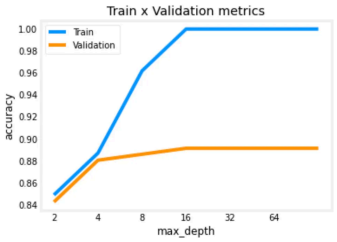
- Following the lectures, if n is the number of features, we will randomly select $\sqrt{n}$ of these features to train each individual tree.
- Note that you can modify this by setting the `max_features` parameter.

You can also speed up your training jobs with another parameter, `n_jobs`.

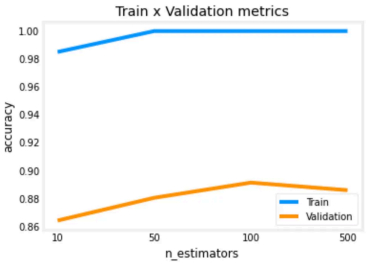
- Since the fitting of each tree is independent of each other, it is possible fit more than one tree in parallel.
- So setting `n_jobs` higher will increase how many CPU cores it will use. Note that the numbers very close to the maximum cores of your CPU may impact on the overall performance of your PC and even lead to freezes.
- Changing this parameter does not impact on the final result but can reduce the training time.



using `min_samples_split`



using `max_depth`



using `n_estimators`

Let's then fit a random forest with the following parameters:

- `max_depth = 16`
- `min_samples_split = 10`
- `n_estimators = 100`

Metrics train:
Accuracy score: 0.9360
Metrics test:
Accuracy score: 0.8859

XGBoost

The boosting methods train several trees, but instead of them being uncorrelated to each other, now the trees are fit one after the other in order to minimize the error.

The model has the same parameters as a decision tree, plus the learning rate.

- The learning rate is the size of the step on the Gradient Descent method that the XGBoost uses internally to minimize the error on each train step.

One interesting thing about the XGBoost is that during fitting, it can take in an evaluation dataset of the form `(X_val,y_val)`.

- On each iteration, it measures the cost (or evaluation metric) on the evaluation datasets.
- Once the cost (or metric) stops decreasing for a number of rounds (called `early_stopping_rounds`), the training will stop.
- More iterations lead to more estimators, and more estimators can result in overfitting.
- By stopping once the validation metric no longer improves, we can limit the number of estimators created, and reduce overfitting.

(we should not use the test set here)

```
xgb_model = XGBClassifier(n_estimators = 500,
                          learning_rate = 0.1,
                          verbosity = 1,
                          random_state = RANDOM_STATE)
xgb_model.fit(X_train_fit,y_train_fit,
              eval_set = [(X_train_eval,y_train_eval)],
              early_stopping_rounds = 10)
```

```
Metrics train:
    Accuracy score: 0.9251
Metrics test:
    Accuracy score: 0.8641
```

Even though we initialized the model to allow up to 500 estimators, the algorithm only fit 26 estimators (over 26 rounds of training). To see why, let's look for the round of training that had the best performance (lowest evaluation metric) with `xgb_model.best_iteration`

The best round of training was round 16, with a log loss of 4.3948.

- For 10 rounds of training after that (from round 17 to 26), the log loss was higher than this.
- Since we set `early_stopping_rounds` to 10, then by the 10th round where the log loss doesn't improve upon the best one, training stops.
- You can try out different values of `early_stopping_rounds` to verify this. If you set it to 20, for instance, the model stops training at round 36 ($16 + 20$).

❓ **QUES:** For the random forest, how do you build each individual tree so that they are not all identical to each other?

- Sample the training data with replacement and select a random subset of features to build each tree 👍
- Sample the training data without replacement
- If you are training B trees, train each one on 1/B of the training set, so each tree is trained on a distinct set of examples.
- Train the algorithm multiple times on the same training set. This will naturally result in different trees.

Explain: Correct. You can generate a training set that is unique for each individual tree by sampling the training data with replacement. The random forest algorithm further avoids identical trees by randomly selecting a subset of features when building the tree ensemble.

❓ **QUES:** You are choosing between a decision tree and a neural network for a classification task where the input x is a 100×100 resolution image. Which would you choose?

- A neural network, because the input is unstructured data and neural networks typically work better with unstructured data. 👍
- A decision tree, because the input is structured data and decision trees typically work better with structured data.
- A decision tree, because the input is unstructured and decision trees typically work better with unstructured data.
- A neural network, because the input is structured data and neural networks typically work better with structured data.

❓ **QUES:** What does sampling with replacement refer to?

- It refers to a process of making an identical copy of the training set.
- It refers to using a new sample of data that we use to permanently overwrite (that is, to replace) the original data.
- Drawing a sequence of examples where, when picking the next example, first replacing all previously drawn examples into the set we are picking from. 👍
- Drawing a sequence of examples where, when picking the next example, first remove all previously drawn examples from the set we are picking from.