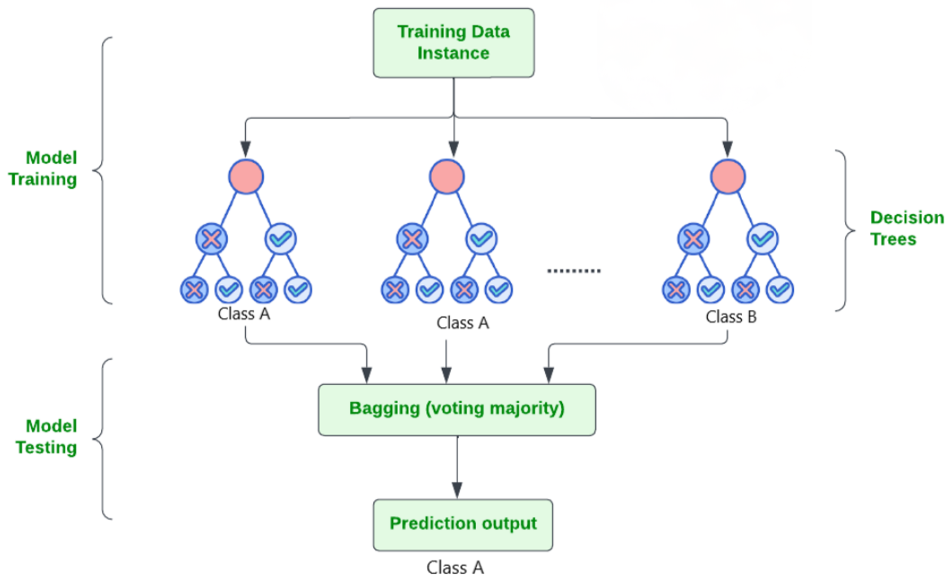
Random forest classifier

- Random Forest algorithm is a powerful tree learning technique in **Machine Learning**. It works by creating a number of **Decision Trees** during the training phase. Each tree is constructed using a random subset of the data set to measure a random subset of features in each partition.
- Random forests can be used for solving regression (numeric target variable) and classification (categorical target variable) problems.
- Random forests are an ensemble method, meaning they combine predictions from other models.
- Each of the smaller models in the random forest ensemble is a decision tree.



Working of Random Forest Classification

- In a random forest classification, multiple decision trees are created using different random subsets of the data and features. Each decision tree is like an expert, providing its opinion on how to classify the data. Predictions are made by calculating the prediction for each decision tree, then taking the most popular result.

- Each tree is exposed to a different number of features and a different sample of the original dataset, and as such, every tree can be different. Each tree makes a prediction. Looking at the first 5 trees, we can see that 4/5 predicted the sample was a Cat. The green circles indicate a hypothetical path the tree took to reach its decision. The random forest would count the number of predictions from decision trees for Cat and for Dog, and choose the most popular prediction.

The Dataset

This dataset consists of direct marketing campaigns by a Portuguese banking institution using phone calls. The campaigns aimed to sell subscriptions to a bank term deposit. We are going to store this dataset in a variable called bank_data.

The columns we will use are:

- age: The age of the person who received the phone call
- default: Whether the person has credit in default
- cons.price.idx: Consumer price index score at the time of the
- callcons.conf.idx:Consumer confidence index score at the time of the call
- y: Whether the person subscribed

Random Forests Workflow

To fit and train this model, we'll be following The Machine Learning Workflow infographic; however, as our data is pretty clean, we won't be carrying out every step.

We will do the following:

- Feature engineering
- Split the data
- Train the model
- Hyperparameter tuning
- Assess model performance

Preprocessing Data for Random Forests

Tree-based models are much more robust to outliers than linear models, and they do not need variables to be normalized to work. As such, we need to do very little preprocessing on our data.

- We will map our ‘default’ column, which contains no and yes, to 0s and 1s, respectively. We will treat unknown values as no for this example.
- We will also map our target, y, to 1s and 0s.

Splitting the Data

- When training any supervised learning model, it is important to split the data into training and test data. The training data is used to fit the model. The algorithm uses the training data to learn the relationship between the features and the target. The test data is used to evaluate the performance of the model.
- The code below splits the data into separate variables for the features and target, then splits into training and test data.

```
# Split the data into features (X) and target (y)
X = bank_data.drop('y', axis=1)y = bank_data['y']
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, tes
```

Hyperparameter Tuning

We define the hyperparameters to use and their ranges in the param_dist dictionary. In our case, we are using:

- n_estimators: the number of decision trees in the forest. Increasing this hyperparameter generally improves the performance of the model but also increases the computational cost of training and predicting.
- max_depth: the maximum depth of each decision tree in the forest. Setting a higher value for max_depth can lead to overfitting while setting it too low can lead to underfitting.

```
param_dist = {'n_estimators': randint(50,500),
              'max_depth': randint(1,20)}
```

```
# Create a random forest classifier
rf = RandomForestClassifier()

# Use random search to find the best hyperparameters
rand_search = RandomizedSearchCV(rf,
                                 param_distributions = param_
                                 n_iter=5,
                                 cv=5)

# Fit the random search object to the data
rand_search.fit(X_train, y_train)
```

`RandomizedSearchCV` will train many models (defined by n_iter_ and save each one as variables, the code below creates a variable for the best model and prints the hyperparameters. In this case, we haven't passed a scoring system to the function, so it defaults to accuracy. This function also uses cross validation, which means it splits the data into five equal-sized groups and uses 4 to train and 1 to test the result. It will loop through each group and give an accuracy score, which is averaged to find the best model.

```
# Create a variable for the best model
best_rf = rand_search.best_estimator_

# Print the best hyperparameters
print('Best hyperparameters:',  rand_search.best_params_)
```

Output:

```
Best hyperparameters: {'max_depth': 5, 'n_estimators': 260}
```