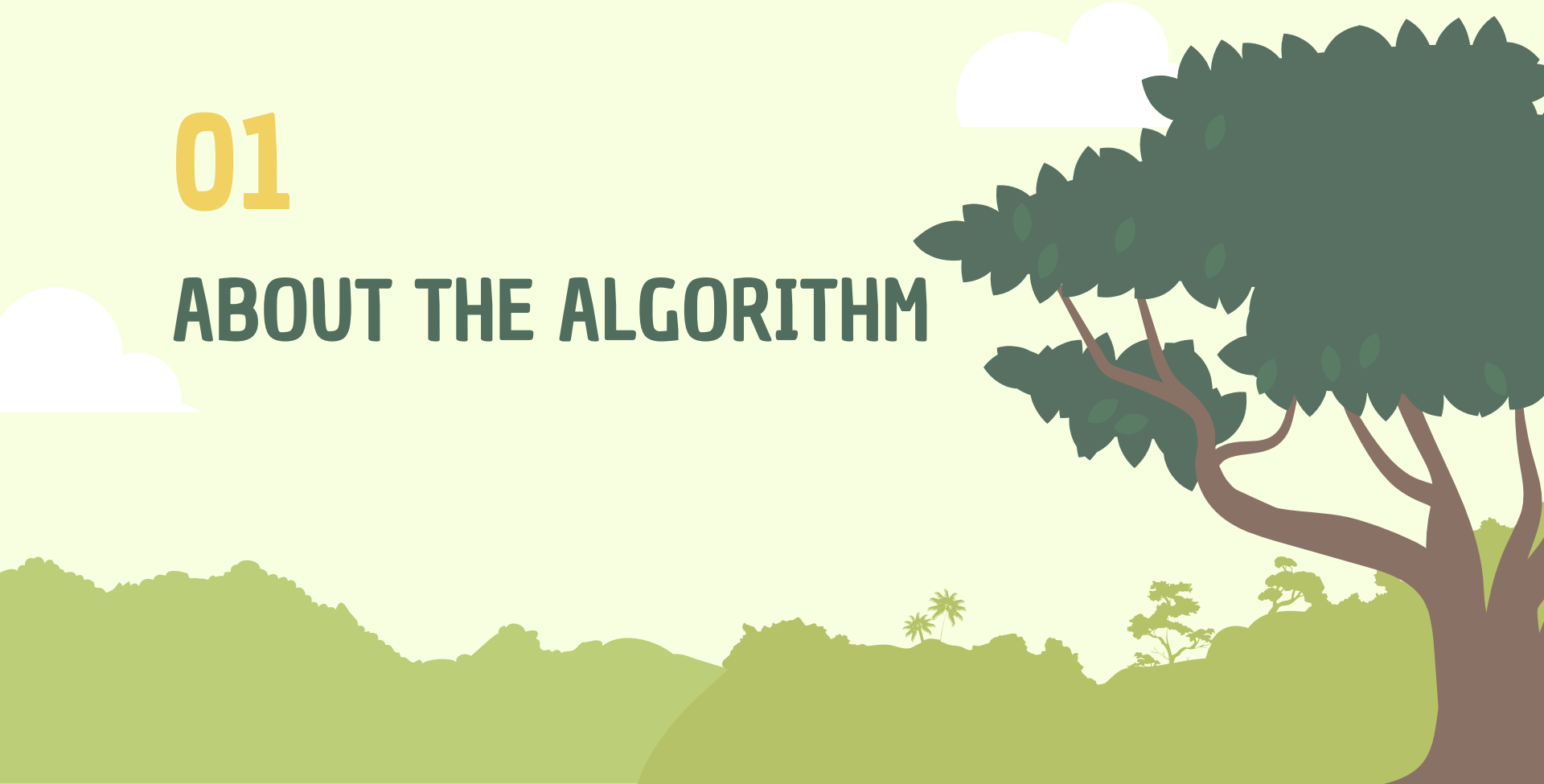
RANDOM FOREST ALGORITHM

Roberto Di Via S4486648 - DIBRIS

TABLE OF CONTENTS

01

ABOUT THE ALGORITHM

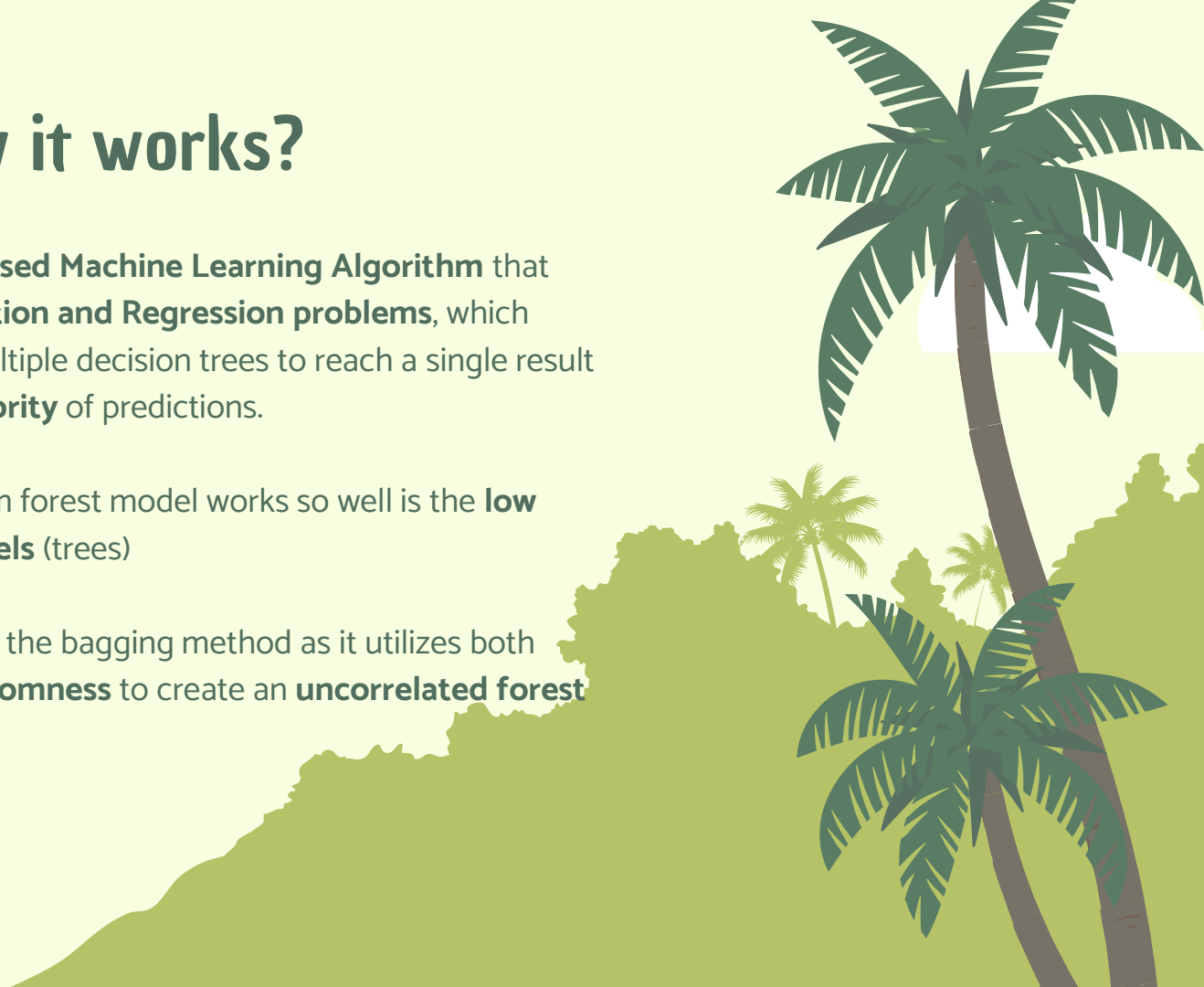


What is? How it works?

Random forest is a **Supervised Machine Learning Algorithm** that is used widely **in Classification and Regression problems**, which combines the output of multiple decision trees to reach a single result taking the **average or majority** of predictions.

The reason that the random forest model works so well is the **low correlation between models** (trees)

Indeed, it is an extension of the bagging method as it utilizes both **bagging and feature randomness** to create an **uncorrelated forest** of decision trees.



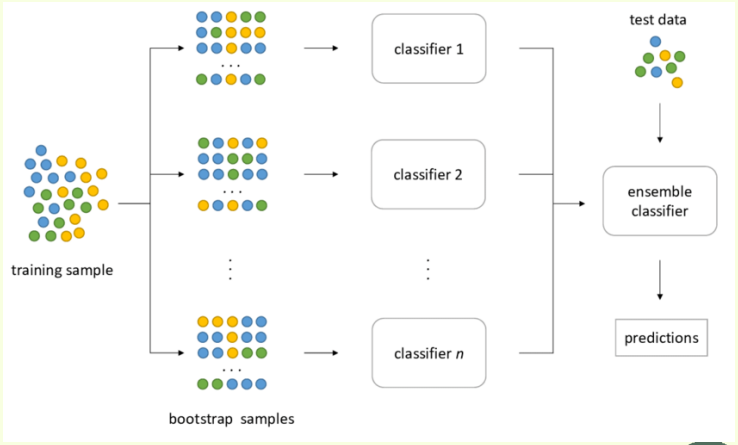
BAGGING and FEATURE RANDOMNESS

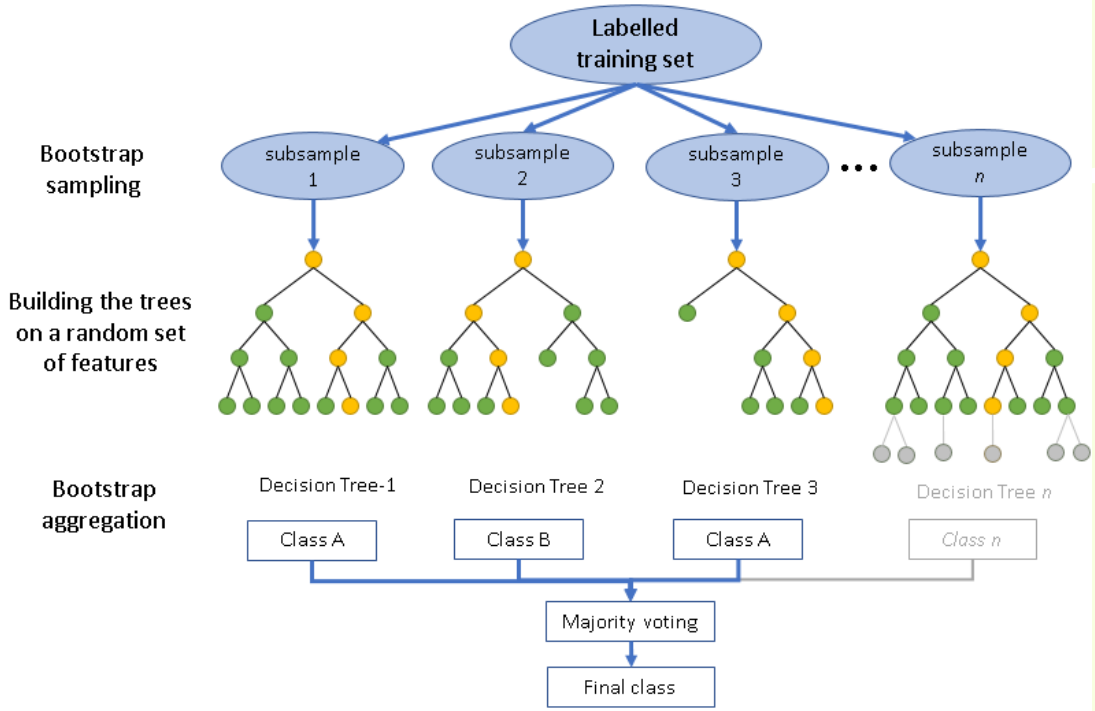
Each tree in the ensemble is comprised of a data sample drawn from a training set **with replacement**, called the **bootstrap sample**. Of that training sample, **one-third** of it is set aside as test data, known as **the out-of-bag** (oob) sample.

Another instance of randomness is then injected through **feature bagging**, that is a random selection of features to split each node.

This adds more diversity to the dataset and reduce the correlation among decision trees.

Finally, the **oob** sample is then used for cross-validation, finalizing that prediction.




Labelled training set
Bootstrap sampling
subsample 1
subsample 2
subsample 3
subsample n
Building the trees on a random set of features
Bootstrap aggregation
Decision Tree-1
Decision Tree 2
Decision Tree 3
Decision Tree n
Class A
Class B
Class A
Class n
Majority voting
Final class

SUMMARY

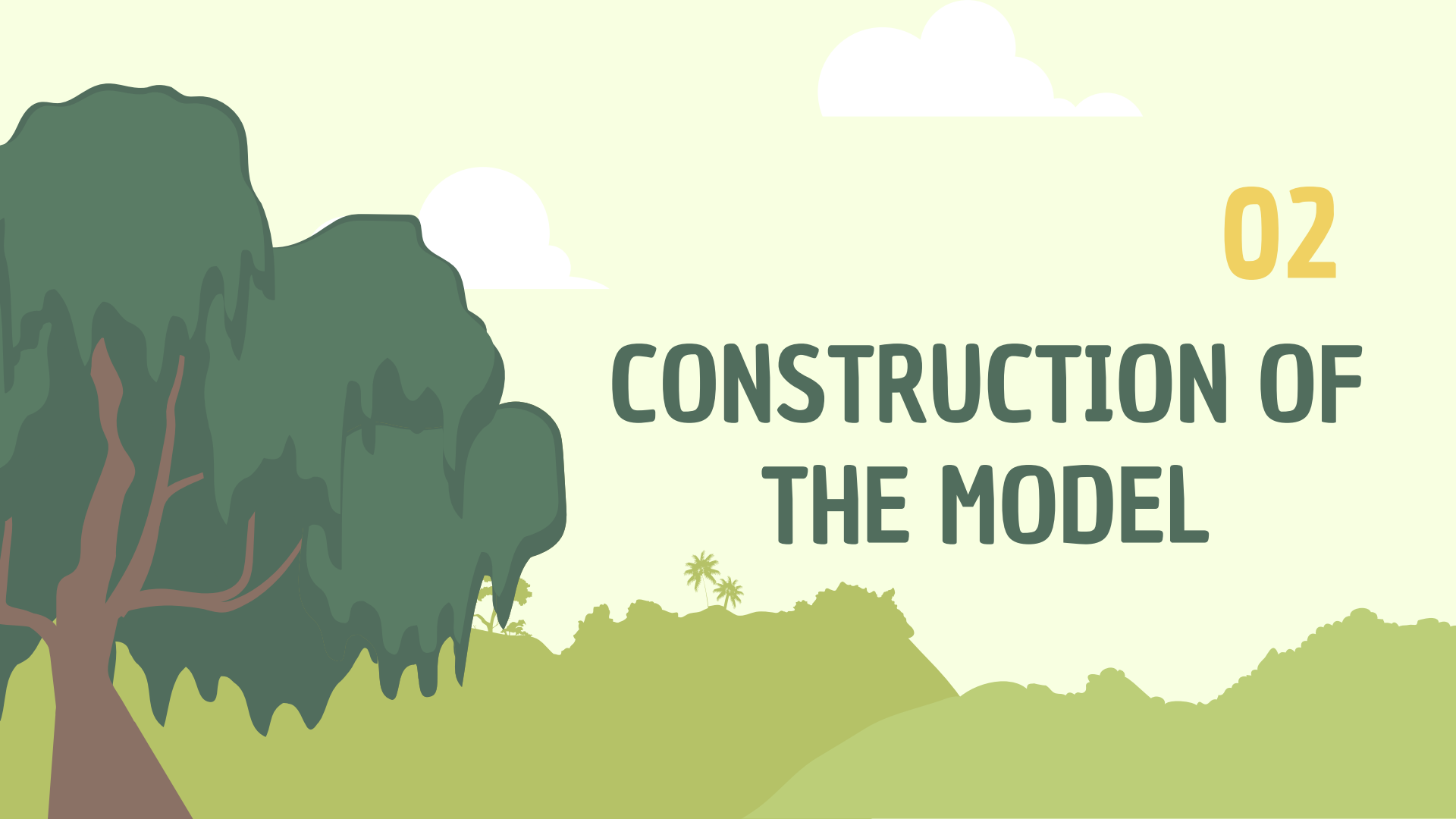
ADVANTAGES:

1. It can be used in **classification and regression** problems.
2. It's more **accurate** than the decision tree algorithm.
3. It **solves** the problem of **overfitting** as output is based on majority voting or averaging.
4. This algorithm is very **stable**. Even if a new data point is introduced in the dataset the overall algorithm is not affected much since new data may impact one tree, but it is very hard for it to impact all the trees.

DISADVANTAGES:

1. Random forest is **highly complex** when compared to decision trees where decisions can be made by following the path of the tree.

2. Training **time is more** compared to other models due to its complexity. Whenever it has to make a prediction each decision tree has to generate output for the given input data.



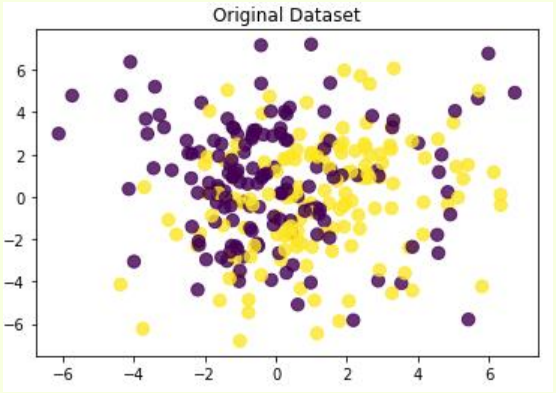
02

CONSTRUCTION OF THE MODEL

DATA GENERATION

sklearn.datasets.make_classification

- **n_samples** → refers to the number of samples
- **n_features** → refers to the number of column/features of dataset
- **flip_y** → refers to the fraction of samples whose class is assigned randomly. Larger values introduce noise in the labels and make the classification task harder. Default is 0.1



sklearn.model_selection.train_test_split

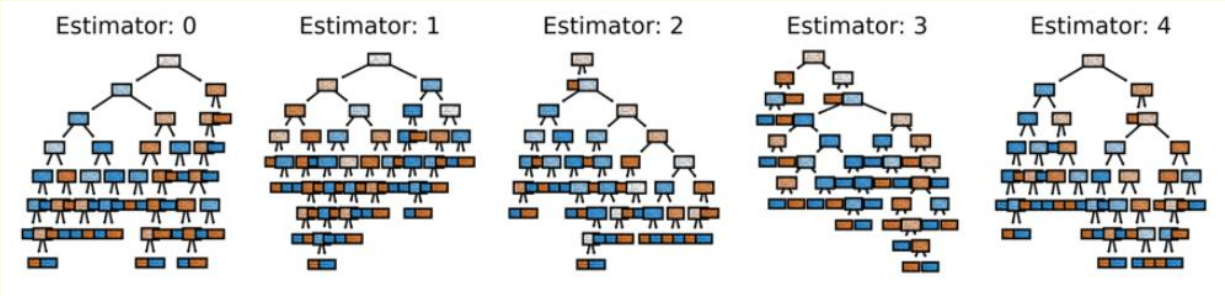
To split the dataset into a random train and test subsets. In my case 70% train set and 30% test set.

- **test_size**→ If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split.

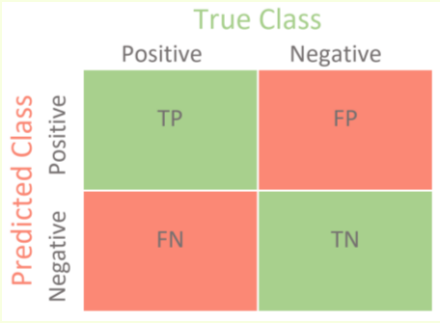
CONSTRUCTION and TRAINING of the Model

sklearn.ensemble.RandomForestClassifier

- **n_estimators** → refers to the number of trees in the forest
- **criterion** → refers to the function to measure the quality of a split. This parameter is tree-specific, the default is “gini”.
- **max_depth** → refers to the maximum depth of the tree. The default is None.
- **max_features**→ refers to the number of features to consider when looking for the best split. The default is *sqrt(n_features)*
- **max_samples** → It can be set to a float between 0 and 1 to control the percentage of the size of the training dataset to make the bootstrap sample used to train each decision tree.



FIRST ANALYSIS




```
Train Result:
================================================
Accuracy Score:  100.00%
________________________________________________
Confusion Matrix:
[[111   0]
[  0  99]]


Test Result:
================================================
Accuracy Score:  85.56%
________________________________________________
Confusion Matrix:
[[37  4]
[ 9 40]]
```

Training set's size: 210 samples
Test set's size: 90 samples

- Only 4 are false positive
- Only 9 are false negative

- Accuracy achieved: 85.56%

03

Comparison between DECISION TREE and RANDOM FOREST

DIFFERENT AMOUNT OF NOISE

In this part i used **different amounts of noise** during the generation of the datasets in order to compare the predictions using the Random Forest Classifier and the Decision Tree Classifier.

From the resulting chart we can observe that the **Random Forest Classifier perform better** than the Decision Tree Classifier, even when the noise increase.

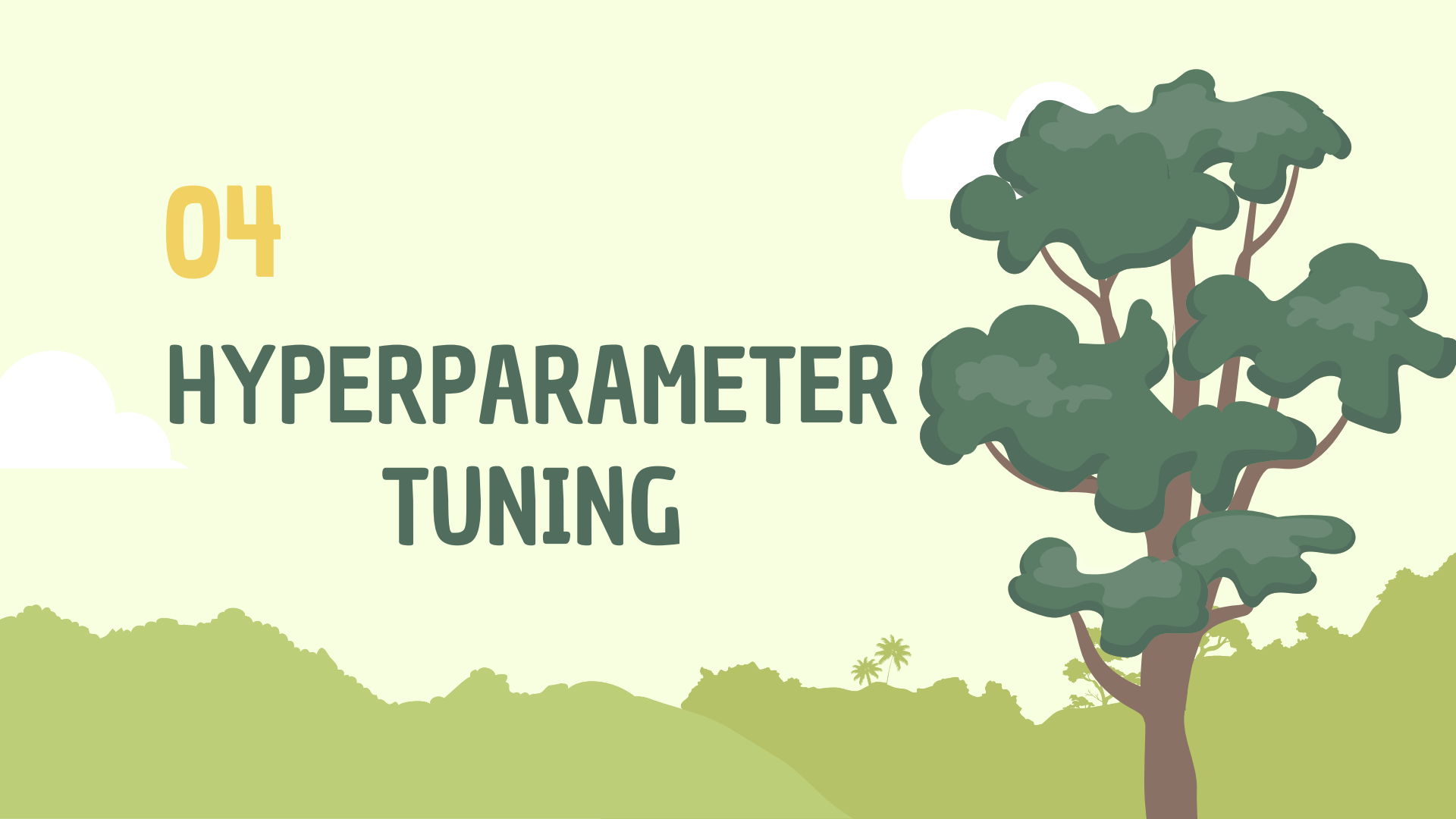
To build the Decision Tree i used the sklearn model: **sklearn.tree**.DecisionTreeClassifier

To compute the error i used the function defined during the ML course


```
def calcError(Ypred, Ytrue):
    return (np.count_nonzero(Ypred != Ytrue)) / len(Ytrue)
```

04

HYPERPARAMETER TUNING



DIFFERENT HYPERPARAMETER TO TUNE

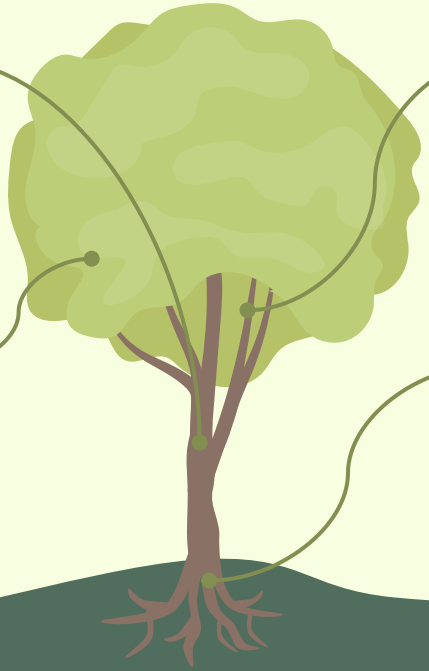
A **hyperparameter** is a parameter whose value is used to control the learning process.

01

NUMBER OF TREES

02

FEATURES PER NODE



03

SAMPLE'S SIZE

04

TREE'S DEPTH

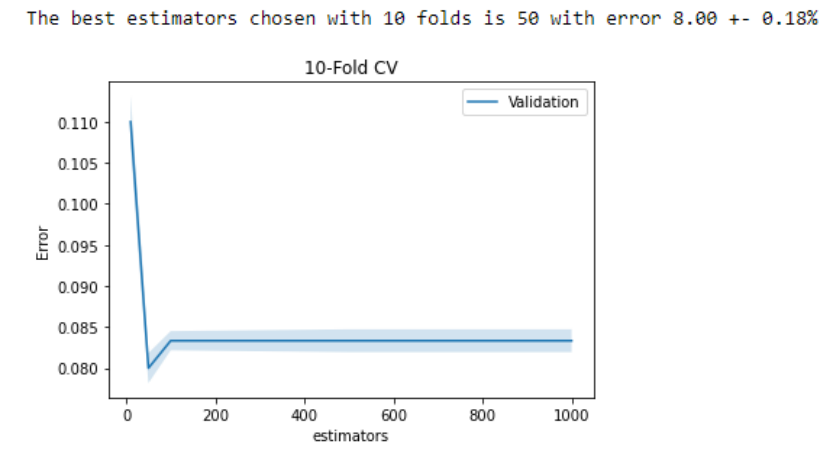
1. DIFFERENT NUMBER OF TREES

Typically, the number of trees is increased until the model performance stabilizes.

Intuition might suggest that more trees will lead to overfitting, although this is not the case. Both bagging and random forest algorithms appear to be somewhat **immune to overfitting** the training dataset given the **stochastic nature** of the learning algorithm.

The number of trees can be set via the "**n_estimators**" argument.

In this case, we can see that **performance is better** after about 50/100 trees.



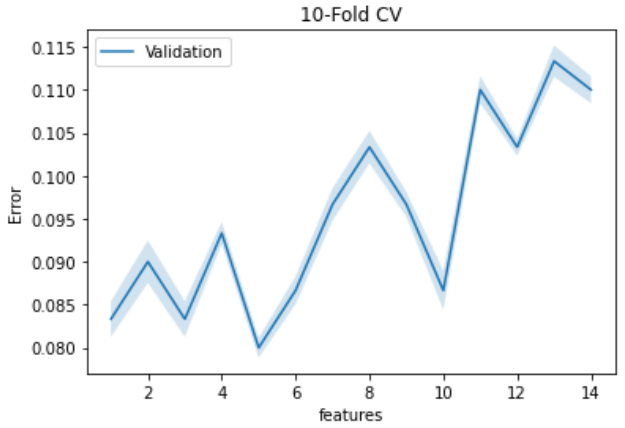
2. DIFFERENT NUMBER OF FEATURES PER NODE

The number of features that is randomly sampled for each split point.

It is set via the "**max_features**" argument. The default is the square root of the number of input features.
In this case, for our test dataset, this would be **sqrt(20) or about four features**.

Indeed, the results suggest that a value between 3 and 5 would be appropriate. Increasing the number of features per node increase also the misclassification.

```
The best features chosen with 10 folds is 5 with error 8.00 +- 0.12%
```



3. DIFFERENT SAMPLE'S SIZE

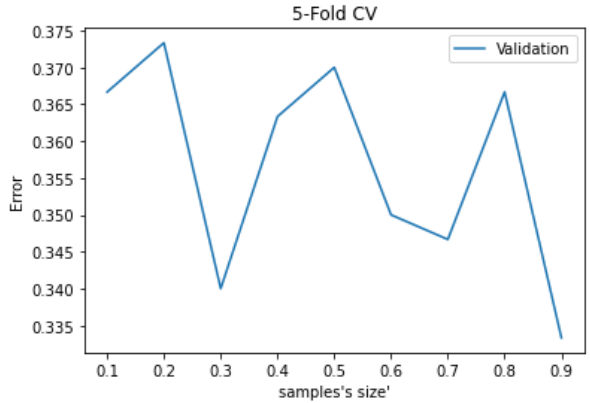
Each decision tree in the ensemble is fit on a bootstrap sample drawn from the training dataset.

The "**max_samples**" argument can be set to a float between 0 and 1 to control the percentage of the size of the training dataset to make the bootstrap sample used to train each decision tree.

A smaller sample size will make trees more different, and a larger sample size will make the trees more similar.

In this case, the results suggest that using a bootstrap sample size that is equal to the size of the training dataset achieves the best results on this dataset.
This is the default and it should probably be used in most cases.

```
The best samples's size chosen with 5 folds is 0.9 with error 33.33 +- 0.67%
```



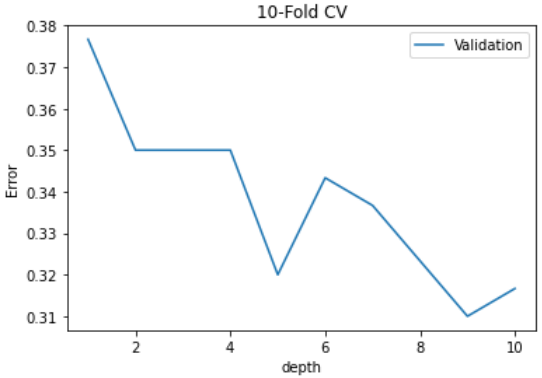
4. DIFFERENT TREE DEPTH

By default, trees are constructed to an arbitrary depth and are not pruned. In this case we fit the forest using trees with **different fixed depths**.

The maximum tree depth can be specified via the "**max_depth**" argument and is set to None (no maximum depth) by default.

In this case, we can see that **larger depth results in better model performance**, with the default of no maximum depth we achieving the best performance on this dataset.

```
The best depth chosen with 10 folds is 9 with error 31.00 +- 1.13%
```



THANKS!

Do you have any questions?