From different sources:
Hands on ML with Scikeat-learn, etc. Slides by Behnam Bahrak

Decision Tree & Random Forest

Introduction to Data Science
Spring 1403

Yadollah Yaghoobzadeh

Decision Trees

Build a Decision Tree

- Train a `DecisionTreeClassifier` on the iris dataset:

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris(as_frame=True)
X_iris = iris.data[["petal length (cm)", "petal width (cm)"]].values
y_iris = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X_iris, y_iris)
```

Visualize a Decision Tree

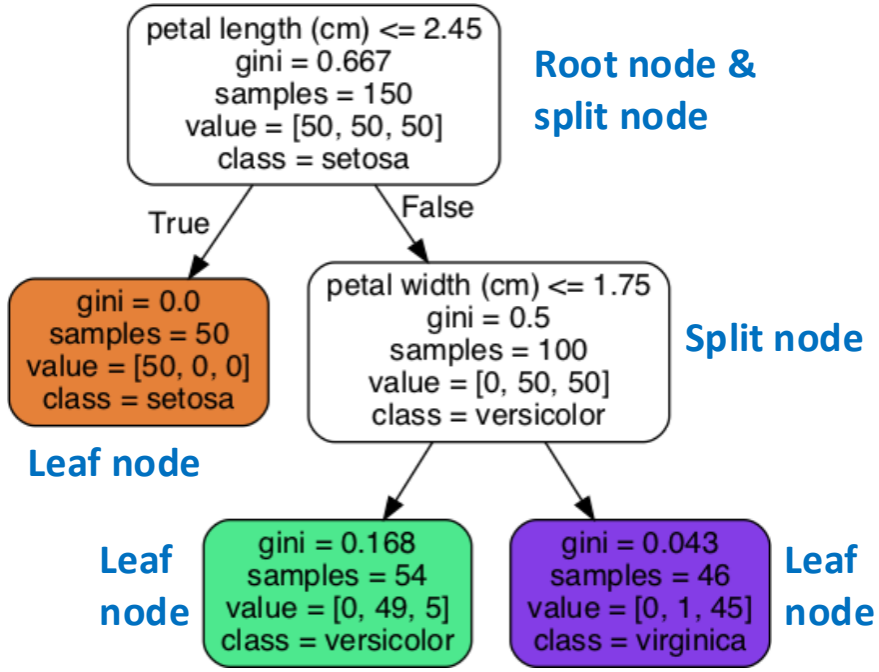
- Visualize the Decision Tree by using the `export_graphviz()` method to output a graph definition file called *iris_tree.dot*:

```
from sklearn.tree import export_graphviz

export_graphviz(
        tree_clf,
        out_file=str(IMAGES_PATH / "iris_tree.dot"),
        feature_names=["petal length (cm)", "petal width (cm)"],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )
```

```
from graphviz import Source

Source.from_file(IMAGES_PATH / "iris_tree.dot")
```

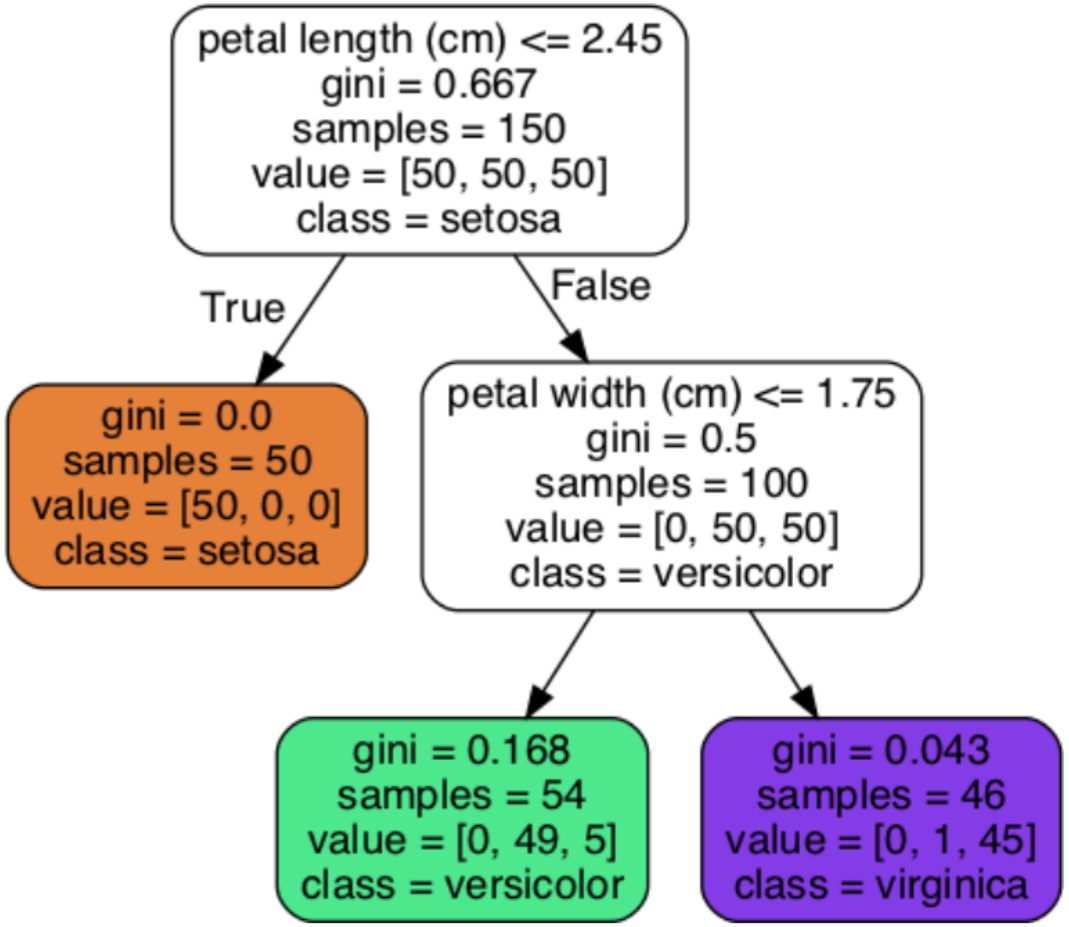


- Use the `dot` command-line tool to convert this *.dot* file to PNG:

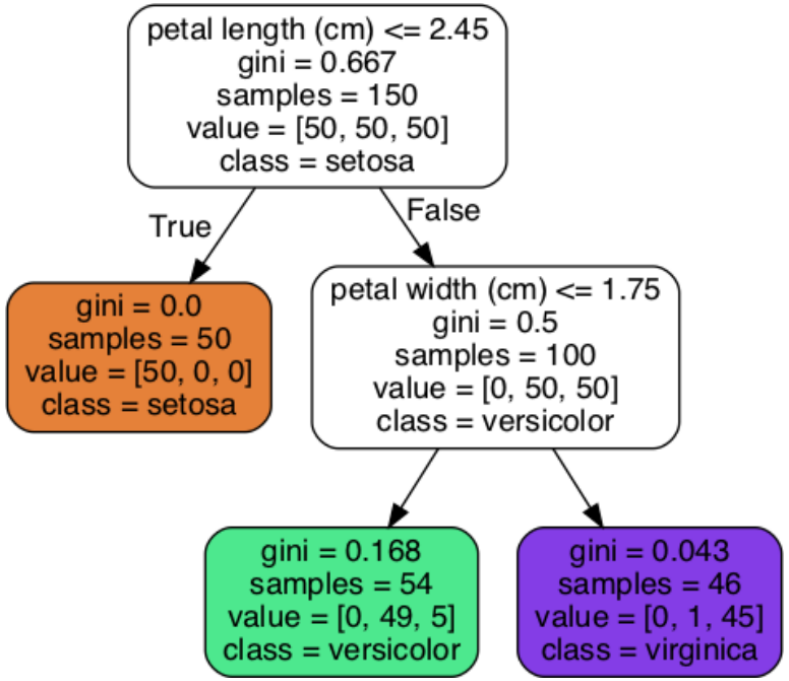
```
$ dot -Tpng iris_tree.dot -o iris_tree.png
```

Node Attributes in a Decision Tree

- `samples` attribute counts how many training instances it applies to.
- `value` attribute tells you how many training instances of each class this node applies to.
- `gini` attribute measures its *Gini impurity*: a node is "pure" (`gini=0`) if all training instances it applies to belong to the same class.



Gini Impurity



- Gini impurity G_i of the i-th node:

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$$

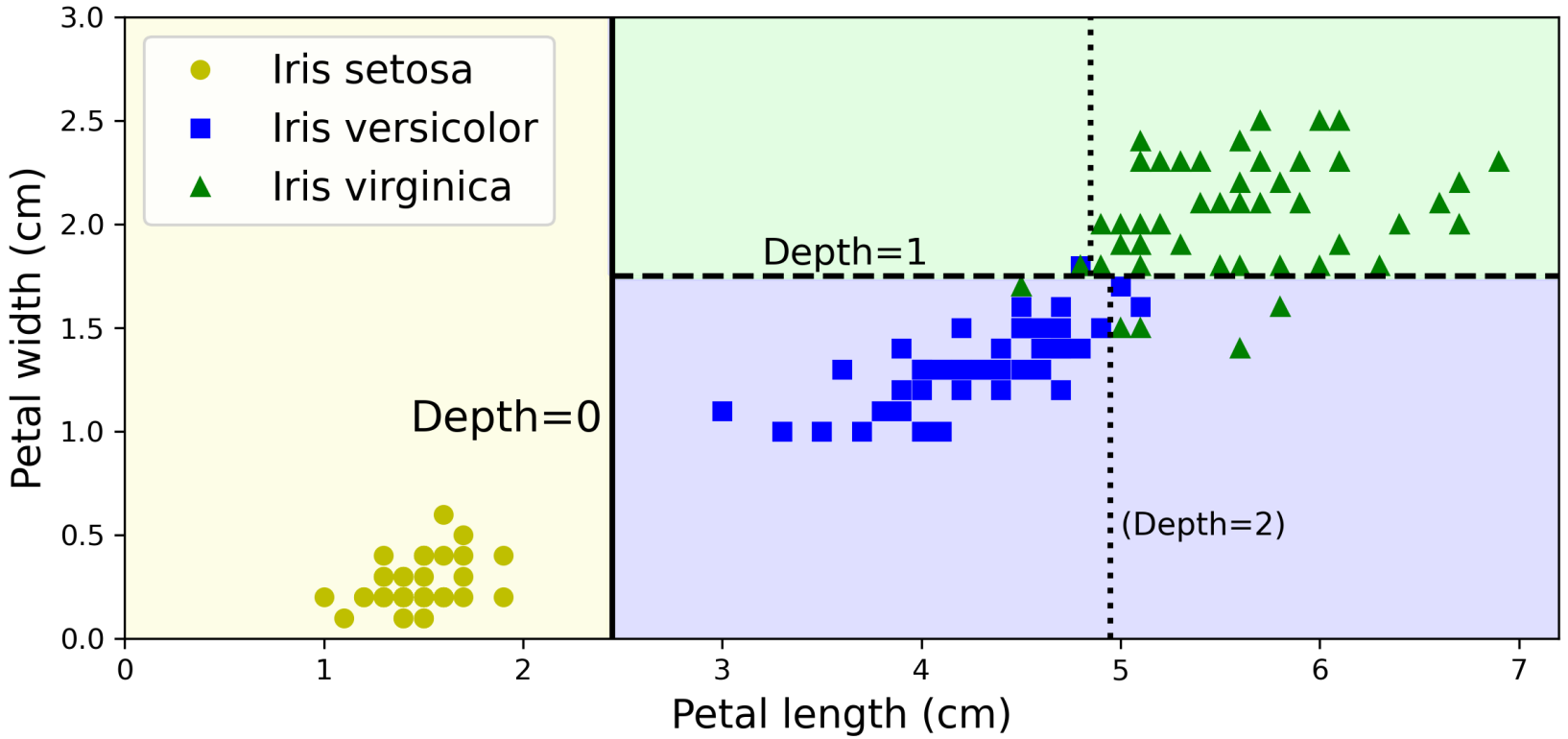
  - $p_{i,k}$ is the ratio of class k instances among the training instances in the i-th node.

  - Example (green node):

$$G = 1 - \left(\frac{0}{54}\right)^2 - \left(\frac{49}{54}\right)^2 - \left(\frac{5}{54}\right)^2 = 0.168$$

Decision Boundaries

- Decision tree's decision boundaries when `max_depth` is set to 3.



Model Interpretation

- Decision trees are intuitive, and their decisions are easy to interpret.
  - Such models are often called *white box models*.
- Neural networks are considered *black box models*: they make great predictions, but it is hard to explain in simple terms why the predictions were made.
- The field of *interpretable ML* aims at creating ML systems that can explain their decisions in a way humans can understand.

The COMPAS Race Bias

Estimating Class Probabilities

- A decision tree can estimate the probability that an instance belongs to a particular class k.
  - ➢ traverse the tree to find the leaf node for this instance, and then return the ratio of training instances of class k in this node.
- *Example*. A flower whose petals are 5 cm long and 1.5 cm wide: 0% for *Iris setosa* (0/54), 90.7% for *Iris versicolor* (49/54), and
  9.3% for *Iris virginica* (5/54).

```
tree_clf.predict_proba([[5, 1.5]]).round(3)
```

```
array([[0.   , 0.907, 0.093]])
```

```
tree_clf.predict([[5, 1.5]])
```

```
array([1])
```

The CART Training Algorithm

The CART Training Algorithm

- Scikit-Learn uses the *Classification and Regression Tree* (CART) algorithm to train decision trees.
  - The algorithm first splits the training set into two subsets using a single feature k and a threshold t_k (e.g., petal length ≤ 2.45 cm).
- How does it choose k and t_k?
  - It searches for the pair (k, t_k) that produces the purest subsets, weighted by their size by minimizing the cost function:

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

  - $G_{\text{left/right}}$ measures the gini impurity of the left/right subset
  - $m_{\text{left/right}}$ is the number of instances in the left/right subset

The CART Training Algorithm

- Once the CART algorithm has split the training set in two, it splits the subsets using the same logic, then the sub-subsets, and so on, recursively.
  - It stops recursing once it reaches the maximum depth (defined by the `max_depth` hyperparameter), or if it cannot find a split that will reduce impurity.
- A few other hyperparameters control additional stopping conditions: `min_samples_split`, `min_samples_leaf`, `min_weight_fraction_leaf`, and `max_leaf_nodes`.

Gini Impurity or Entropy?

- By default, the `DecisionTreeClassifier` class uses the Gini impurity measure, but you can select the *entropy* measure instead by setting the `criterion` hyperparameter to "`entropy`".
- In ML, entropy is frequently used as an impurity measure: a set's entropy is zero when it contains instances of only one class.

$$H_i = -\sum_{k=1}^{n} p_{i,k} \log_2 p_{i,k}$$

- Gini impurity is slightly faster to compute, while entropy tends to produce slightly more balanced trees.

Regularization

Regularization Hyperparameters

- Decision trees make few assumptions about the training data.
  - e.g. linear models assume that the data is linear.
- If left unconstrained, the tree structure will adapt itself to the training data, fitting it very closely—most likely overfitting it.
- Decision tree is a nonparametric model: the number of parameters is not determined prior to training.
  - A parametric model, e.g. a linear model, has a predetermined number of parameters, so its degree of freedom is limited, reducing the risk of overfitting.

Regularization Hyperparameters

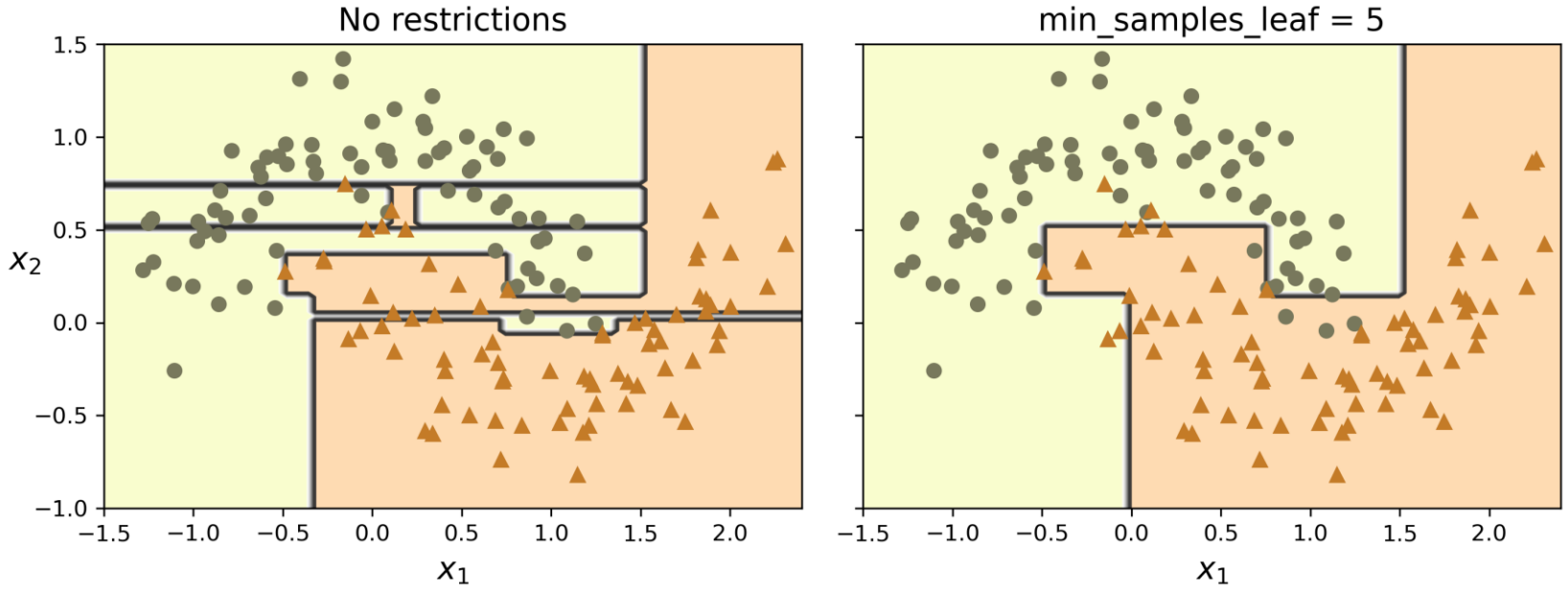
- `max_depth`: maximum depth of the decision tree
- `max_features`: maximum number of features that are evaluated for splitting at each node
- `max_leaf_nodes`: maximum number of leaf nodes
- `min_samples_split`: minimum number of samples a node must have before it can be split
- `min_samples_leaf`: minimum number of samples a leaf node must have to be created
- `min_weight_fraction_leaf`: same as `min_samples_leaf` but expressed as a fraction of the total number of weighted instances

Regularized Decision Tree

```python
from sklearn.datasets import make_moons

X_moons, y_moons = make_moons(n_samples=150, noise=0.2, random_state=42)

tree_clf1 = DecisionTreeClassifier(random_state=42)
tree_clf2 = DecisionTreeClassifier(min_samples_leaf=5, random_state=42)
tree_clf1.fit(X_moons, y_moons)
tree_clf2.fit(X_moons, y_moons)
```



Decision Trees have a High Variance

- The main issue with decision trees: high variance.
  - small changes to the hyperparameters or to the data may produce very different models.
- Since the training algorithm used by Scikit-Learn randomly selects the set of features to evaluate at each node, even retraining the same decision tree on the exact same data may produce a very different model.
- By averaging predictions over many trees, it's possible to reduce variance significantly.
  - Such an ensemble of trees is called a random forest.

Random Forests

- Democracy for decision trees!
- An example of **ensemble learning**
- For each decision tree do *bagging*:
  - Train on a *sample* of data points
  - Train on a s*ubset* of features.

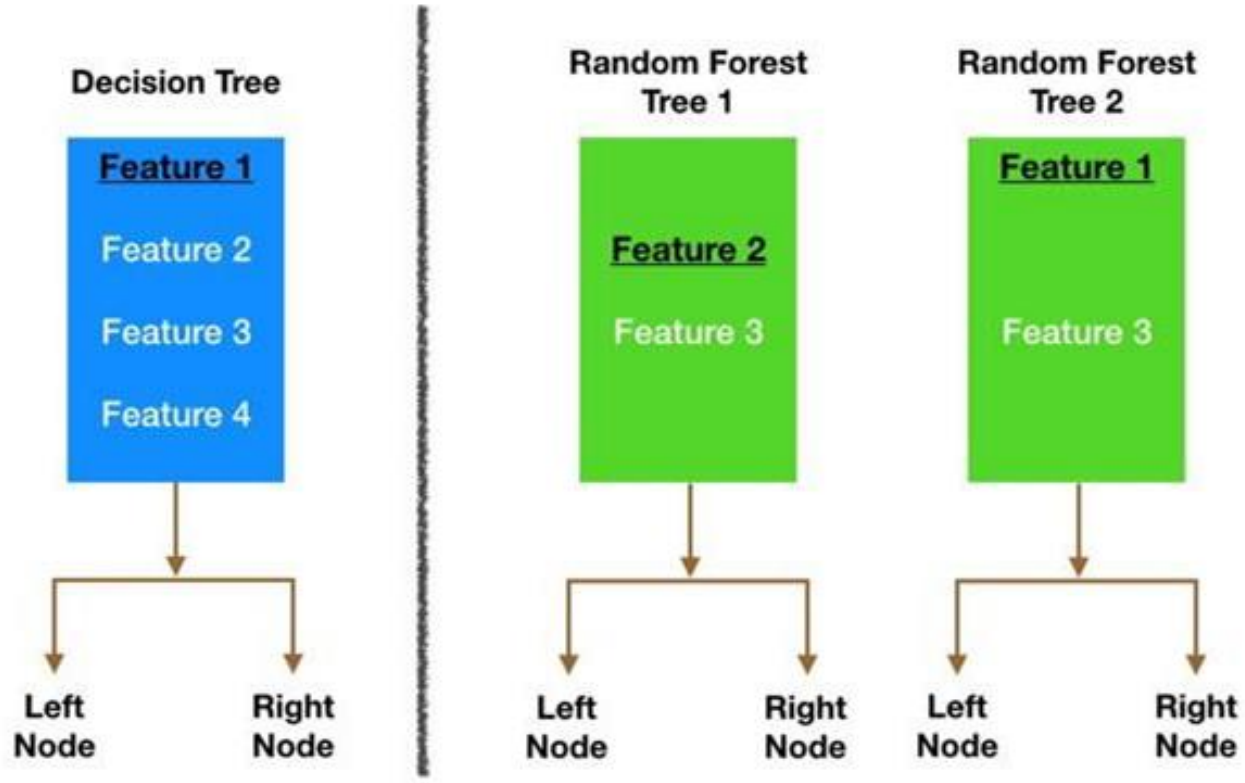


Image:https://towardsdatascience.com/understanding-random-forest-58381e0602d2

Random Forests

- For each decision tree do:
  - Train on a *sample* of data points
  - Train on a s*ubset* of features.
- The goal is to ***reduce*** correlation between different trees
  - It would be pointless to ask the same question from the same “decision maker” and expect a different answer/behavior!

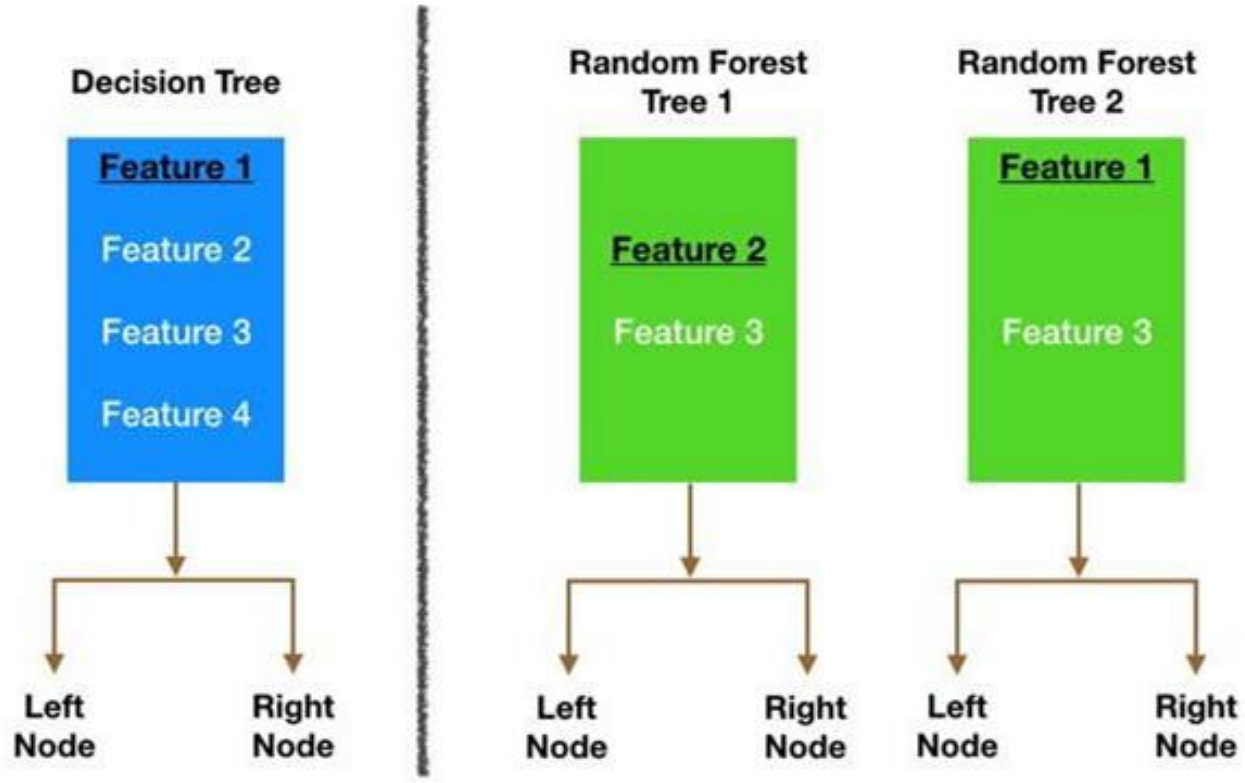
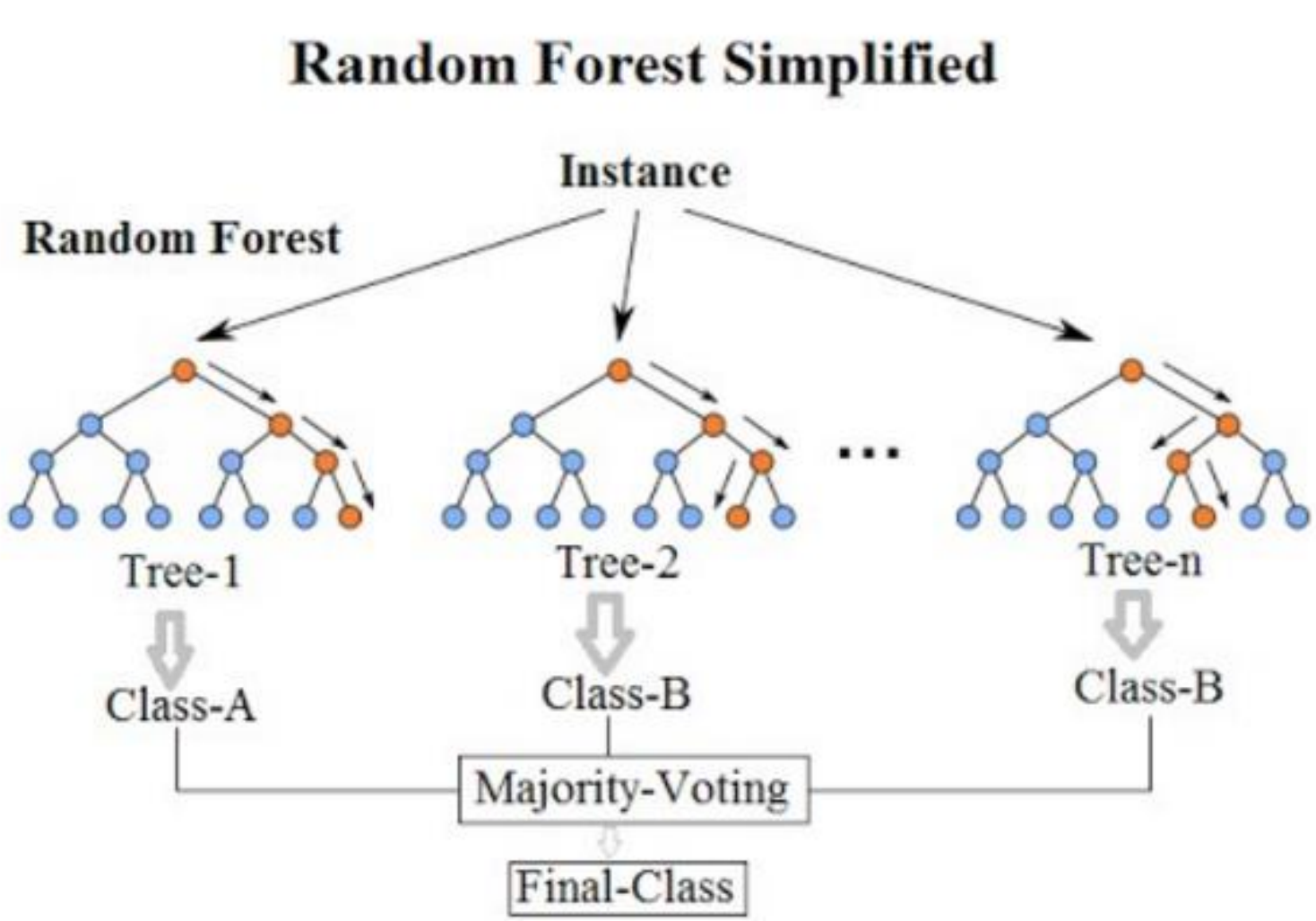


Image:https://towardsdatascience.com/understanding-random-forest-58381e0602d2

Random Forests

- For prediction, take majority voting, mean calculation or other aggregation mechanisms.



Ensemble learning

- Bagging
  - Train multiple classifiers and aggregate their predictions
  - Random forest is an example
- Boosting
  - Train classifiers sequentially, each trying to correct its predecessor.
  - Adaboost: A new predictor corrects its predecessor by paying attention to the training instances that the predecessor underfit.
    - New predictors focus more and more on the hard cases