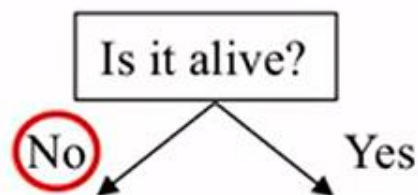
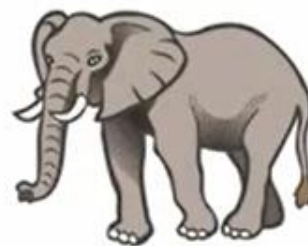


# Decision Trees and Random Forests

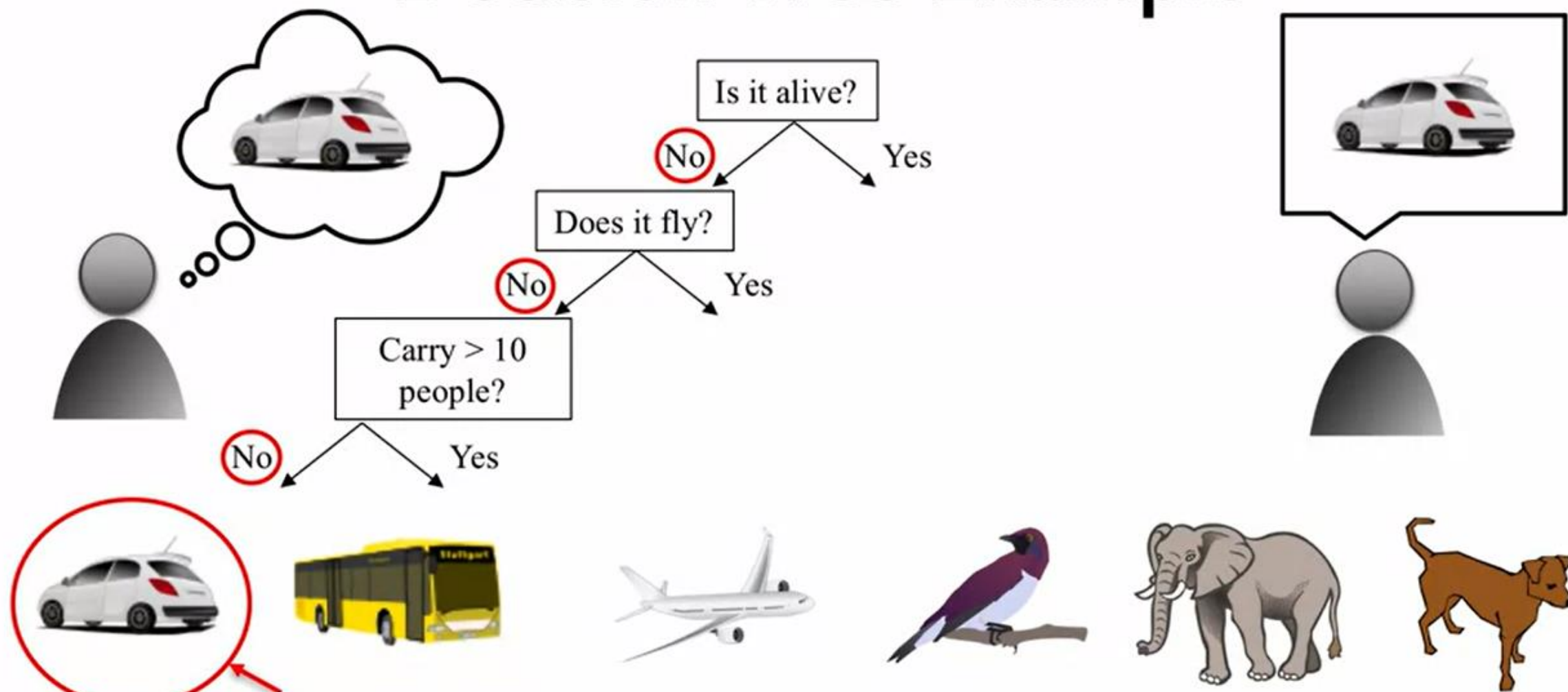
# Decision Tree Example



Objects with Alive == No



# Decision Tree Example



Objects with:  $\text{Alive} == \text{No}$  AND  $\text{Fly} == \text{No}$  AND  $\text{Carry} > 10 == \text{No}$

Root node

```
graph TD; A[Is it alive?] -- No --> B[Does it fly?]; A -- Yes --> C[Does it have a trunk?]; B -- No --> D[Dog]; B -- Yes --> E[Elephant]; C -- No --> D; C -- Yes --> E;
```

Alive == Yes  
Fly == No  
Trunk == No

Leaf node

Alive == Yes  
Fly == No  
Trunk == Yes

Alive == Yes  
Fly == No  
Trunk == No



Alive == Yes  
Fly == No  
Trunk == Yes



# The Iris Dataset



*Iris setosa*

*Iris versicolor*

*Iris virginica*

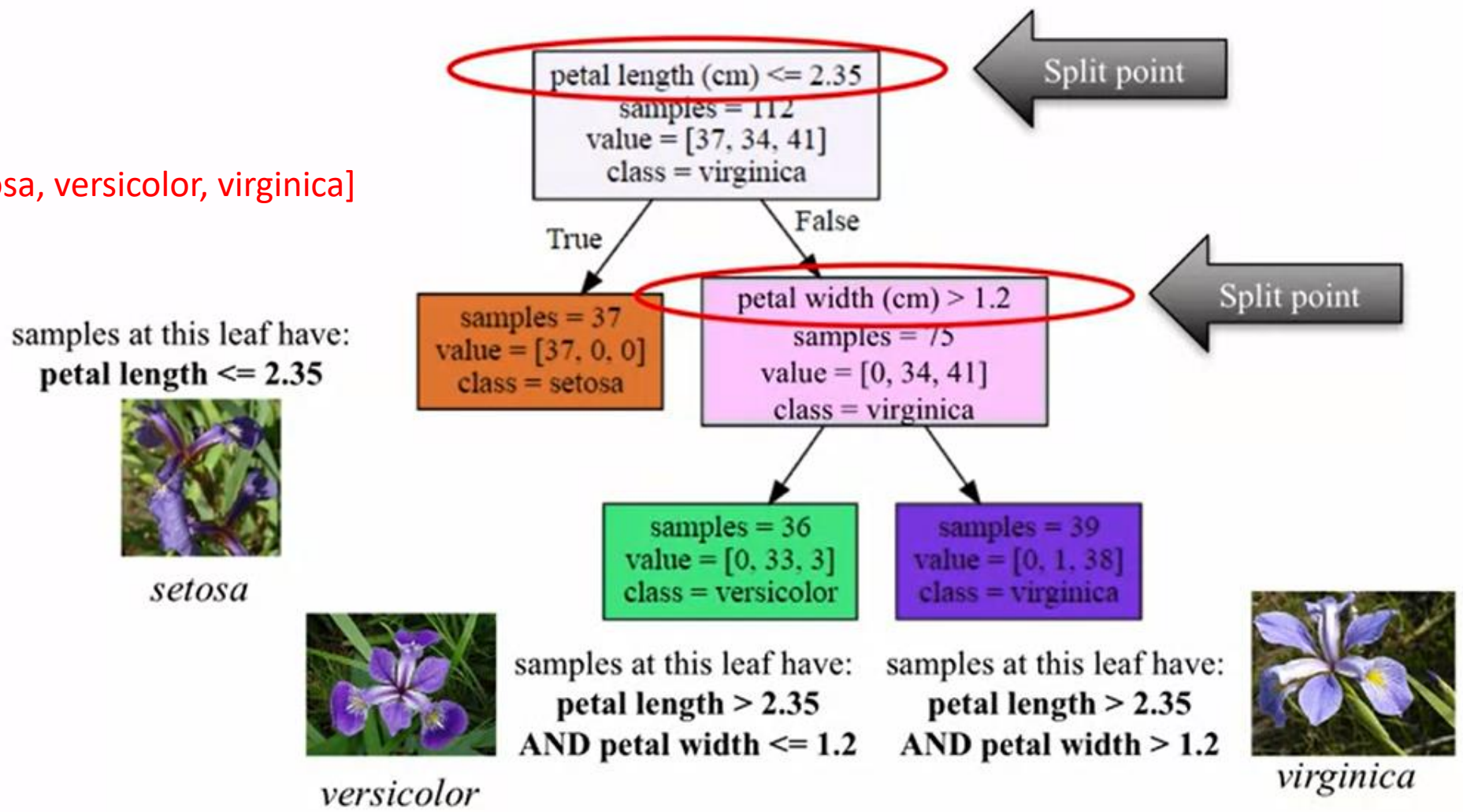
150 flowers  
3 species  
50 examples/species

# The Iris Dataset

- Each instance in the dataset represents one of three different species of Iris, a type of flower.
  - setosa
  - Versicolor
  - virginica
- There are four attributes or features for each instance, that represent measurements of different parts of the flower.
  - The sepal [*canak yapragi*] length in centimeters,
  - the sepal width in centimeters,
  - petal [*taç yaprağı*] length in centimeters
  - petal width in centimeters.
- The data set has measurements for 150 flowers with 50 examples of each species.
- The classification task is to predict which of the three species and instances given these measurements.
- The iris data sets features are continuous values. Unlike our simple example involving yes or no questions which were binary features of the object.
- So the rules to be learned will be of the form for example, "Is Petal length greater than 2.3 centimeters?"

# Decision Tree Splits

[Setosa, versicolor, virginica]

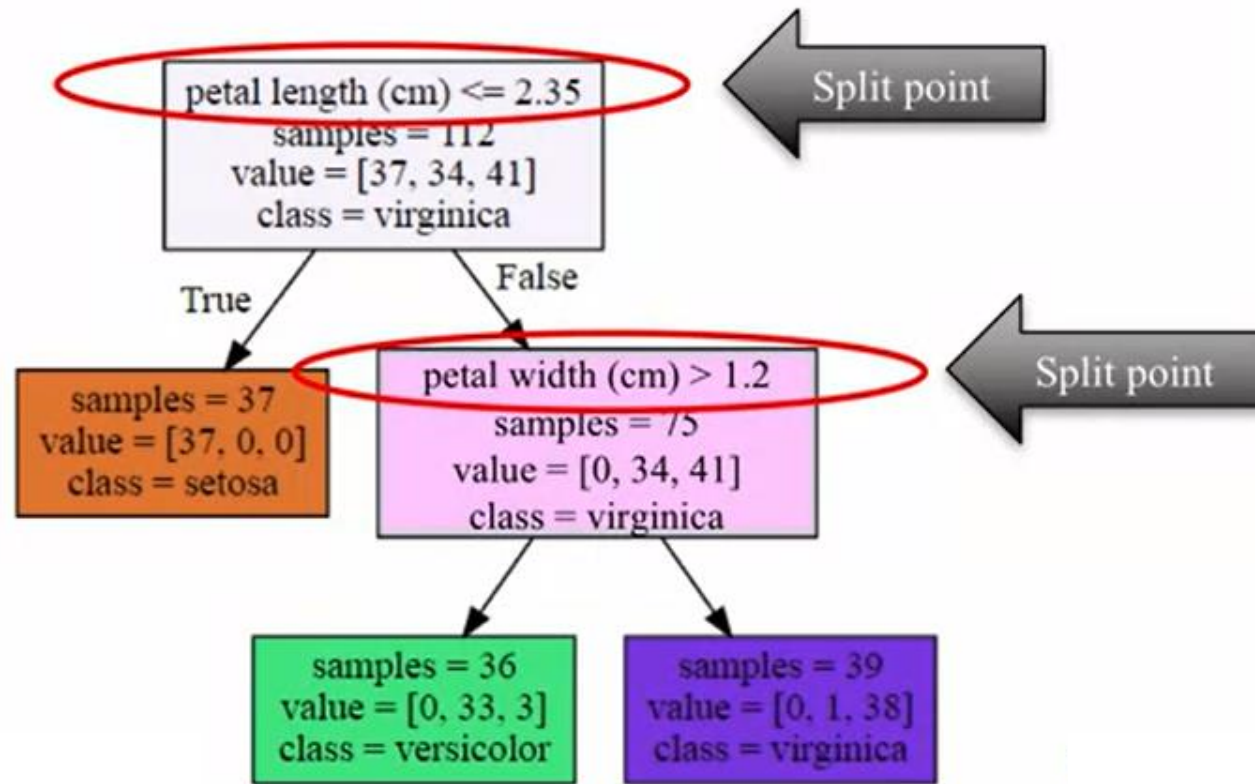


# Decision Tree Splits

The *value* list gives the number of samples of each class that end up at this leaf node during training.

The iris dataset has 3 classes, so there are three counts.

This leaf has 37 setosa samples, zero versicolor, and zero virginica samples.



This leaf has 0 setosa, 33 versicolor, and 3 virginica samples.

This leaf has 0 setosa, 1 versicolor, and 38 virginica samples.

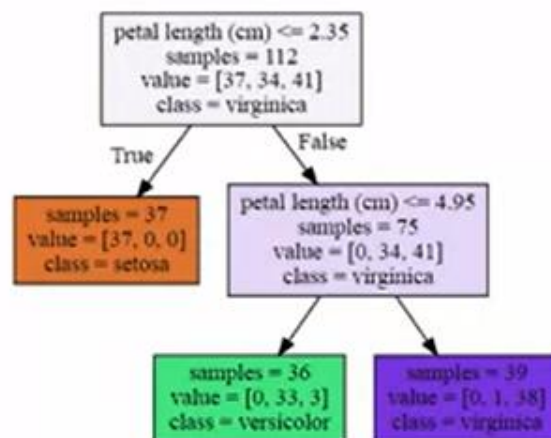


# Informativeness of Splits

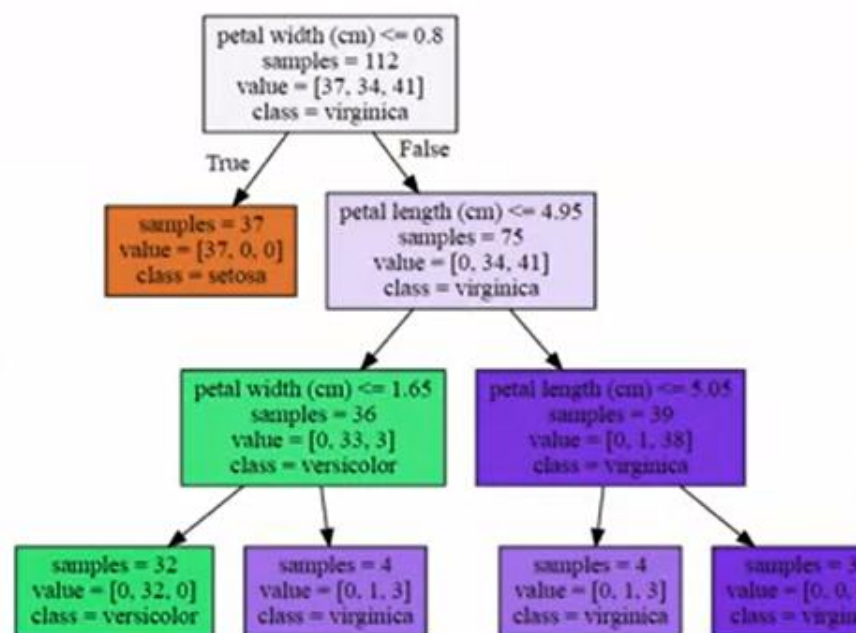
- So to build the decision tree, the decision tree building algorithm starts by finding the feature that leads to **the most informative split**.
- For any given split of the data on a particular feature value, even for the best split it's likely that some examples will still be incorrectly classified or need further splitting.
- In the iris data set, if we look at all the flowers with petal length greater than 2.35 centimeters for example, using that split leaves a pool of instances that are a combination of virginica and versicolor that still need to be distinguished further

# Controlling the Model Complexity of Decision Trees

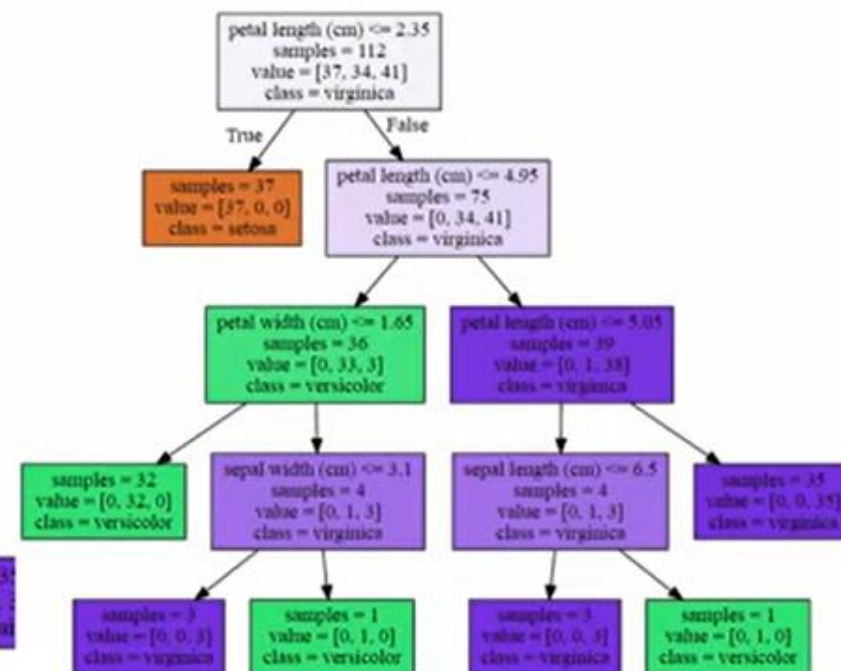
`max_depth = 2`





`max_depth = 3`



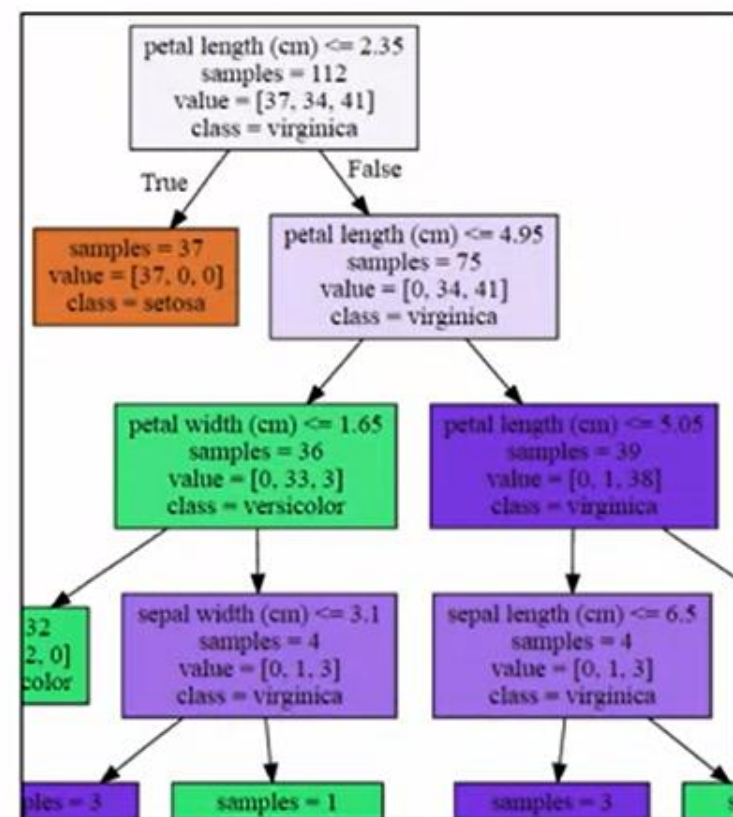
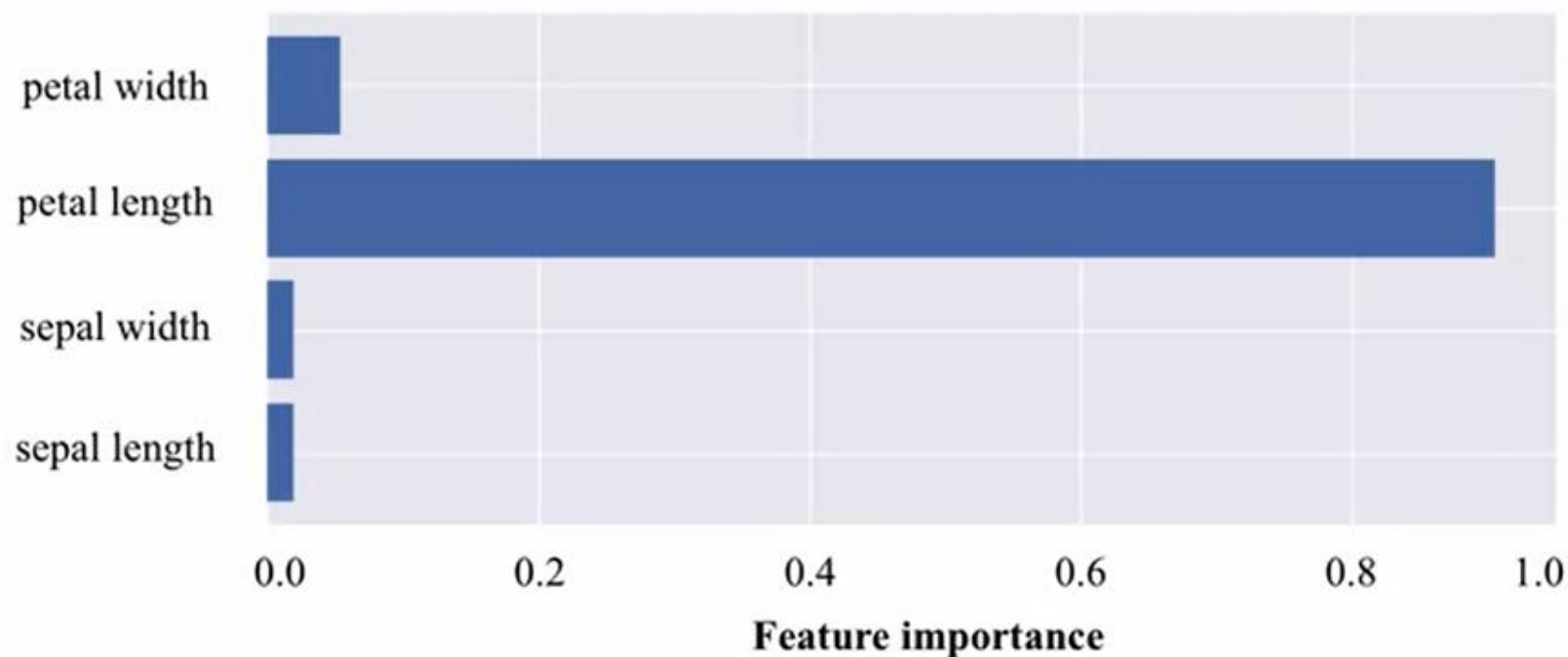
`max_depth = 4`



## Feature Importance: How important is a feature to overall prediction accuracy?

- A number between 0 and 1 assigned to each feature.
- Feature importance of 0  the feature was not used in prediction.
- Feature importance of 1  the feature predicts the target perfectly.
- All feature importances are normalized to sum to 1.

# Feature Importance Chart



Decision tree



# Decision Trees: Pros and Cons

## Pros:

- Easily visualized and interpreted.
- No feature normalization or scaling typically needed.
- Work well with datasets using a mixture of feature types (continuous, categorical, binary)

## Cons:

- Even after tuning, decision trees can often still overfit.
- Usually need an ensemble of trees for better generalization performance.

## Decision Trees: DecisionTreeClassifier Key Parameters

- `max_depth`: controls maximum depth (number of split points). Most common way to reduce tree complexity and overfitting.
- `min_samples_leaf`: threshold for the minimum # of data instances a leaf can have to avoid further splitting.
- `max_leaf_nodes`: limits total number of leaves in the tree.
- In practice, adjusting only one of these (e.g. `max_depth`) is enough to reduce overfitting.

# Random Forests

# Ensemble Learning

- A widely used and effective method in machine learning involves creating learning models known as ensembles (.i.e. groups).
- An ensemble takes multiple individual learning models and combines them to produce an aggregate model that is more powerful than any of its individual learning models alone.



# Ensemble Learning - Why are ensembles effective?

- Well, one reason is that if we have different learning models, although each of them might perform well individually, they'll tend to make different kinds of mistakes on the data set
- Or, each individual model might overfit to a different part of the data.
- By combining different individual models into an ensemble, we can **average** out their individual mistakes to reduce the risk of overfitting while maintaining strong prediction performance.
- Random forests are an **example of the ensemble idea** applied to decision trees.
- Random forests are widely used in practice and achieve very good results on a wide variety of problems.

# Random Forests

- An ensemble of trees, not just one tree.
- Widely used, very good results on many problems.
- `sklearn.ensemble` module:
  - *Classification: `RandomForestClassifier`*
  - *Regression: `RandomForestRegressor`*
- One decision tree → Prone to overfitting.
- Many decision trees → More stable, better generalization
- Ensemble of trees should be diverse: introduce random variation into tree-building.

The idea is that each of the individual trees in a random forest should do reasonably well at predicting the target values in the training set **but** should also be constructed to be **different** in some way from the other trees in the forest

# Random Forests

- This difference is accomplished by introducing random variation into the process of building each decision tree.
- This random variation during tree building happens in two ways.
  - First, the data used to build each tree is selected randomly and
  - Second the features chosen in each split tests are also randomly selected
- To create a random forest model you first decide on **how many trees** to build.
  - This is set using the **n\_estimated** parameter.
- Each tree were built from a different random sample of the data called the **bootstrap sample**.

# Random Forests

- Bootstrap samples are commonly used in statistics and machine learning.
- If your training set has  $N$  instances or samples in total, a bootstrap sample of size  $N$  is created by just repeatedly picking one of the  $N$  dataset rows at **random with replacement**, that is, allowing for the possibility of picking the same row again at each selection.
- You repeat this random selection process  $N$  times.
- The resulting bootstrap sample has  $N$  rows just like the original training set but with possibly some rows from the original dataset missing and others occurring multiple times just due to the nature of the random selection with replacement



# Random Forest Process: Bootstrap Samples

Bootstrap sample 1

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

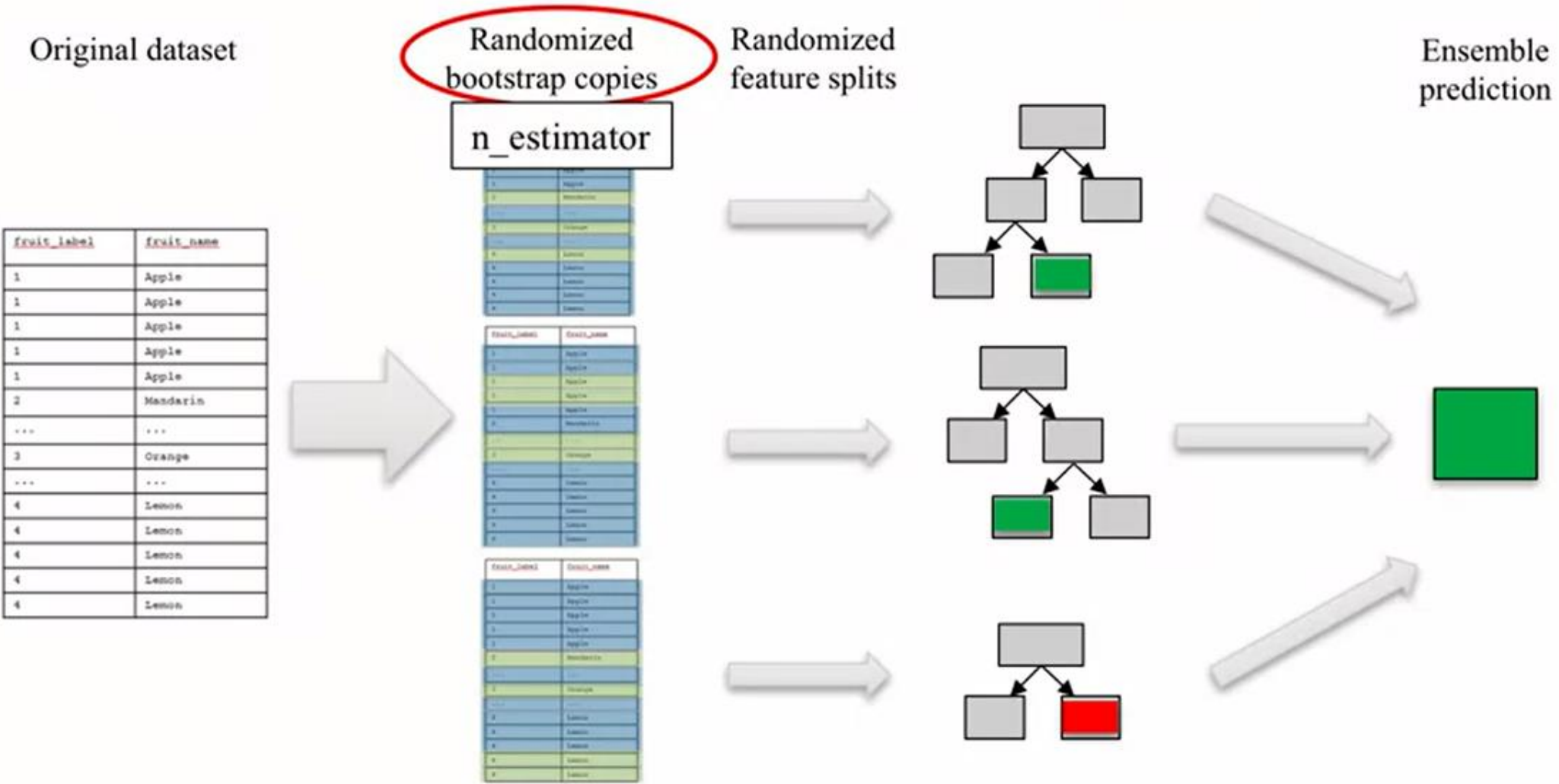
Bootstrap sample 2

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

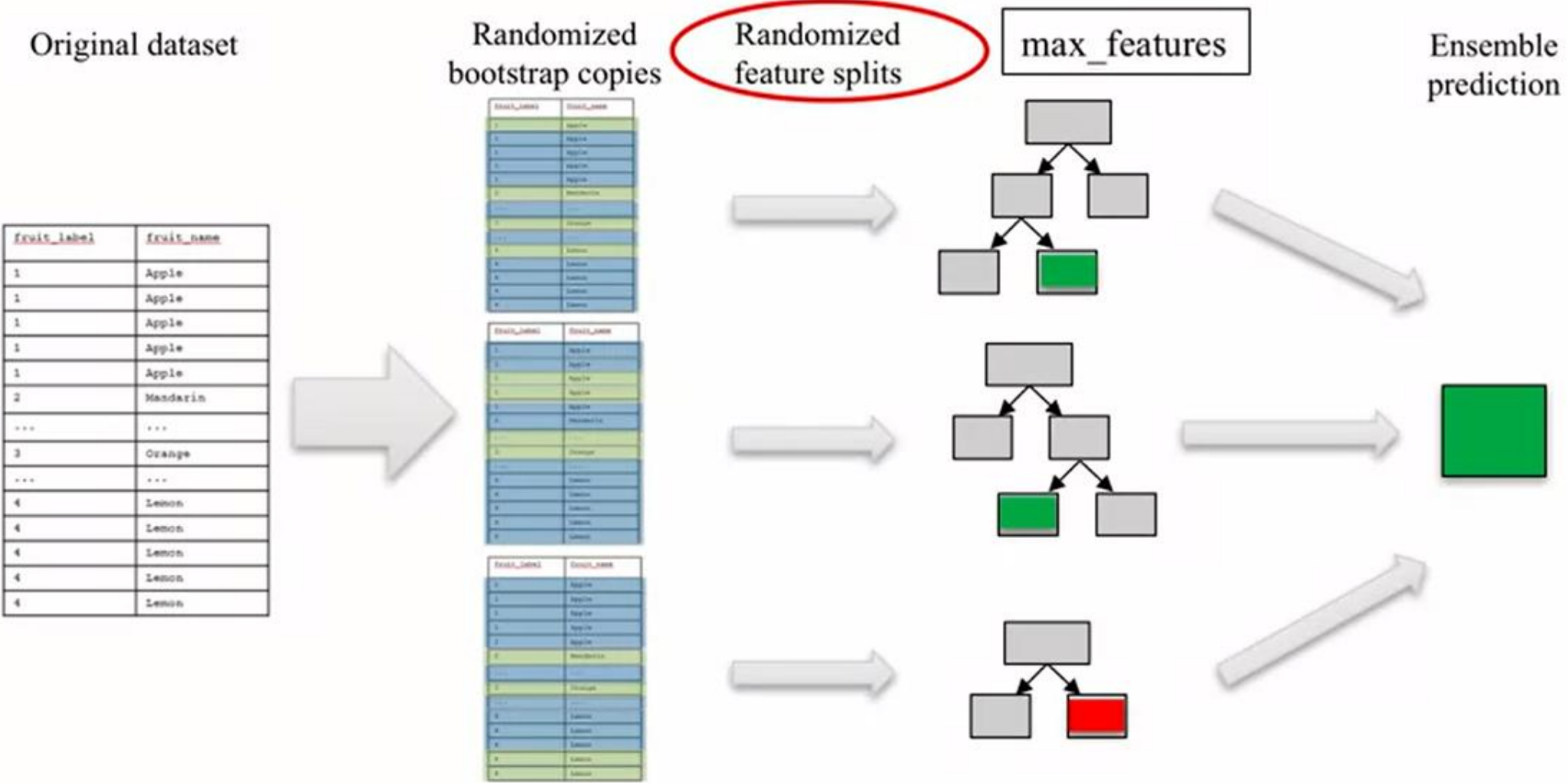
Bootstrap sample 3

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

# Random Forest Process



# Random Forest Process



# Random Forest `max_features` Parameter

- Learning is quite sensitive to `max_features`.
- Setting `max_features = 1` leads to forests with diverse, more complex trees.
- Setting `max_features = <close to number of features>` will lead to similar forests with simpler trees.



# max\_features Parameter

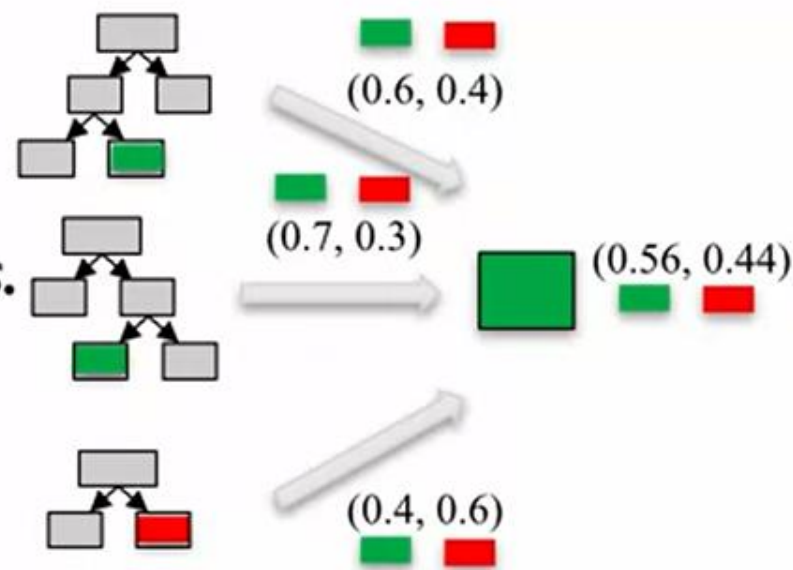
- Max\_feature: These are the maximum number of features Random Forest is allowed to try in individual tree.
- Learning is quite sensitive to max\_features.
  - Setting max features = 1 leads to forests with diverse, more complex trees.
  - Setting max\_features = <close to number of features> will lead to similar forests with simpler trees.

# Prediction Using Random Forests

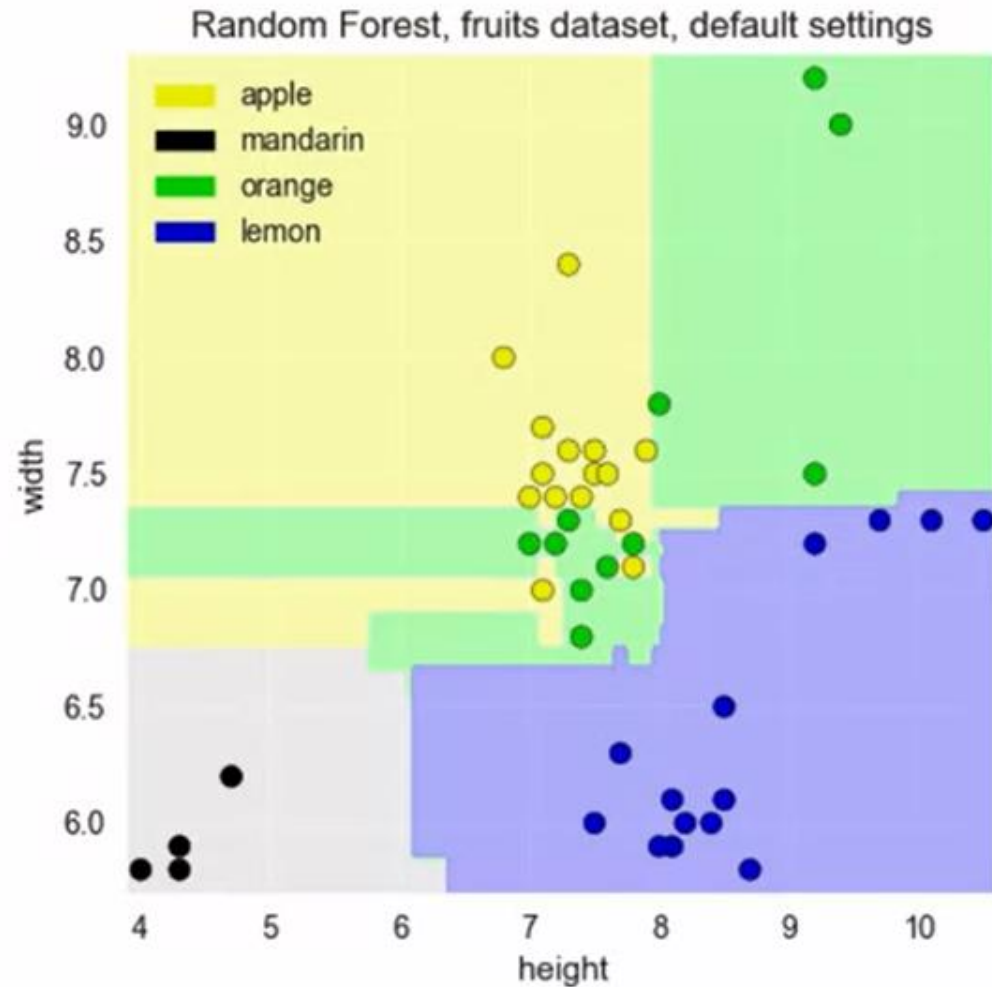
1. Make a prediction for every tree in the forest.

2. Combine individual predictions

- *Regression: mean of individual tree predictions.*
- *Classification:*
  - *Each tree gives probability for each class.*
  - *Probabilities averaged across trees.*
  - *Predict the class with highest probability.*



# Random Forest: Fruit Dataset



# Random Forest: Pros and Cons

## Pros:

- Widely used, excellent prediction performance on many problems.
- Doesn't require careful normalization of features or extensive parameter tuning.
- Like decision trees, handles a mixture of feature types.
- Easily parallelized across multiple CPUs.

## Cons:

- The resulting models are often difficult for humans to interpret.
- Like decision trees, random forests may not be a good choice for very high-dimensional tasks (e.g. text classifiers) compared to fast, accurate linear models.



## Random Forests: RandomForestClassifier

### Key Parameters

- **n\_estimators**: number of trees to use in ensemble (default: 10).
  - *Should be larger for larger datasets to reduce overfitting (but uses more computation).*
- **max\_features**: has a strong effect on performance. Influences the diversity of trees in the forest.
  - *Default works well in practice, but adjusting may lead to some further gains.*
- **max\_depth**: controls the depth of each tree (default: None. Splits until all leaves are pure).
- **n\_jobs**: How many cores to use in parallel during training.
- Choose a fixed setting for the **random\_state** parameter if you need reproducible results.