UNIVERSITY of York
Europe Campus
CITY College

Supervised Learning: Classification

Support Vector Machines, Decision Trees, Ensemble Methods (Random Forests)

Dr Ioanna Stamatopoulou

All material in these lecture notes is based on our textbook:
Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*, 3rd ed., O'Reilly, 2022

Support Vector Machines

Support Vector Machines

- Used for:
  - Linear and non-linear Regression
  - Linear and non-linear Classification
  - Novelty Detection (outside the scope of this module)
- Perform **best** with **small- to medium-sized datasets**
  - i.e. datasets with hundreds to thousands of instances
  - They do not scale very well to (very) large datasets
- Perform **best** with **scaled feature values**
  - see slide 7

```
**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
```

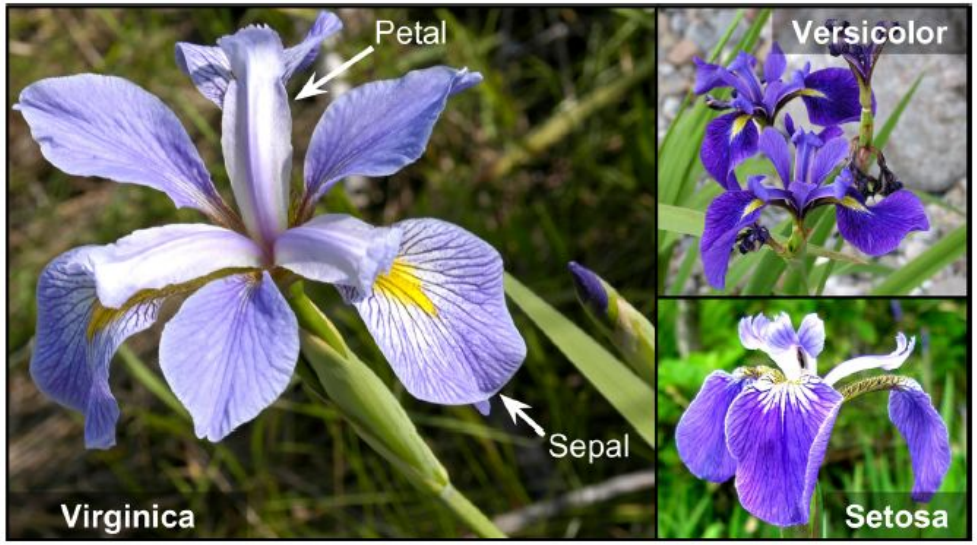
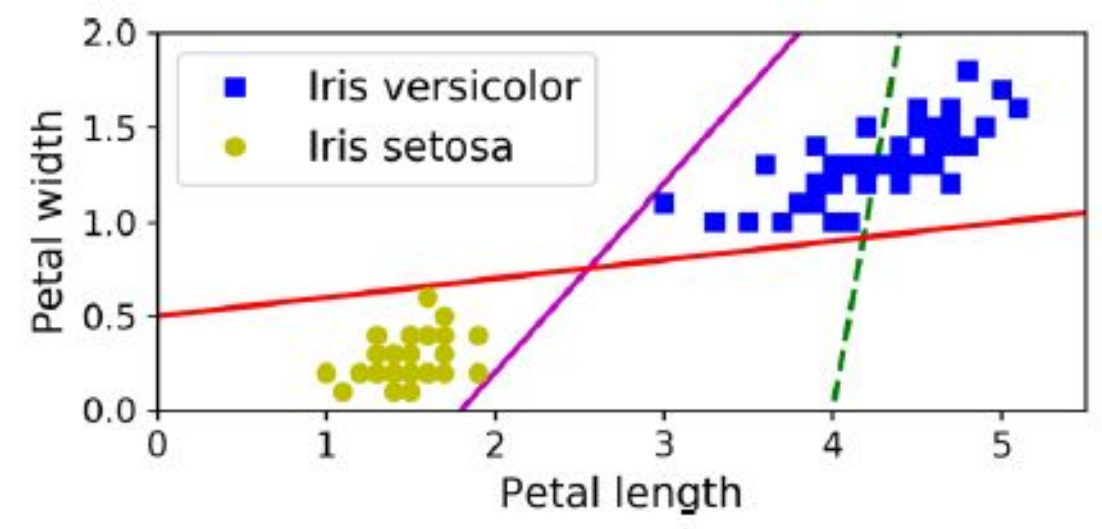


Figure 4-22. Flowers of three iris plant species[14]

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2

The iris dataset

Examples of bad linear classification models



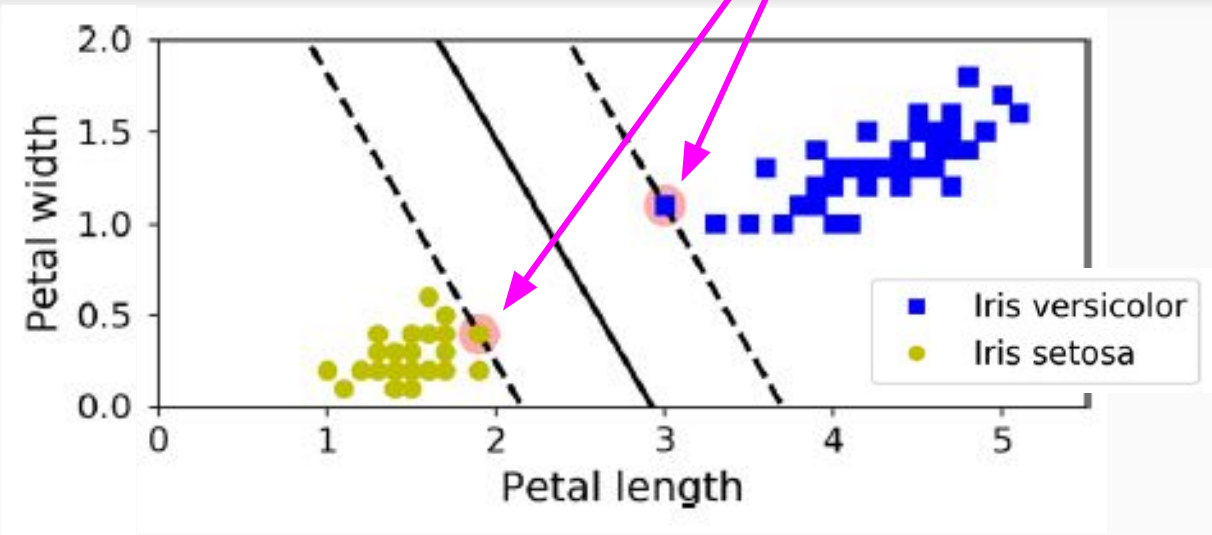
- - - : does not even separate the classes

— and — : their decision boundaries are too close to the instances

⇒ — and — will not generalise well

SVM
Linear Classification

Support Vectors:
The **instances** located on the edge of the street: they **support** (determine) the **boundaries**



Large margin classification:

— : the model separates the classes

while staying as far away from the instances as possible - - -

The SVM fits the **widest possible street between the classes**

SVM Linear Classification

Why is scaling important?

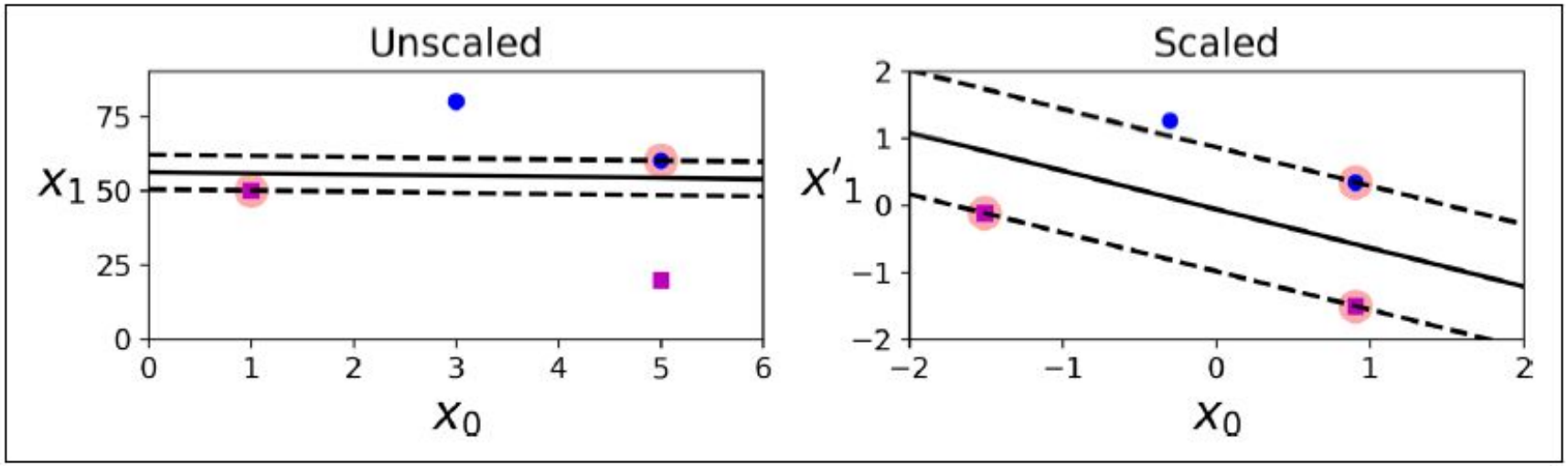


Figure 5-2. Sensitivity to feature scales

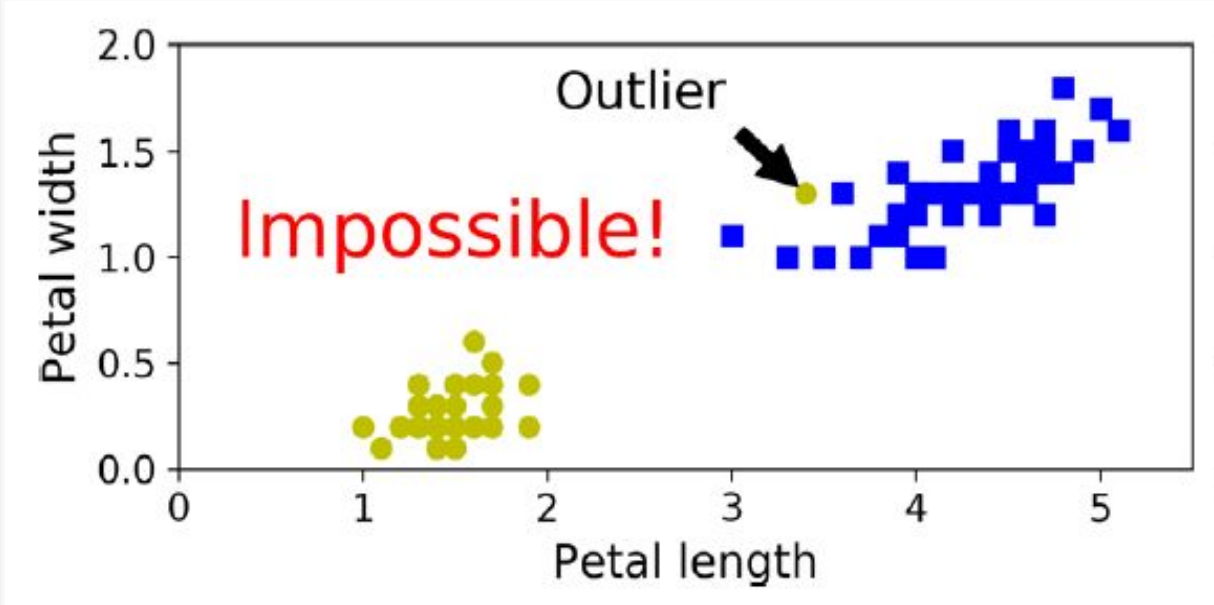
SVM Linear Classification: **Hard Margin** Classification

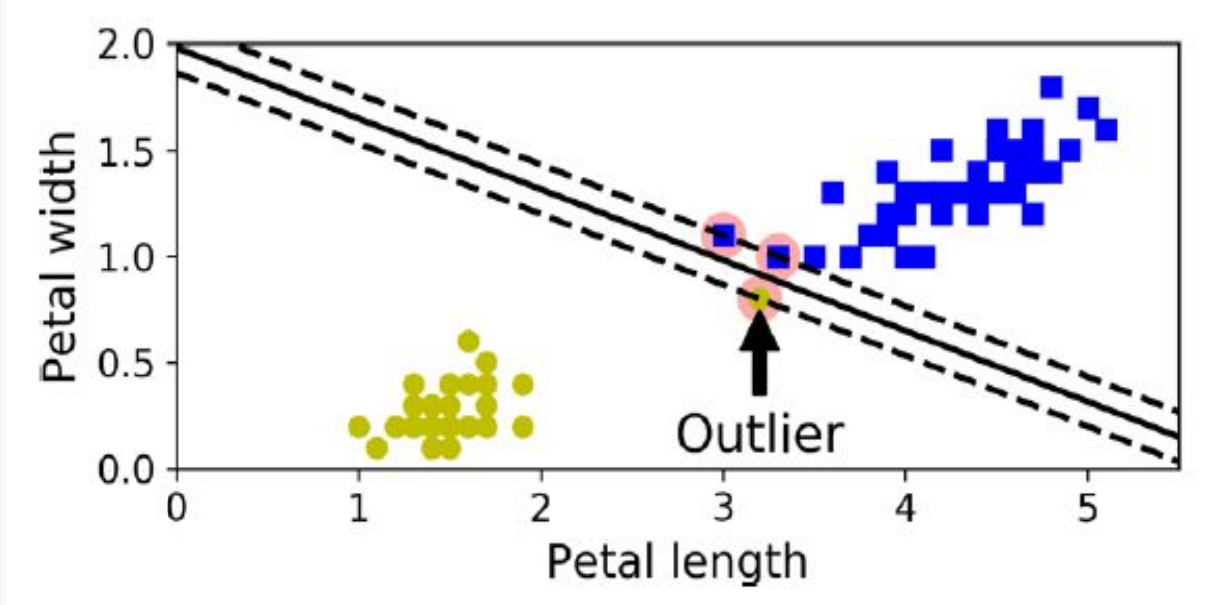
Imposes that all instances must be:

- Off the street
- On the correct side

Two problems:

- Data must indeed be linearly separable
- It is sensitive to outliers





SVM Linear Classification: **Soft Margin** Classification

The C hyperparameter

If the SVM model is overfitting, regularise it by reducing **C**

- **Soft Margin** Classification is about finding a balance between:
  - Keeping the street as wide as possible
  - Limiting the margin violations
- The **C hyperparameter** can control this:
  - Reduce C:
    ⇒ wider street
    ⇒ more support vectors (instances on the margin)
    ⇒ more margin violations
    ⇒ less risk of overfitting
  - Reduce C too much ⇒ risk of underfitting

SVM: **Soft Margin** Classification

The C hyperparameter (cont'd)

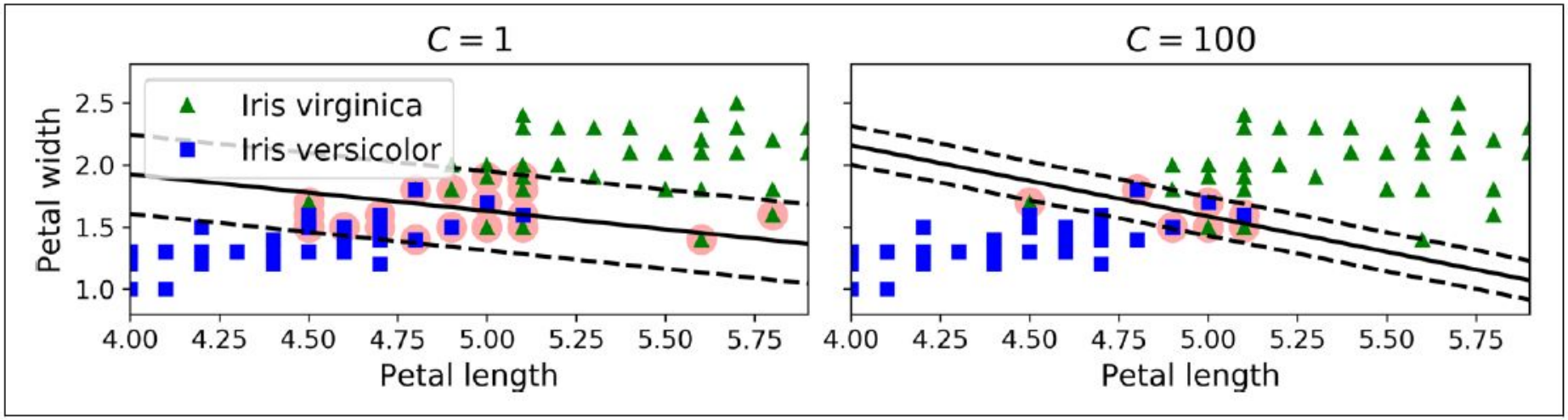


Figure 5-4. Large margin (left) versus fewer margin violations (right)

SVM Soft Margin Classification

The iris dataset

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

iris = load_iris(as_frame=True)
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = (iris.target == 2)  # Iris virginica

svm_clf = make_pipeline(StandardScaler(),
                        LinearSVC(C=1, dual=True, random_state=42))
svm_clf.fit(X, y)
```

```
X_new = [[5.5, 1.7], [5.0, 1.5]]
svm_clf.predict(X_new)
```

```
array([ True, False])
```

SVM Nonlinear Classification

Various ways to perform nonlinear classification using SVMs

- Using the `LinearSVC` class, you have to turn your non-linearly separable data into linearly separable by:
  - adding Polynomial Features
  - adding Similarity Features
- Using the `SVC` class, you may use the **kernel trick**:
  - Polynomial Kernel: gives you the same result as if you had added many polynomial features
  - Gaussian RBF Kernel: gives you the same result as if you had added many similarity features

SVM Regression

- SVMs may also be used for **regression**
  - *Outside the scope of this module*

Support Vector Machines: scikit classes

LinearSVC for Linear **C**lassification
https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

SVC for **C**lassification
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

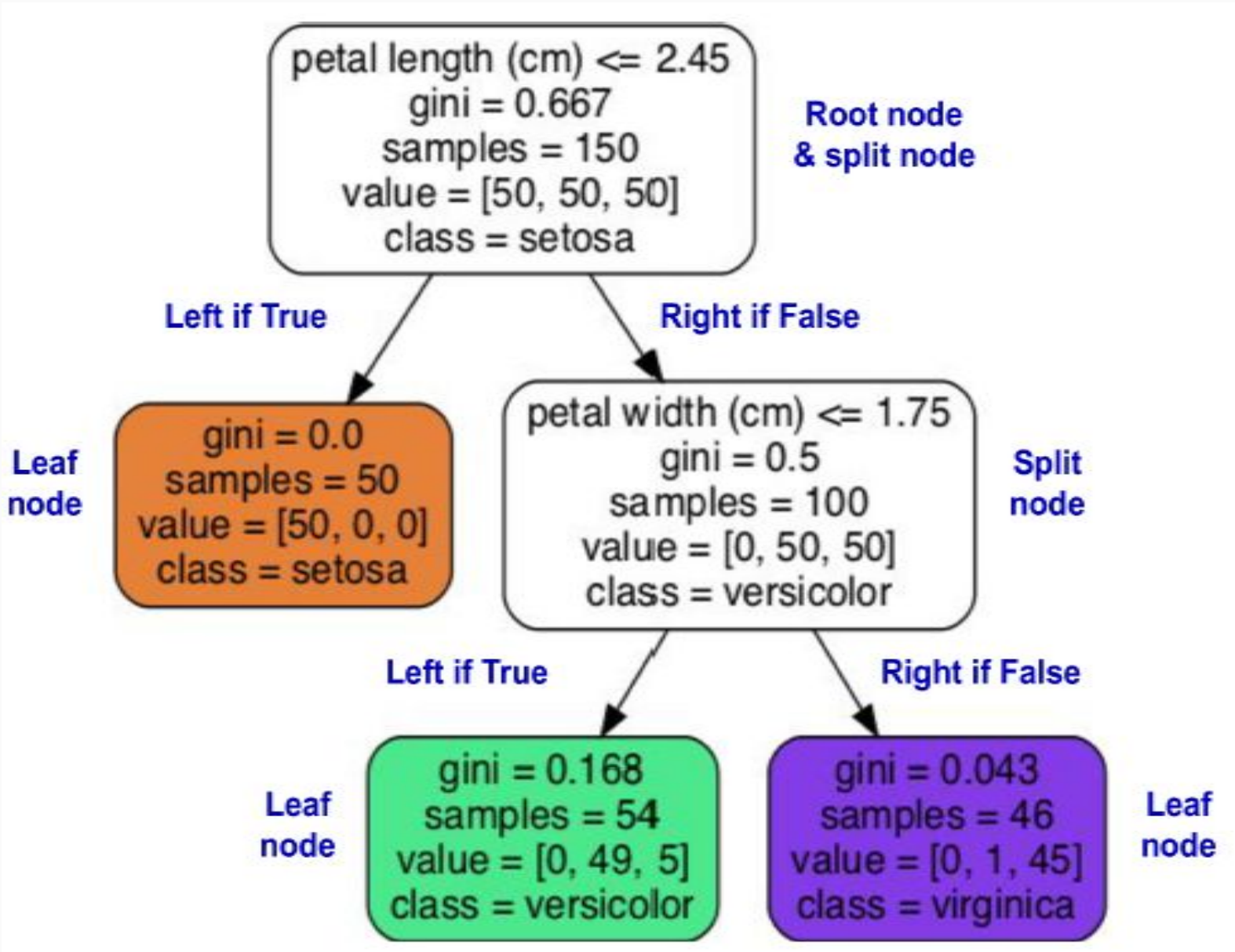
LinearSVR the equivalent of `LinearSVC` but for **R**egression
https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVR.html

SVR the equivalent of `SVC` but for **R**egression
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

Decision Trees

Decision Trees

- Used for:
  - Regression
  - Classification
  - Multioutput tasks (outside the scope of this module)
- They are also the fundamental components of **Random Forests**
- The typically do **not require any data preparation**, such as feature scaling



A possible **Classification Decision Tree** for the iris dataset

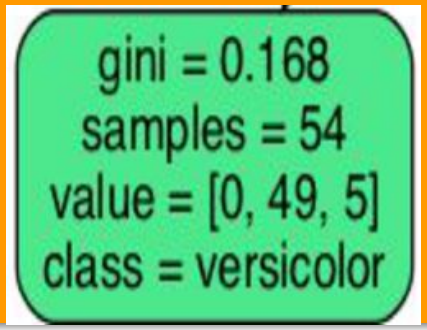
Predictions using a Decision Tree
Types of Nodes

Decision Trees contain two types of nodes:

- **Split nodes**
  - Ask a question
  - Depending on the answer to the question, you are directed to a child node on the tree level below
- **Leaf nodes**
  - Do not have children nodes
  - They predict the class

Gini impurity of the node below:
$1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168$

Decision Trees: Node Attributes



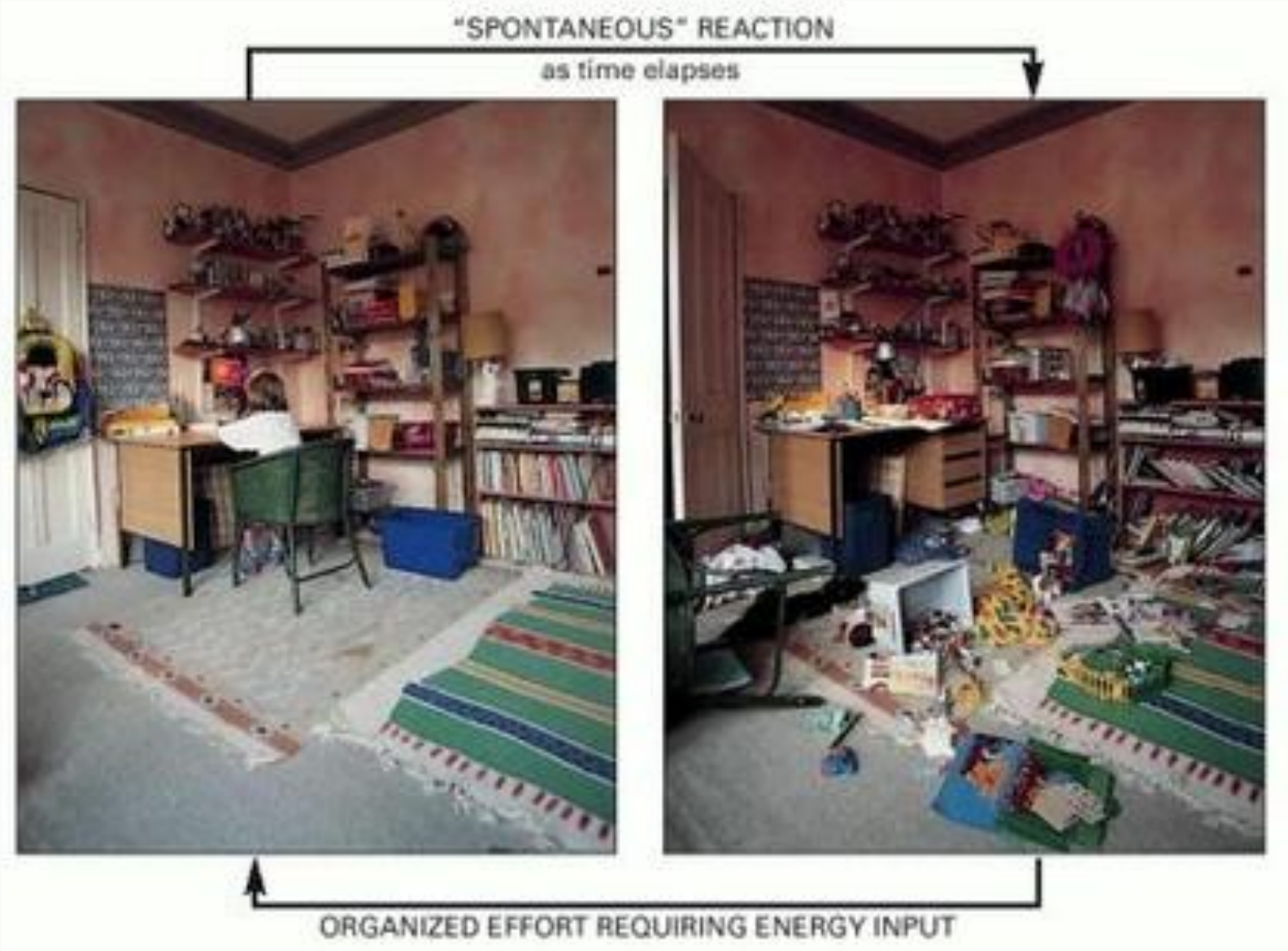
- Samples
  - The number of instances the node applies to
- Value
  - The number of instances of each class the node applied to
- Gini impurity
  - A node is **pure** `(gini = 0)` when all instances it applies to belong to the same class
- Class Probability
  - The ratio of instances of the class over the total instances that a node applies to
  - The class predicted by the decision tree is the class with the higher ratio

Types of Decision Trees

- The **CART** algorithm creates **Binary Decision Trees**, i.e. split nodes have exactly two children; the question is a condition that may be true or false for any new instance
- The **ID3** algorithm produces trees whose **split nodes can have more than two children**
  - Uses **Entropy**

Entropy

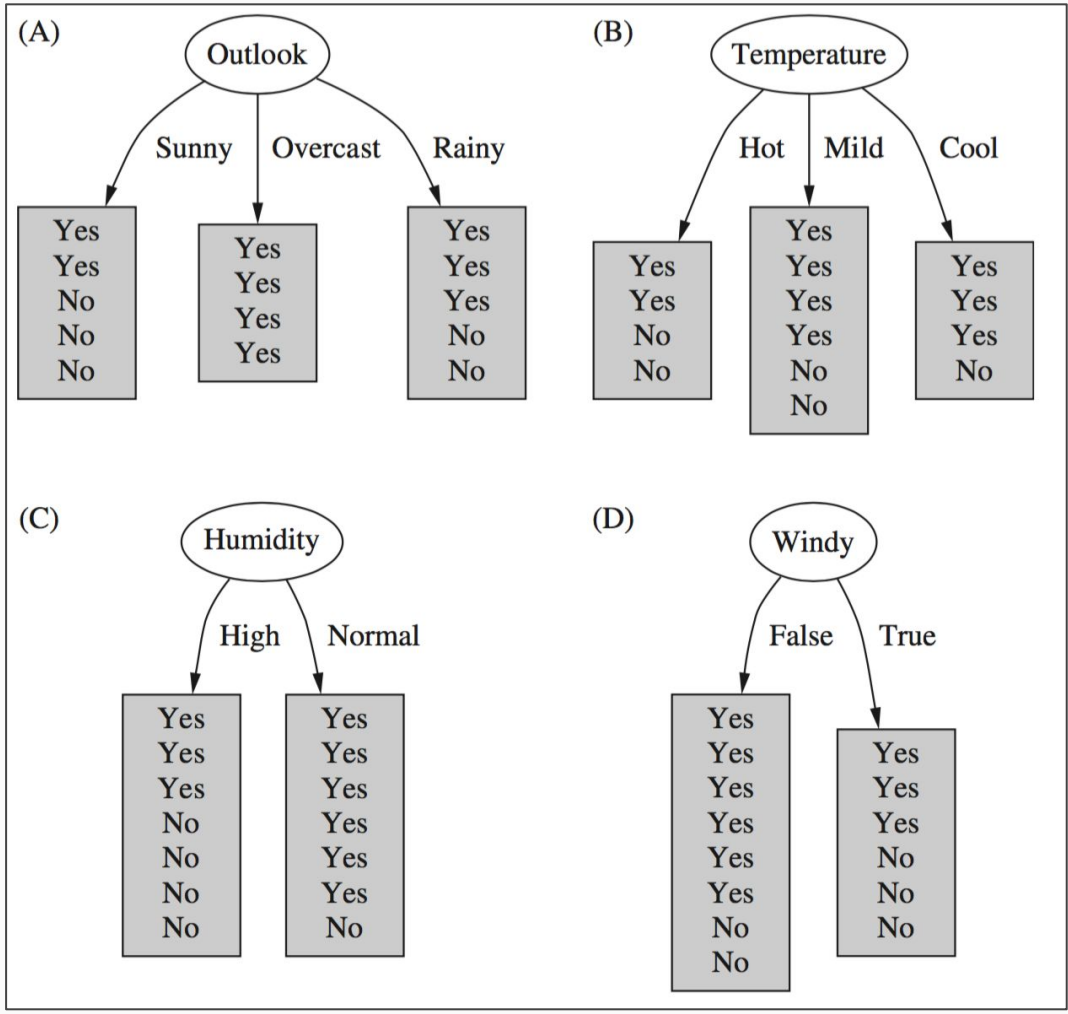
- Entropy
  - An alternative impurity measure that helps identifying the **information gain** of a feature
  - Same as the Gini impurity, entropy is zero when all instances of a node belong to the same class
- Entropy vs Gini
  - Gini is faster to compute
  - Entropy produces more balanced trees


"SPONTANEOUS" REACTION
as time elapses
ORGANIZED EFFORT REQUIRING ENERGY INPUT

Entropy

outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Which attribute should be selected?



ID3: Play tennis or not?

ID3
Using entropy to select the most appropriate attribute per branch

- Determine the attribute that best classifies the training data (i.e. has the most **information gain**); use this attribute at the root of the tree
- Repeat this process for each branch until
  - either all examples belong to the same class (positive or negative)
  - or there are no more properties, in which case, return a leaf node with a disjunction of classes
- Problem: How do we choose the best attribute?
- Answer: Use the attribute with the highest information gain

Information Entropy

- S is the training set
- p_+ is the ratio: *positive examples / all examples*
- p_- is the ratio: *negative examples / all examples*

- The **entropy** value of S is:

$$E(S) = -p_+ \times log_2(p_+) - p_- \times log_2(p_-)$$

- Compute the entropy for the **whole** data set
- Out of 14 instances, 9 are classified as positive and 5 as negative

$$E(S) = -p_+ \times \log_2(p_+) - p_- \times \log_2(p_-)$$

$$= -(9/14) \times \log_2(9/14) - (5/14) \times \log_2(5/14)$$

$$= 0.41 + 0.53 = 0.94$$

ID3: Play tennis or not?

Information Gain

$$\text{Gain}(S,A) = E(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} E(S_v)$$

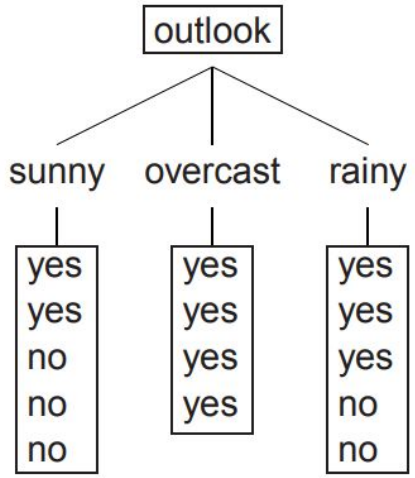
- S: the training set
- Values(A): set of all possible values of attribute A
- S_v: subset of S for which A has value v
- | S |: size of S
- | S_v |: size of S_v
- E(S): entropy value of the entire set S

- Compute the information gain for the attribute **windy** and **humidity** in the data set
- Humidity provides greater information gain than windy

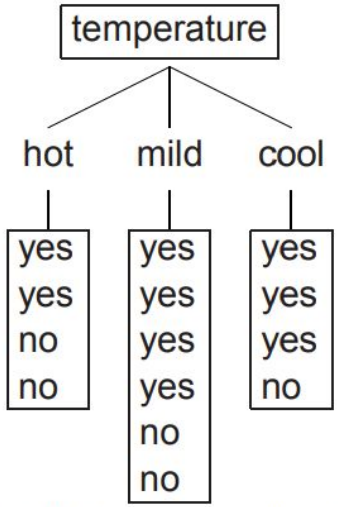
$$\begin{aligned}
\text{Gain}(S, \text{windy}) &= E(S) - \frac{|S_{\text{false}}|}{|S|}E(S_{\text{false}}) - \frac{|S_{\text{true}}|}{|S|}E(S_{\text{true}}) \\
&= 0.94 - (8/14)0.81 - (6/14)1.00 \\
&= 0.048
\end{aligned}$$

$$\begin{aligned}
\text{Gain}(S, \text{humidity}) &= E(S) - \frac{|S_{\text{high}}|}{|S|}E(S_{\text{high}}) - \frac{|S_{\text{normal}}|}{|S|}E(S_{\text{normal}}) \\
&= 0.94 - (7/14)0.96 - (7/14)0.59 \\
&= \boxed{0.15}
\end{aligned}$$

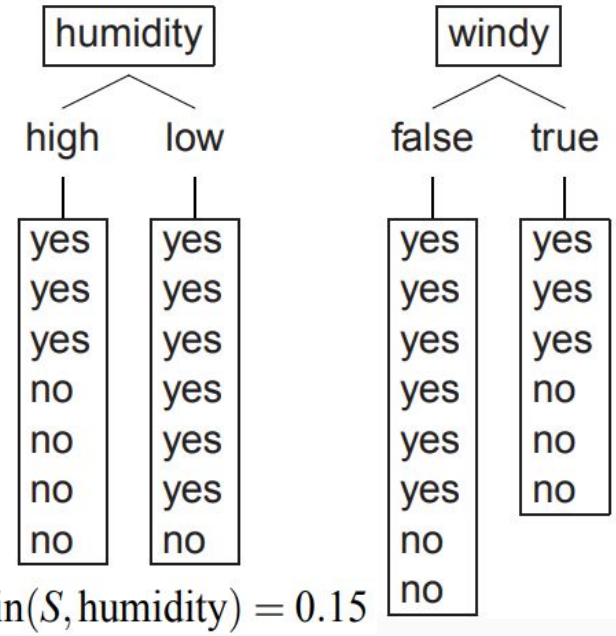
Example: Information Gain for tennis playing



$\text{Gain}(S, \text{outlook}) = 0.25$



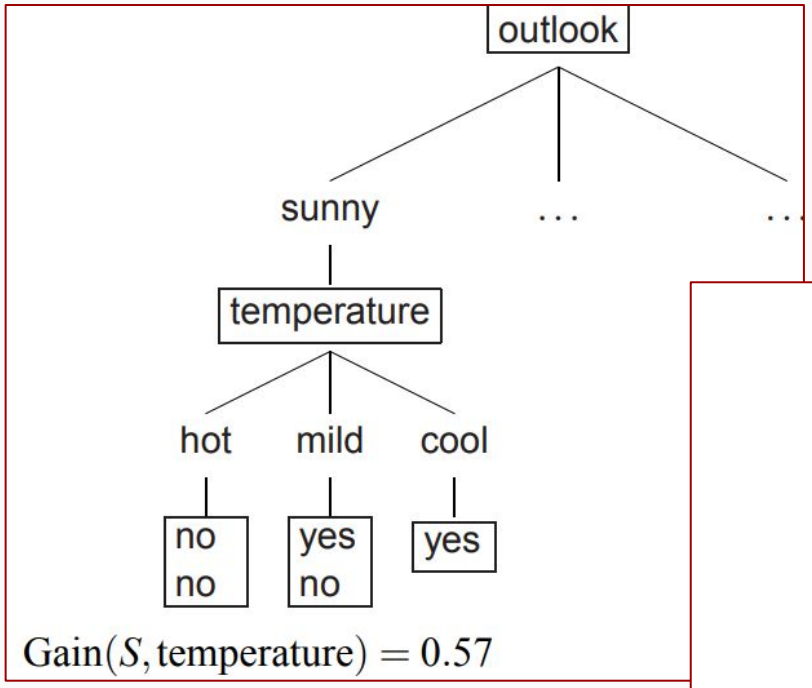
$\text{Gain}(S, \text{temperature}) = 0.03$

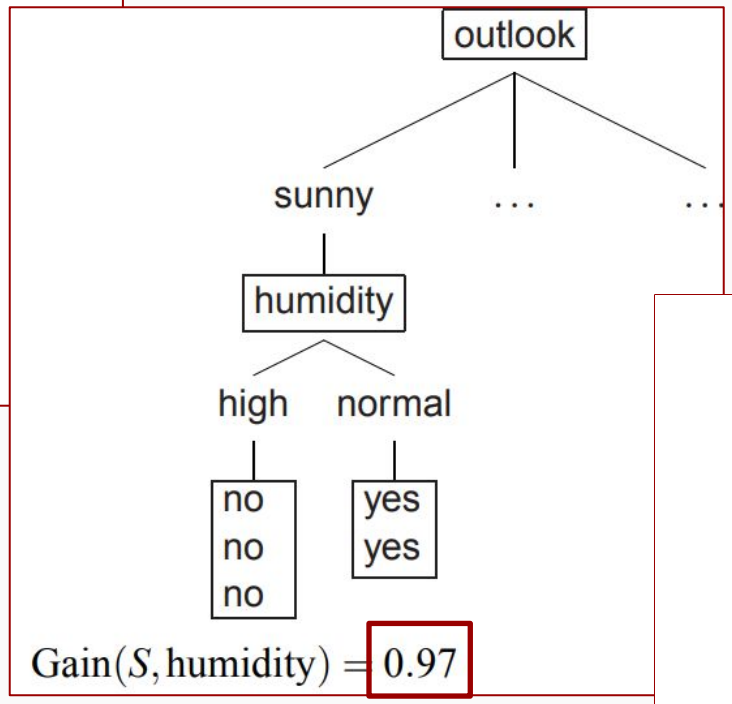


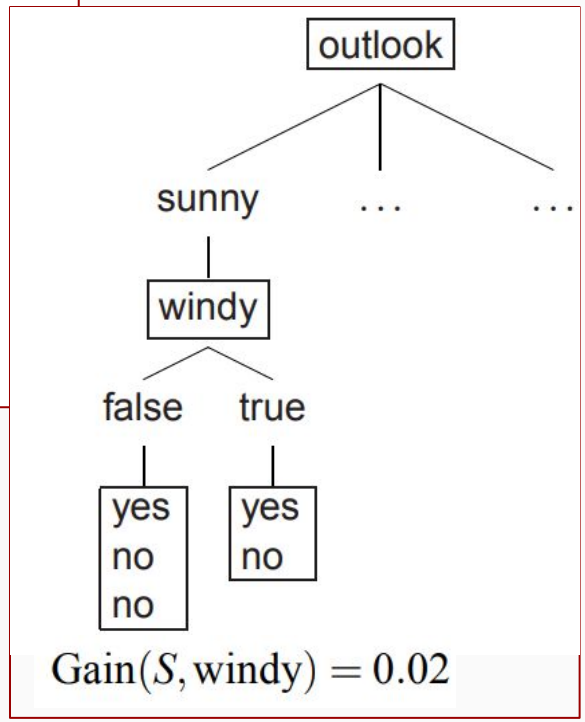
$\text{Gain}(S, \text{humidity}) = 0.15$

$\text{Gain}(S, \text{windy}) = 0.05$

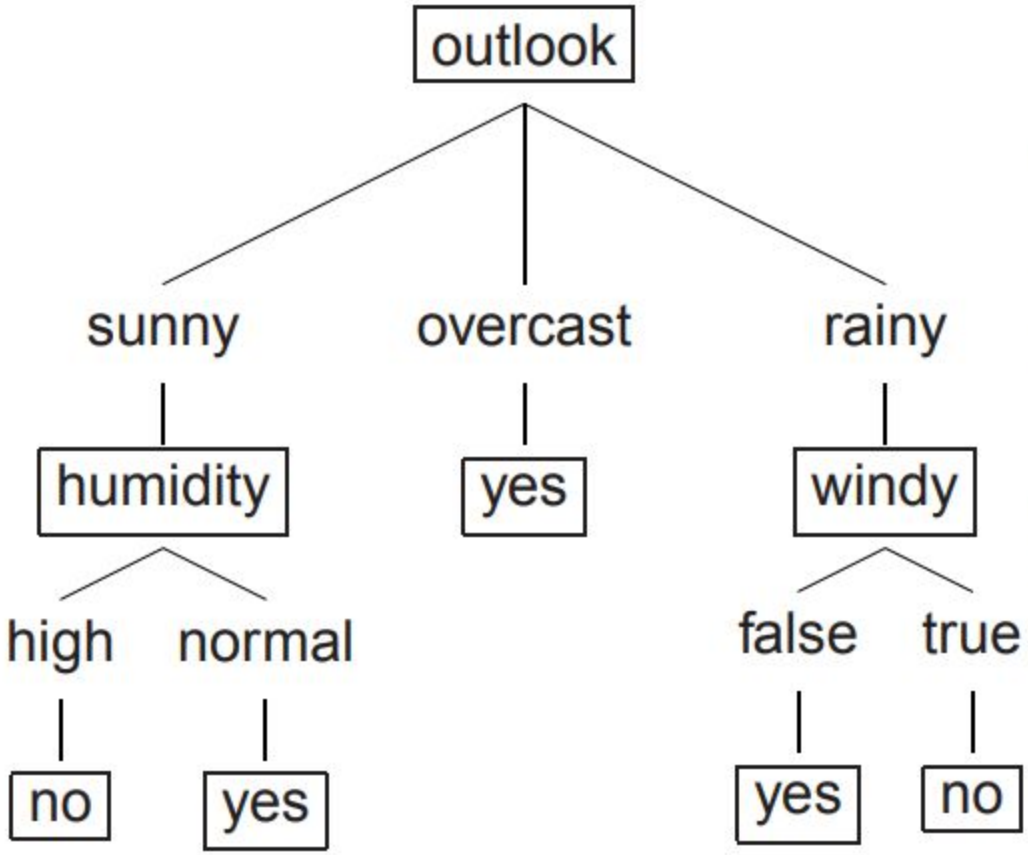
We choose outlook to be the root node because it has the highest gain







And then we have three choices



Note: all leaf nodes are associated with training examples from the same class (entropy = 0).

Note: the attribute temperature is not used.

Decision Trees: Regularisation

- Decision Trees make no assumptions about the data
- We call these **nonparametric** models:
  - Non-mathematical
  - Rely on finite set of parameters

  ⇒ if left unconstrained, they will **most likely overfit** the training data
- To avoid overfitting, we need to restrict the Decision Tree's freedom

Decision Trees: Hyperparameters

`max_depth`
The most common regularisation hyperparameter

`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

To regularise:
increase the `min_*` and **reduce** the `max_*` hyperparameters

Decision Tree Classifier

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)
```

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

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

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

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

Decision Trees for Regression

- Decision Trees may also be used for **regression**
  - *Outside the scope of this module*

Limitations of Decision Trees

- DTs are quite sensitive to variations in the training data
  - Even small ones, like adding/removing a few instances
  - Particularly sensitive by data rotation
    ⇒ Principal Component Analysis
    (outside the scope of this module)
- Random Forests limit these by averaging predictions over many trees

Ensemble Methods

Ensemble Methods

- Wisdom of the Crowd
  - What is the best lifeline in “who wants to be a millionaire”?
- The Law of Large Numbers
  - The average results of a large number of trials tends to be the expected

    e.g. consider a coin toss
- By aggregating the predictions of a group of predictors (classifiers or regressors) you get better results than by any one of the individual predictors

Popular Ensemble Methods

- Voting Classifiers
- Bagging and Pasting Ensembles
  - Random Forests
- Boosting ensembles
- Stacking ensembles

Ensemble Methods

Voting Classifiers

- Train many diverse classifiers
- **Hard Voting** ensemble classifier
  - the predicted class is the one that was "voted" most by the individual classifiers
- **Soft Voting** ensemble classifier
  - if your individual predictors estimate class probabilities, then
  - the predicted class is the one with the highest class probability, averaged over all the individual classifiers

Ensemble Methods

Diversity among the classifiers

- Why is diversity important?
- So that the individual predictors make uncorrelated, independent errors!
- Ensuring diversity
  - Use very **different training algorithms** for your predictors
    OR
  - use same training algorithm but **train on different random subsets of the training set**

Ensemble Methods

Instance Sampling with Bagging or Pasting

- Scikit class: **`BaggingClassifier`**
- **Bagging is sampling <u>with</u> replacement**
  - An instance sampled for one classifier is placed back in the training set and may, therefore, be sampled again for another classifier
  - Hyperparameter **`bootstrap = true`**
  - Hyperparameter **`max_samples`** defines the size of the subset
- **Pasting is sampling <u>without</u> replacement**
  - An instance sampled for one classifier is not placed back in the training set and will not, therefore, be sampled again for another classifier

Ensemble Methods

Out-of-Bag Evaluation

- Since each predictor sees only a subset of the training set, there is another subset that the predictor has never seen
  - Out-of-bag (OOB) instances
- The OOB instances subset may be used for evaluation
  - No need for putting aside a separate validation set
  - Hyperparameter **`oob_score = true`**

Ensemble Methods

Feature Sampling

- The **`BaggingClassifier`** class also supports sampling the features
- Each predictor is trained on a **random subset of features**
- Hyperparameters (similar to instances sampling): **`max_features`** and **`bootstrap_features`**
- **Particularly useful technique when dealing with high-dimensional inputs**

Ensemble Methods

Feature Sampling (cont'd)

- **Random Patches**
  - Sampling only features
  - i.e. NOT sampling instances
- **Random Subspaces**
  - Sampling BOTH instances AND features

Ensemble Methods

Random Forests

- Ensemble of Decision Trees
- Typically trained using the bagging technique
- Introduces extra randomness when growing the trees:
  - Instead of searching for the best feature of all when splitting a node, it searches for the best feature of a random subset of features

  ⇒ greater tree diversity!

Ensemble Methods

Random Forests in Scikit

- Instead of using a **`BaggingClassifier`** and passing it a **`DecisionTreeClassifier`**
- use **`RandomForestClassifier`**

  (or **`RandomForestRegressor`**)

  ⇒ optimised for Decision Trees!

Random Forest Classifier

The iris dataset

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16,
                                 n_jobs=-1, random_state=42)
rnd_clf.fit(X_train, y_train)
y_pred_rf = rnd_clf.predict(X_test)
```

A Random Forest is equivalent to a bag of decision trees:

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(max_features="sqrt", max_leaf_nodes=16),
    n_estimators=500, n_jobs=-1, random_state=42)
```

Ensemble Methods

Random Forests: Feature importance

- We can measure a feature's importance by looking at how much the tree nodes that use that feature reduce impurity on average (across all trees in the forest)
- Scikit computes this score automatically and scales the results so that the sum of all importances is equal to 1

```
sepal length (cm) 0.112492250999
sepal width (cm) 0.0231192882825
petal length (cm) 0.441030464364
petal width (cm) 0.423357996355
```

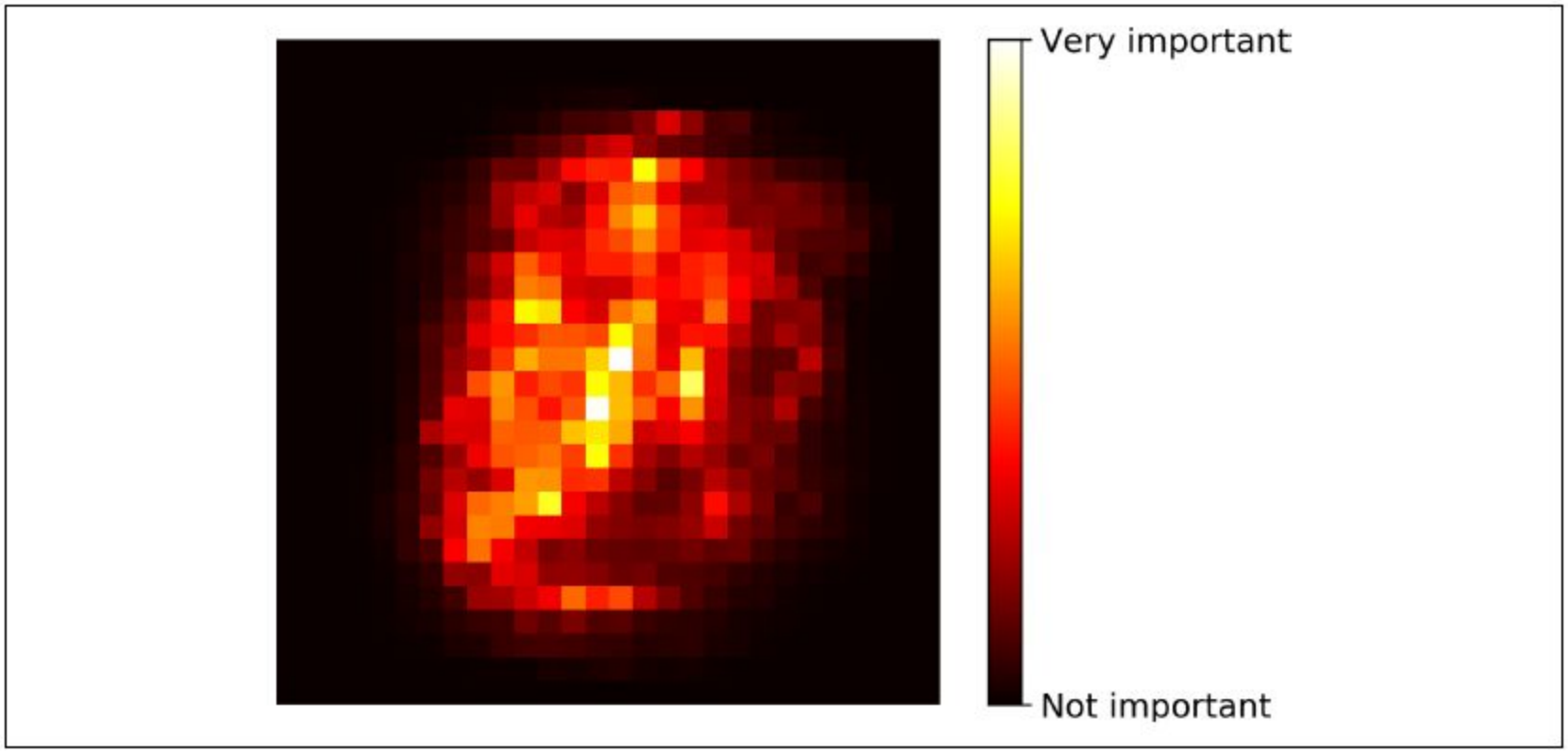


Figure 7-6. MNIST pixel importance (according to a Random Forest classifier)

Feature importance for the IRIS and MNIST datasets

Thank you!

Coming up next:
Unsupervised Learning: Clustering