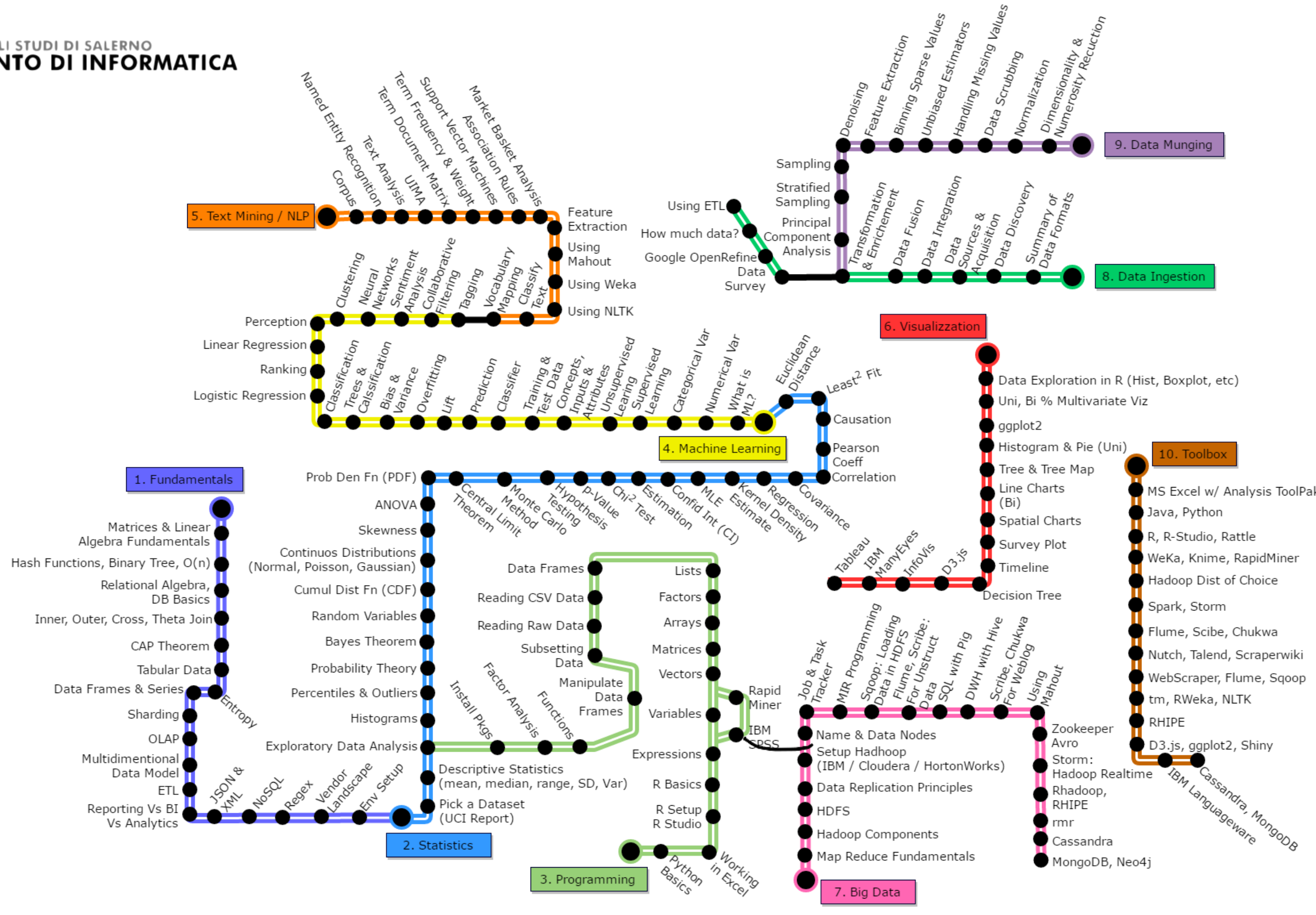
UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA



Fondamenti di Data Science e Machine Learning

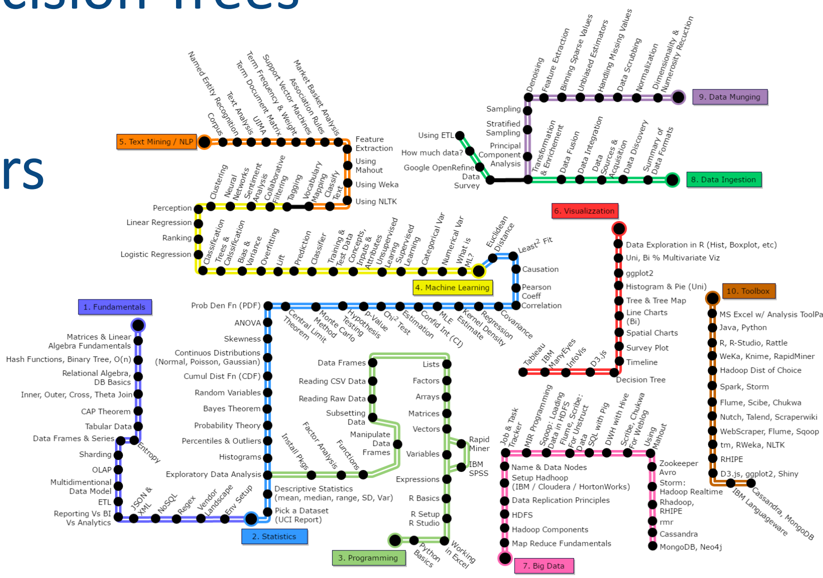
Decision Trees (Chapter 6 Geron's Book)

Aurelien Geron: «Hands on Machine Learning with Scikit Learn and TensorFlow, O'Reilly ed.

Prof. Giuseppe Polese, aa 2024-25

Outline

- **Decision Trees**
  - Training and Visualizing a Decision Tree
  - Making Predictions
  - Estimating Class Probabilities
  - Algorithms for constructing Decision Trees
  - Computational Complexity
  - Regularization Hyperparameters
- Decision Trees Regression
  - Instability

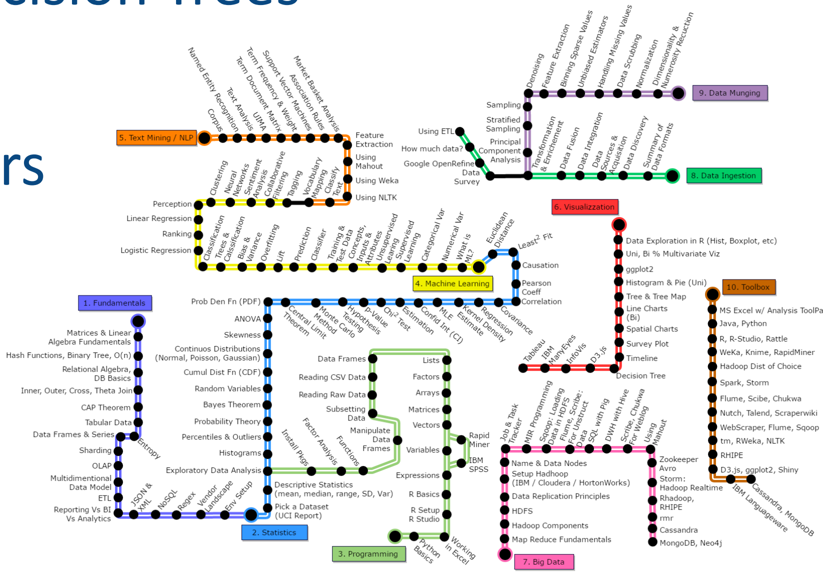


Decision Trees

- Like SVMs, *Decision Trees* are versatile Machine Learning algorithms that can perform
  - classification tasks
  - regression tasks
  - multi-output tasks
- They are very powerful algorithms, capable of fitting complex datasets
- Decision Trees are also the fundamental components of *Random Forests*, which are among the most powerful Machine Learning algorithms available today

Outline

- **Decision Trees**
  - **Training and Visualizing a Decision Tree**
  - Making Predictions
  - Estimating Class Probabilities
  - Algorithms for constructing Decision Trees
  - Computational Complexity
  - Regularization Hyperparameters
- Decision Trees Regression
  - Instability

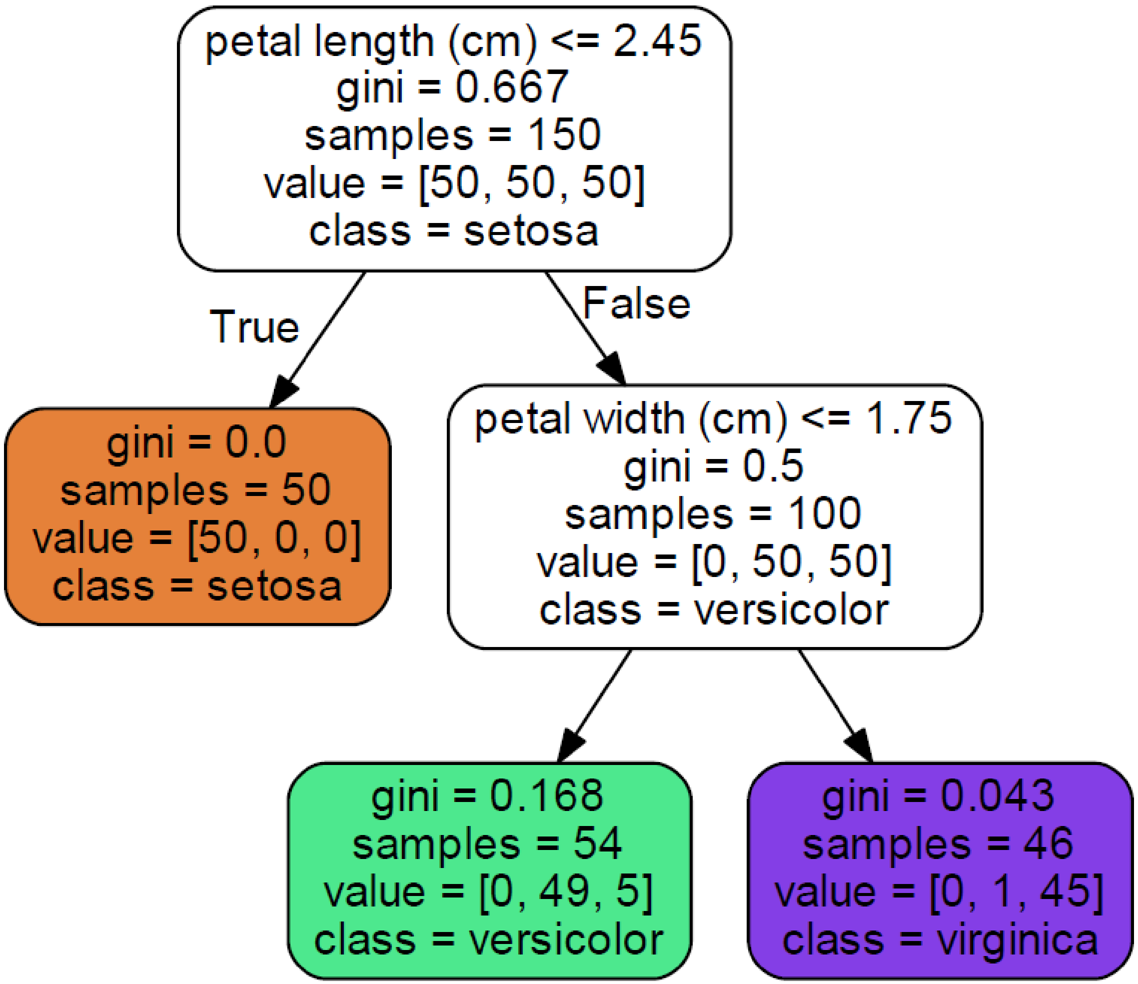


Training and Visualizing a Decision Tree

- Let's just build one and take a look at how it makes predictions. The following code trains a *DecisionTreeClassifier* on the iris dataset:

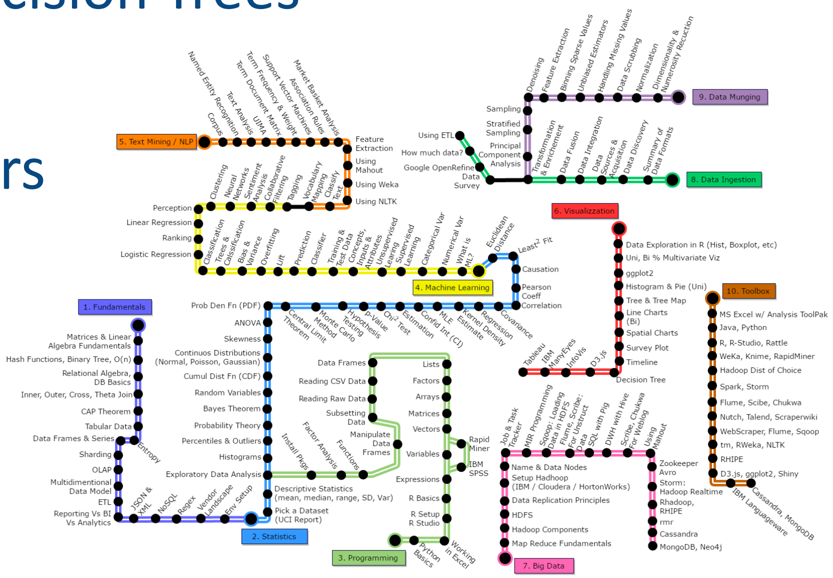
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from graphviz import Source

iris = load_iris() #Load iris dataset
#Petal length and width
X = iris.data[:, 2:]
y = iris.target
#Application of the Decision Trees classification
tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X, y)
#Costruction of .dot file for plotting graphic
export_graphviz(tree_clf, out_file="decisiontree.dot",
class_names=iris.target_names,
     feature_names=iris.feature_names[2:], rounded= True, filled=True)
#Source of .dot file
path = 'folder-containing-the-file/decisiontree.dot'
s = Source.from_file(path)
s.view()
```

Training and Visualizing a Decision Tree



Outline

- **Decision Trees**
  - Training and Visualizing a Decision Tree
  - **Making Predictions**
  - Estimating Class Probabilities
  - Algorithms for constructing Decision Trees
  - Computational Complexity
  - Regularization Hyperparameters
- Decision Trees Regression
  - Instability



Making Predictions (1)

- Suppose you find an iris flower and you want to classify it

- You start at the *root node* (depth 0, at the top)
  - this node asks whether the flower's petal length is smaller than 2.45 cm

- If it is, then you move down to the root's left child node (depth 1, left)
  - it is a *leaf node*, so it does not ask any questions
  - the Decision Tree predicts that your flower is an Iris-Setosa

Making Predictions (2)

- Now suppose you find another flower, but this time the petal length is greater than 2.45 cm
- You must move down to the root's right child node (depth 1, right)
  - it asks another question: is the petal width smaller than 1.75 cm?
- If it is
  - your flower is most likely an Iris-Versicolor (depth 2, left)
- If not
  - it is likely an Iris-Virginica (depth 2, right).

Making Predictions (3)

- A **node's samples** attribute counts how many training instances it applies to.
  - For example, 100 training instances have a petal length greater than 2.45 cm (depth 1, right), among which 54 have a petal width smaller than 1.75 cm (depth 2, left)
- A **node's value** attribute tells you how many training instances of each class this node applies to
  - For example, the bottom-right node applies to 0 Iris-Setosa, 1 Iris- Versicolor, and 45 Iris-Virginica

Making Predictions (4)

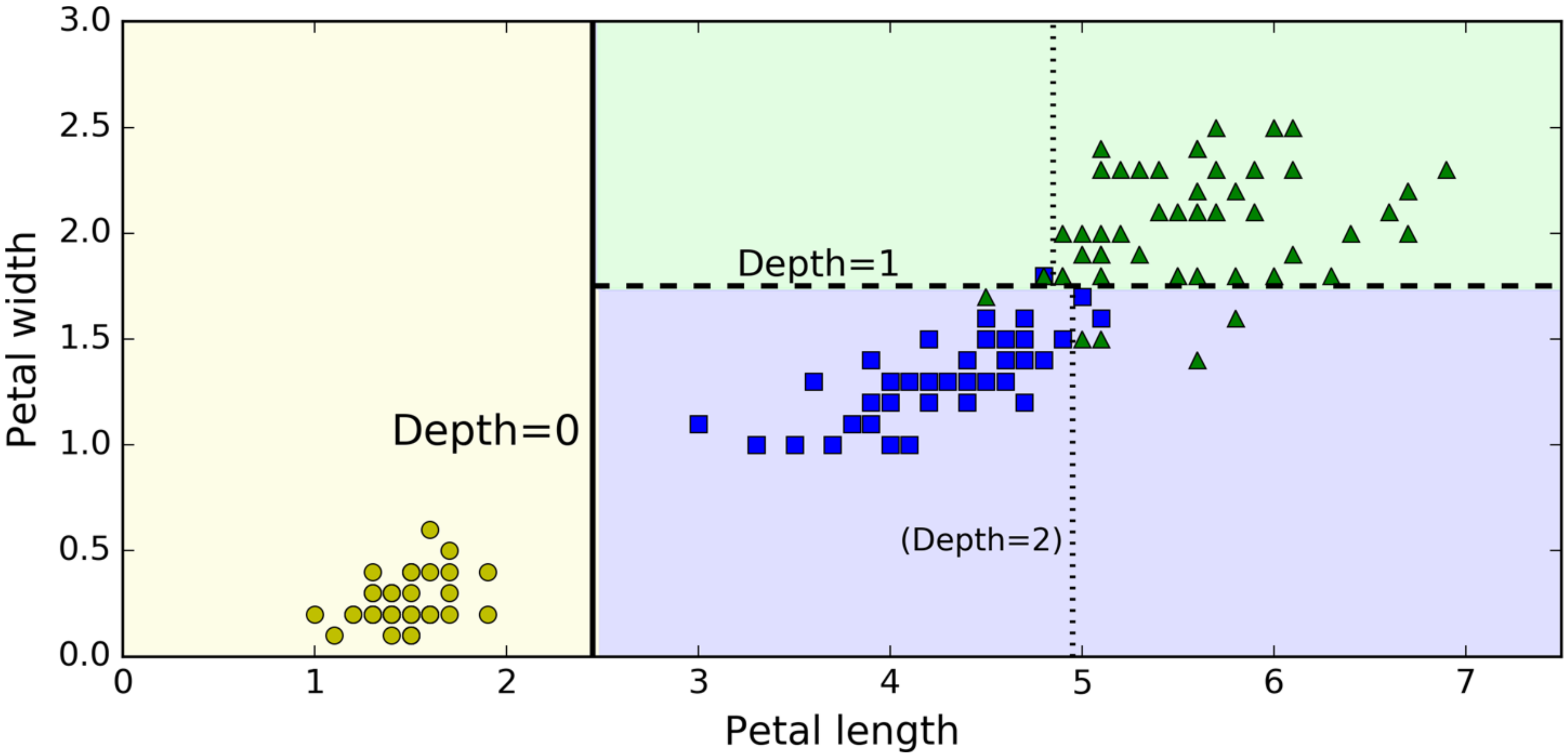
- A **node's gini** attribute measures its impurity
  - a node is "pure" (gini=0) if all training instances it applies to belong to the same class
  - For example, since the depth-1 left node applies only to Iris-Setosa training instances, it is pure and its gini score is 0
- The following equation shows how the training algorithm computes the Gini score 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
- For example, the depth-2 left node has a **gini** score equal to $1 – (0/54)^2 – (49/54)^2 – (5/54)^2 \approx 0.168$

Recursive Formulation of Gini

- Like other split indices, **Gini's index** measures the quality of alternative split criteria for an attribute
- Given a dataset T with examples belonging to n classes, with p_i frequence of class *i* in *T*
- If T is subdivided in T_1 with examples belonging to n_1 classes and T_2 with examples belonging to n_2 classes:
  - $gini_{split}(T) = (n_1/n)\, gini(T_1) + (n_2/n)\, gini(T_2)$
- Splits with lower indices are better ones

Decision Boundaries (1)

- This Figure shows the Decision Tree’s decision boundaries

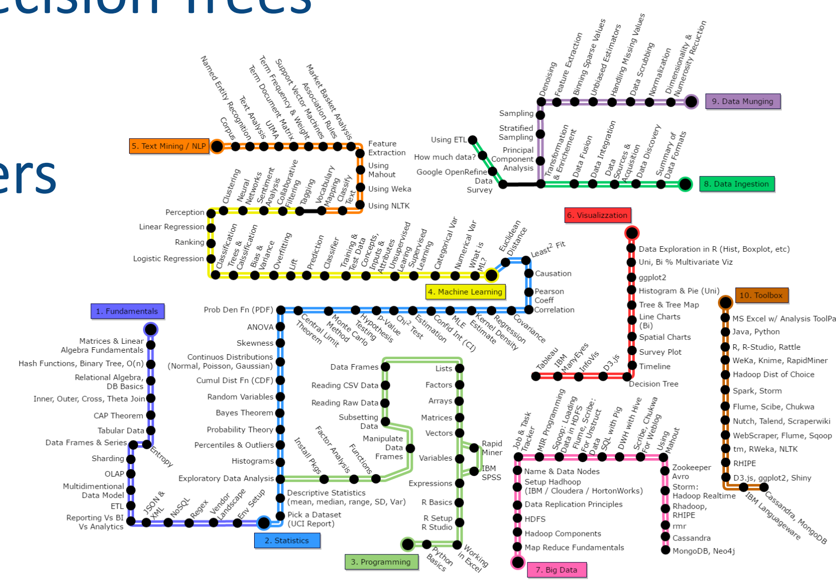


Decision Boundaries (6)

- The thick vertical line represents the decision boundary of the root node (depth 0)
  - petal length = 2.45 cm
- Since the left area is pure (only Iris-Setosa), it cannot be split any further
- The right area is impure
  - the depth-1 right node splits it at petal width = 1.75 cm
    - represented by the dashed line
- Since *max_depth* was set to 2, the Decision Tree stops here
  - However, if you set *max_depth* to 3, then the two depth-2 nodes would each add another decision boundary
    - represented by the dotted lines

Outline

- **Decision Trees**
  - Training and Visualizing a Decision Tree
  - Making Predictions
  - **Estimating Class Probabilities**
  - Algorithms for constructing Decision Trees
  - Computational Complexity
  - Regularization Hyperparameters
- Decision Trees Regression
  - Instability



Estimating Class Probabilities

- A Decision Tree can also estimate the probability that an instance belongs to a particular class *k*
  - first it traverses the tree to find the leaf node for this instance, then it returns the ratio of training instances of class *k* in this node
- Suppose you have found a flower whose petals are 5cm long and 1.5cm wide
  - The corresponding leaf node is the depth-2 left node
- The Decision Tree should output the following probabilities
  - 0% for Iris-Setosa (0/54)
  - 90.7% for Iris-Versicolor (49/54)
  - 9.3% for Iris-Virginica (5/54)
- If you ask it to predict the class, it should output Iris-Versicolor (class 1) since it has the highest probability

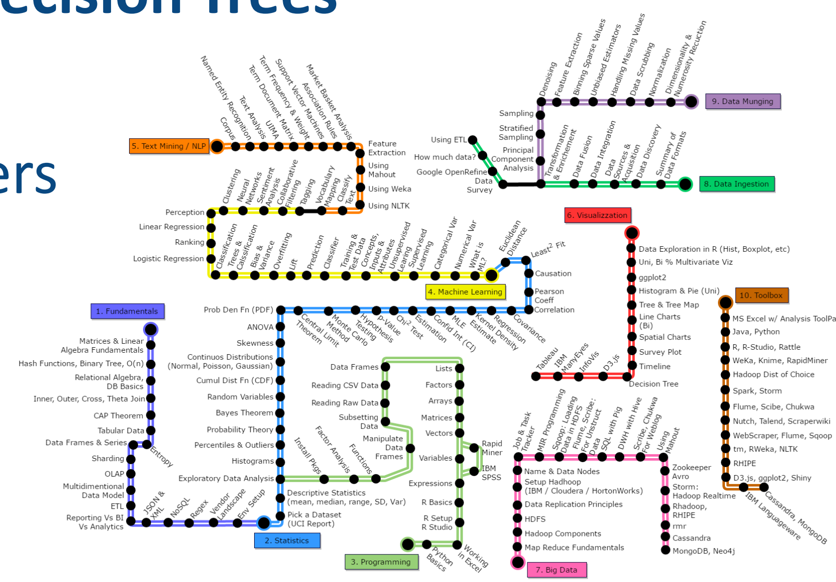
Estimating Class Probabilities: Example Code

```python
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
#Load iris dataset
iris = load_iris()
#Petal length and width
"""iris.data is a matrix from which we take all rows (:,), and 2
out of 4 columns, from index 2 on"""
X = iris.data[:, 2:]
y = iris.target
#Application of the Decision Trees classification
tree_clf = DecisionTreeClassifier(max_depth=2,random_state=42)
tree_clf.fit(X, y)
#Probability
prob = tree_clf.predict_proba([[5,1.5]])
print ("Prob: {}".format(prob))
#Prediction
pred = tree_clf.predict([[5,1.5]])
print ("Pred: {}".format(pred))
```

```
Prob: [[0.         0.90740741 0.09259259]]
Pred: [1]

Process finished with exit code 0
```

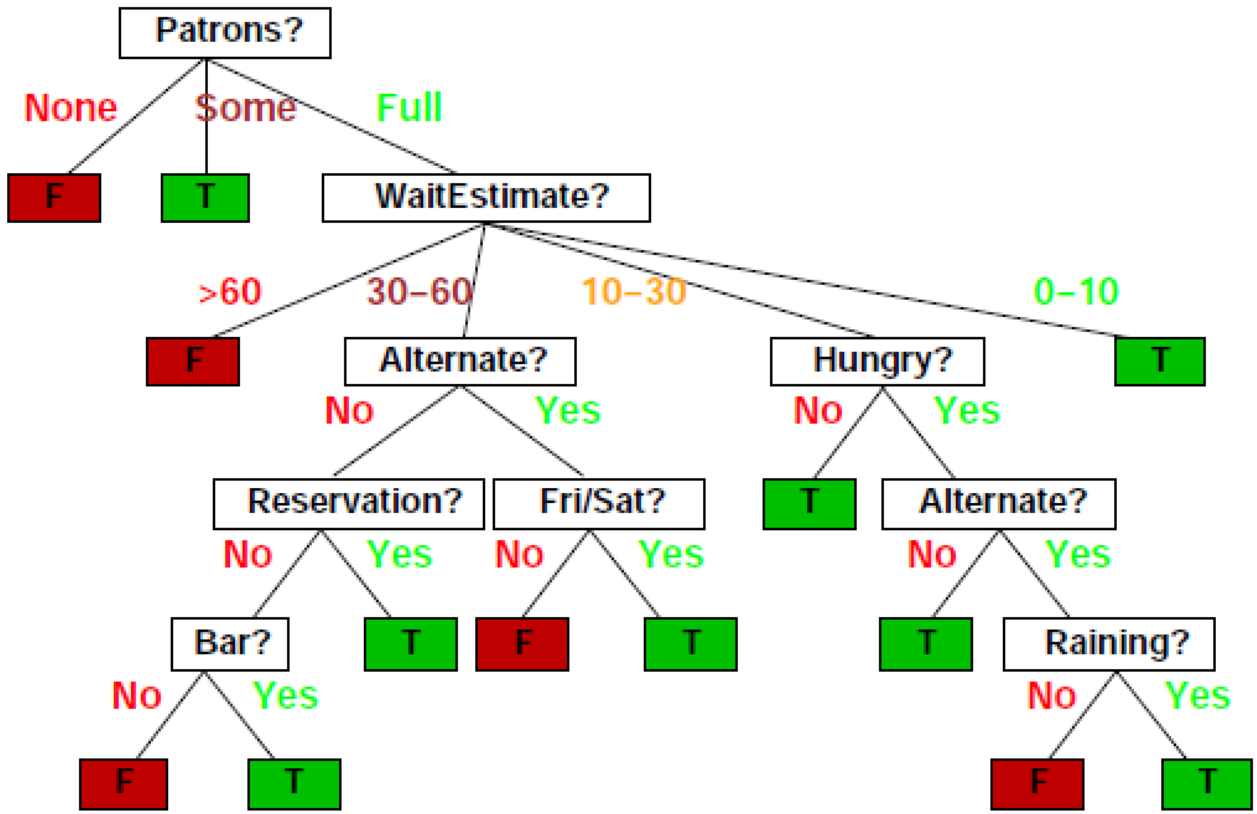
Outline

- **Decision Trees**
  - Training and Visualizing a Decision Tree
  - Making Predictions
  - Estimating Class Probabilities
  - **Algorithms for constructing Decision Trees**
  - Computational Complexity
  - Regularization Hyperparameters
- Decision Trees Regression
  - Instability



Constructing Decision Trees: Example

- A Decision Tree for deciding whether to wait for a seat at a restaurant



A sample Training Dataset

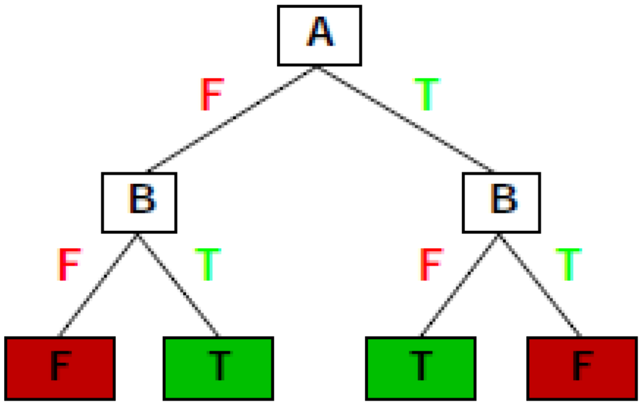
- Let us consider a training dataset, which should allow us to decide whether to wait outside of a restaurant

Instance ID	**Predictive Attributes**										**Dependent Attribute** (Will Wait)
	Alternatives Nearby	Comfort Bar Inside	Fri-Sat	Hungry	Patrons (Crowded)	Price	Rains Out	Have Reservation	Type	Estimated Wait(Waiter)	
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0-10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30-60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*			*Burger*	*0-10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10-30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0-10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0-10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0-10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10-30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0-10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30-60*	*T*

Expressiveness

- Decision trees can express any function of predictive attributes (for Boolean functions, a truth table row → path to leaf):

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- There is a consistent decision tree for any training set: one path to leaf for each example (unless f nondeterministic in x)
- But it probably won't generalize to new examples
- Prefer to find more compact decision trees

Hypothesis Spaces

- How many distinct decision trees with n Boolean attributes??
- = number of Boolean functions
- = number of distinct truth tables with 2^n rows = 2^{2^n}
- E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

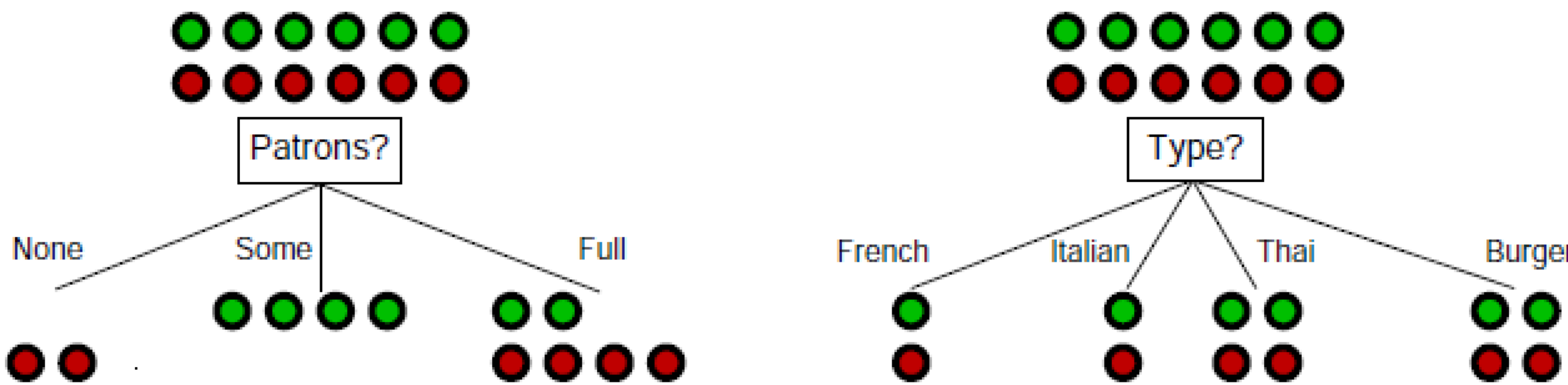
Decision Tree Learning

- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

function DTL(*examples, attributes, default*) **returns** a decision tree
    if *examples* is empty **then return** *default*
    else if all *examples* have the same classication then return the classification
    else if attributes is empty then return Majority-Value(*examples*)
    else
        best ← Choose-Attribute(*attributes, examples*)
        tree ← a new decision tree with root test *best*
        for each value v_i of *best* do
            $examples_i$ {elements of *examples* with *best* = vi}
            subtree ← DTL($examples_i$; *attributes* - best, Majority-Value(*examples*))
            add a branch to *tree* with label v_i and subtree *subtree*
        return *tree*

Choosing most significant attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) “all positive" or “all negative“



- Patrons? is a better choice

Information

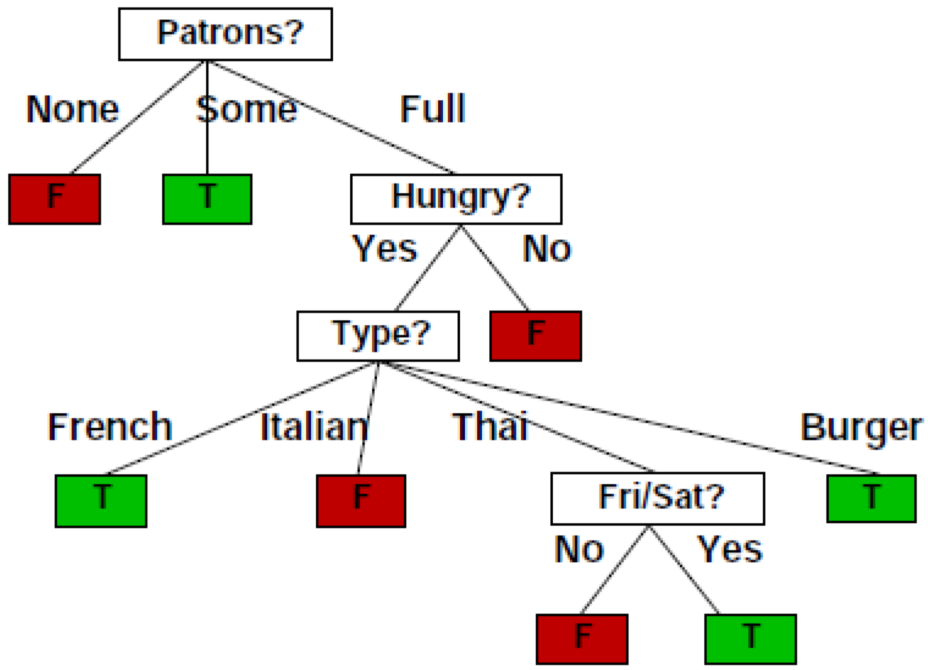
- Information answers questions
- The more clueless I am about the answer initially, the more information is contained in the answer
- Scale: 1 bit = answer to Boolean question with prior $<0:5; 0:5>$
- Information in an answer when prior is $<P_1;\ldots; P_n>$ is
  - $H(<P_1;\ldots; Pn>) = -\sum_{i=1}^{n} P_i \log_2 P_i$
  - Also called entropy of the prior

Choosing Best Attribute with Entropy

- *p* positive and *n* negative examples at the root, then $H(<p/(p+n), n/(p+n)>)$ *bits to classify a new example*
  - E.g., for 12 restaurant examples, p=n=6 so we need 1 bit
- An attribute splits examples E into subsets E_i, each of which hopefully needs less information to complete the classification
- Let E_i have p_i positive and n_i negative examples, then $H(<p_i/(p_i+n_i), n_i/(p_i+n_i)>)$ *bits to classify a new example*
- Expected number of bits per example over all branches is
  - $\sum_i \frac{p_i+ni}{p+n} H(<pi_{/(}pi+ni),ni/(pi+ni)>)$
  - For *Patrons?*, this is 0.459 bits, for *Type* is (still) 1 bit
  - Choose the attribute minimizing remaining information needed!

Learned Decision Tree

- Decision tree learned from the 12 examples:



The CART Training Algorithm (1)

- **Scikit-Learn** uses the Classification And Regression Tree (CART) algorithm to train Decision Trees (also called "growing" trees)
  - The algorithm first splits the training set in 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)
- The cost function that the algorithm tries to minimize is:

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

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

The CART Training Algorithm (2)

- Once it has successfully 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

- Unfortunately, finding the optimal tree is known to be an *NP-Complete* problem
  - It requires $O(\exp(m))$ time, making the problem intractable even for fairly small training sets

Gini Impurity or Entropy?

- By default, the Gini impurity measure is used
- You can select the *entropy* impurity measure instead by setting the criterion hyperparameter to "entropy"
  - In Information Theory entropy measures the average information content of a message, it is zero when all messages are identical
  - In ML entropy is frequently used as an impurity measure: a set's entropy is zero when it contains instances of only one class
- Equation Entropy:
  - $H_i = -\sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log(p_{i,k})$
- For example, the depth-2 left node in the Iris Decision Tree has an entropy equal to $-\frac{49}{54}\log\left(\frac{49}{54}\right) - \frac{5}{54}\log\left(\frac{5}{54}\right) \approx 0.31$

Gini's example

AGE	CAR TYPE	RISK
40	Station wagon	low
65	Sport	high
20	Economy	high
25	Sport	high
50	Station wagon	low
48	Economy	high

- Two classes, n = 2
- Let us consider the candidate split AGE <= 25
- T1= {t3, t4}, n_1 = 1, T2={t1, t2, t5, t6}, n_2 = 2
- gini_split(T) = 1/2 (1 - 1) + 2/2 (1 - (1/4 +1/4)) = 1/2
- Best splits are those with low Gini's index

Split Algorithms

- Given an attribute A, they determine the best split predicate for it:
  - Two problems:
    - Selection of the predicate
    - Evaluating the quality of the chosen predicate
- Selection of the predicate:
  - Depends on the type of attribute
- Optimality:
  - A good predicate allows to achieve:
    - Less rules (less splits, tree nodes with lower con fan-out)
    - Higher support and confidence values

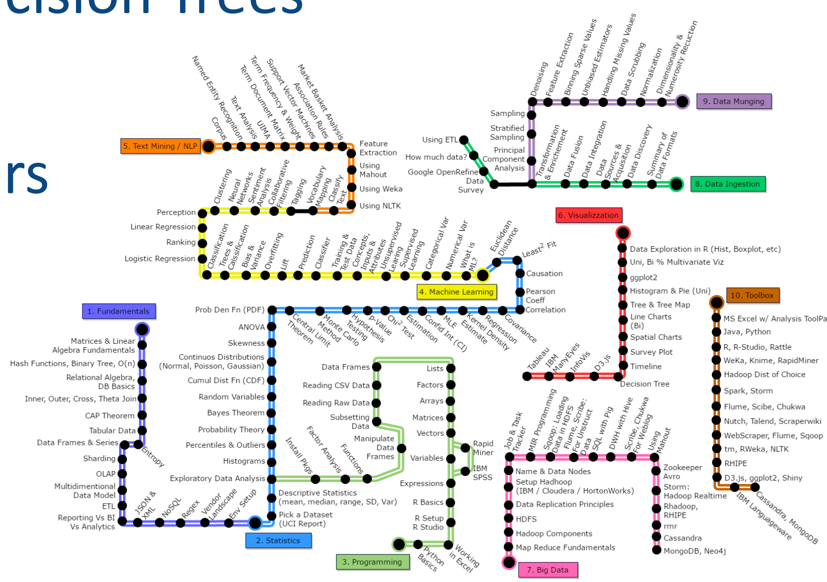
Split Predicates

- Splits can be
  - binary
  - multiple
- For numerical attributes
  - Binary split: $A <= v, A > v$
  - Multiple split: $A <= v1, v1 < A <= v2, \ldots, vn\text{-}1 < A <= vn$
- For categorical attributes, if domain of A is S:
  - Binary split: $A \in S', A \in S - S'$ con $S' \subset S$
  - Multiple split: $A \in S1, \ldots, A \in Sn$

with $S1 \cup \ldots \cup Sn = S$, $Si \cap Sj = \{\}$

Outline

- **Decision Trees**
  - Training and Visualizing a Decision Tree
  - Making Predictions
  - Estimating Class Probabilities
  - Algorithms for constructing Decision Trees
  - **Computational Complexity**
  - Regularization Hyperparameters
- Decision Trees Regression
  - Instability

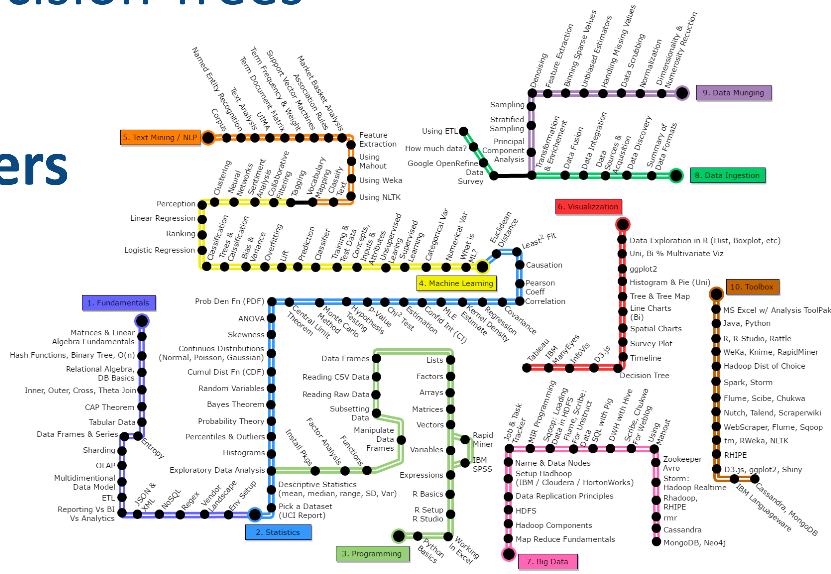


Computational Complexity

- Making predictions requires traversing the Decision Tree from the root to a leaf
  - Decision Trees are generally approximately balanced, so traversing the Decision Tree requires going through roughly $O(\log_2(m))$ nodes
  - Since each node only requires checking the value of one feature, the overall prediction complexity is just $O(\log_2(m))$, independent of the number of features
- However, the training algorithm compares all features (or less if max_features is set) on all samples at each node
  - This results in a training complexity of $O(n \times m\log(m))$

Outline

- **Decision Trees**
  - Training and Visualizing a Decision Tree
  - Making Predictions
  - Estimating Class Probabilities
  - Algorithms for constructing Decision Trees
  - Computational Complexity
  - **Regularization Hyperparameters**
- Decision Trees Regression
  - Instability



Regularization Hyperparameters (1)

- Decision Trees make very few assumptions about the training data
  - If left unconstrained, the tree structure will adapt to training data, fitting it very closely, and most likely overfitting it
- Such a model is often called a nonparametric model
  - The number of parameters is not determined prior to training, so the model structure is free to stick closely to the data
- To avoid overfitting the training data, we need to restrict the Decision Tree's freedom during training

Regularization Hyperparameters (2)

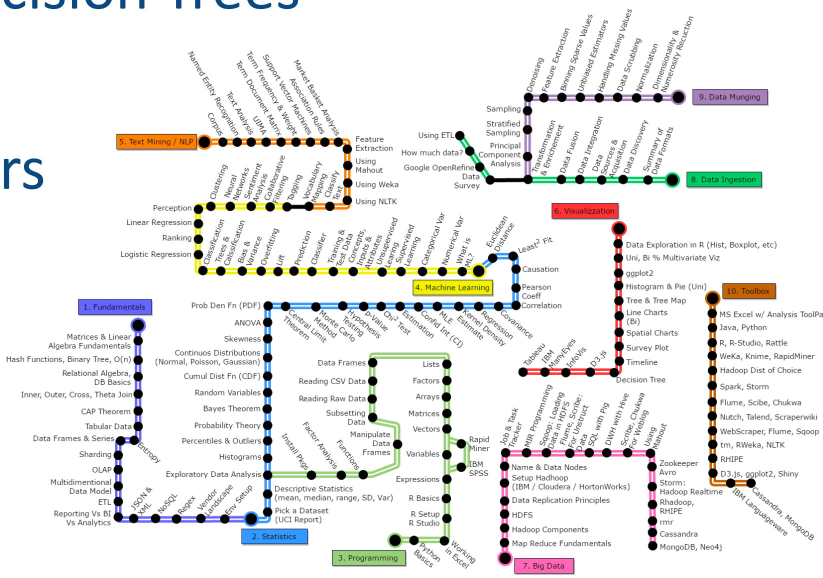
- The regularization hyperparameters depend on the algorithm used
  - generally we can at least restrict the maximum depth of the Decision Tree
- In **Scikit-Learn**, this is controlled by the *max_depth* hyperparameter
  - the default value is None, which means unlimited
- Reducing *max_depth* will regularize the model and thus reduce the risk of overfitting

Regularization Hyperparameters (3)

- DecisionTreeClassifier class has a few other parameters to restrict the shape of the Decision Tree:
  - *min_samples_split* (the minimum number of samples a node must have before it can be split)
  - *min_samples_leaf* (the minimum number of samples a leaf node must have)
  - *min_weight_fraction_leaf* (same as *min_samples_leaf* but expressed as a fraction of the total number of weighted instances)
  - max_leaf_nodes (maximum number of leaf nodes)
  - max_features (maximum number of features that are evaluated for splitting at each node)

Outline

- Decision Trees
  - Training and Visualizing a Decision Tree
  - Making Predictions
  - Estimating Class Probabilities
  - Algorithms for constructing Decision Trees
  - Computational Complexity
  - Regularization Hyperparameters
- **Decision Trees Regression**
  - Instability

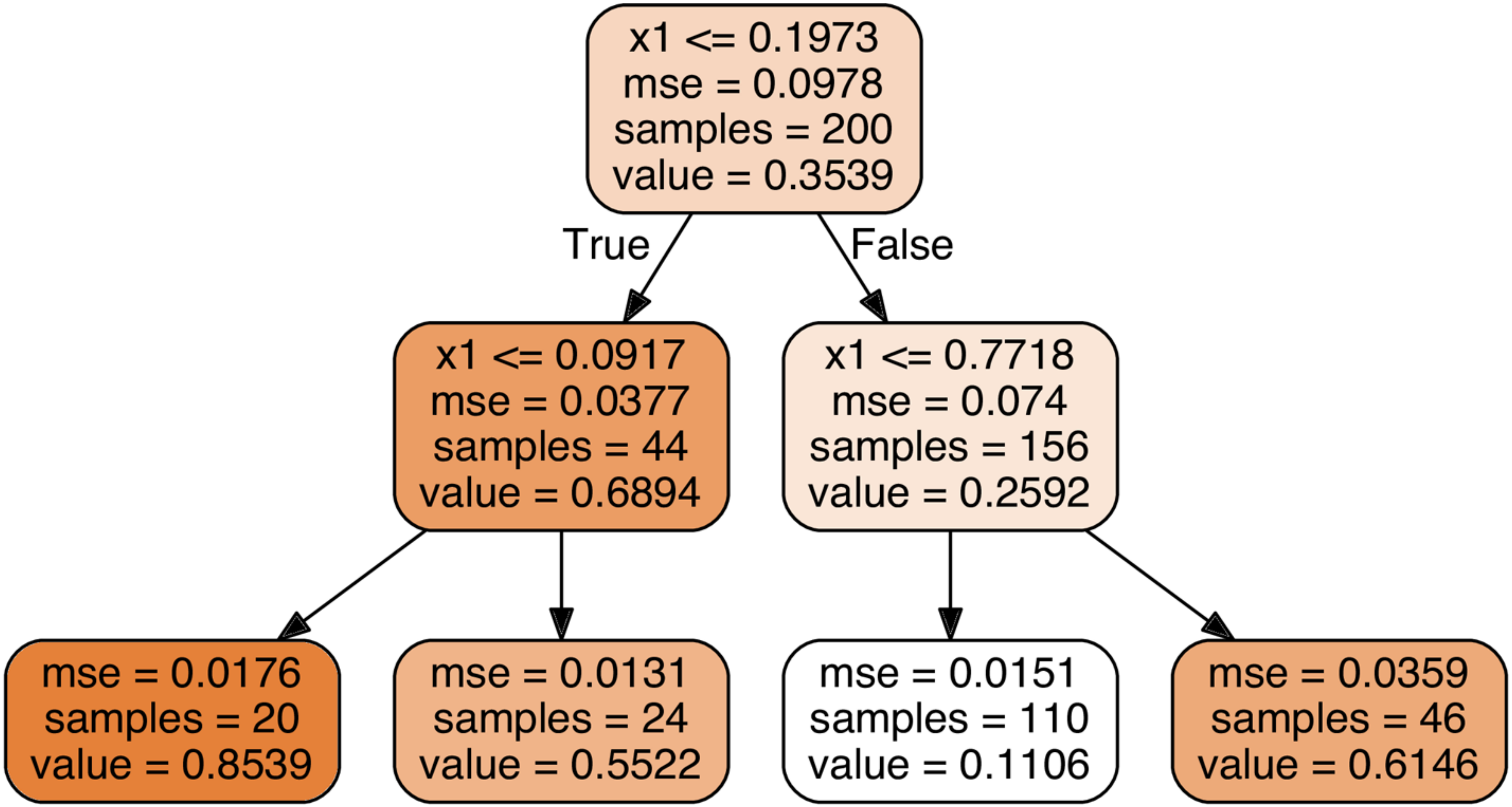


Decision Trees Regression (1)

- Decision Trees can also perform regression tasks. The following is an ad-hoc example code on noisy quadratic dataset with *max_depth = 2*:

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import export_graphviz
from graphviz import Source

# Quadratic training set + noise
np.random.seed(42)
m = 200
X = np.random.rand(m, 1)
y = 4 * (X - 0.5) ** 2
y = y + np.random.randn(m, 1) / 10
print("X:{}".format(X.shape))
print("y:{}".format(y.shape))

#Application of the Decision Trees Regressor
tree_reg = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg.fit(X, y)

#Costruction of .dot file for plotting graphic
export_graphviz(tree_reg, out_file="noisy.dot", feature_names=["x1"], rounded= True,
filled=True)

#Source of .dot file
path = 'folder-containing-the-file/noisy.dot'
s = Source.from_file(path)
s.view()
```

Decision Trees Regression (2)

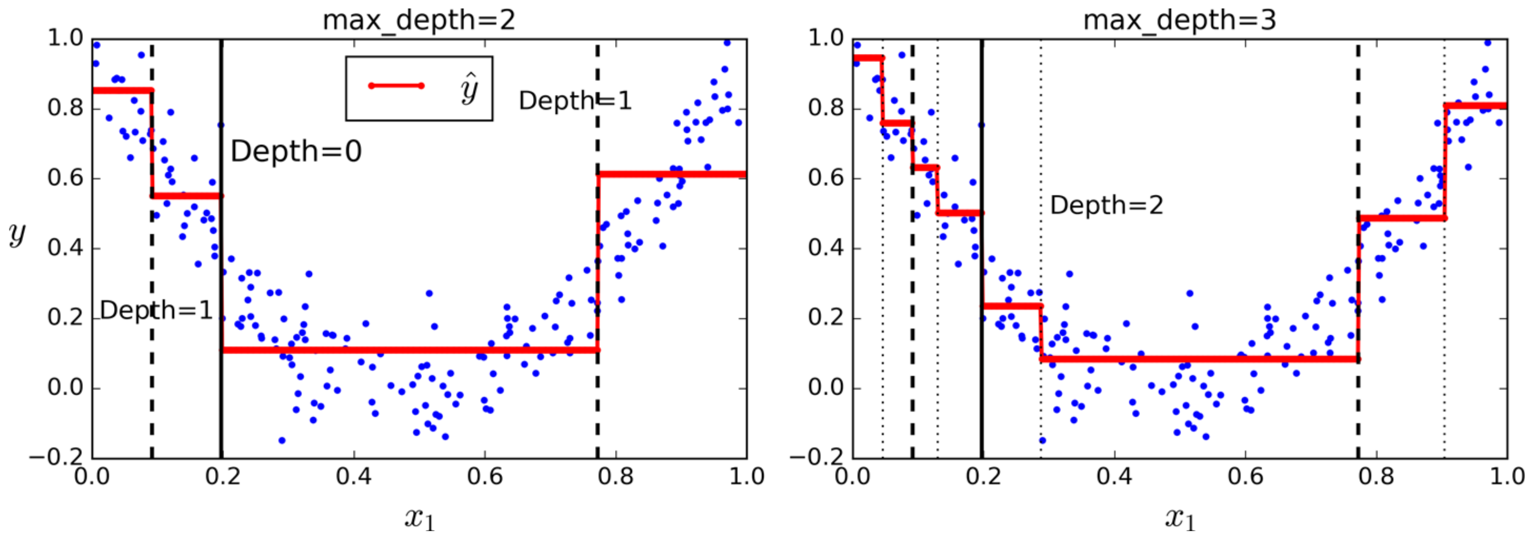


Decision Trees Regression: An example (1)

- The main difference with respect to *Decision Tree Classification* is that instead of predicting a class in each node, it predicts a value
  - Suppose we want to make a prediction for a new instance with $x_1 = 0.6$
  - We traverse the tree starting at the root, and eventually reach the leaf node that predicts *value*=0.1106
- This prediction is simply the average target value of the 110 training instances associated to this leaf node
  - This prediction results in a Mean Squared Error (MSE) equal to 0.0151 over these 110 instances

Decision Trees Regression: An example (2)

- This model's predictions are represented on the left figure
- The model on the right has *max_depth*=3 set
- Notice how the predicted value for each region is always the average target value of the instances in that region
  - The algorithm splits each region by making most training instances as close as possible to that predicted value



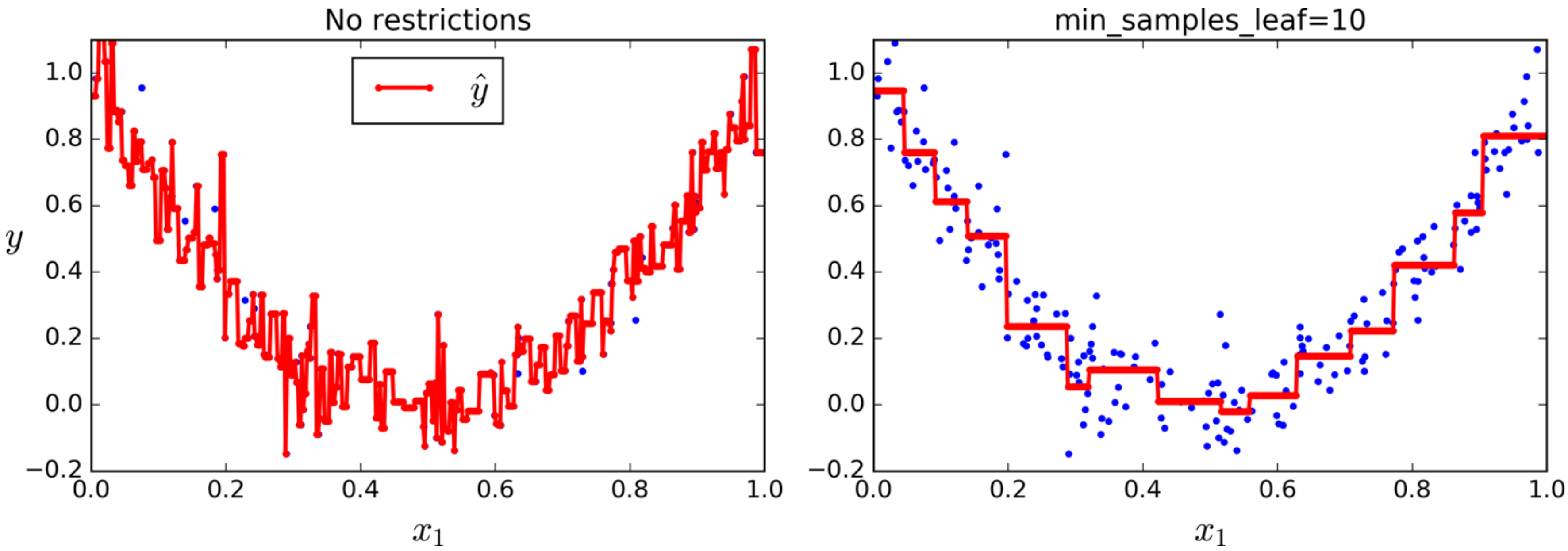
CART Algorithm for Regression

- Instead of splitting the training set aiming to minimize impurity, the CART algorithm now aims to minimize MSE
- The cost function that the algorithm tries to minimize is:

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

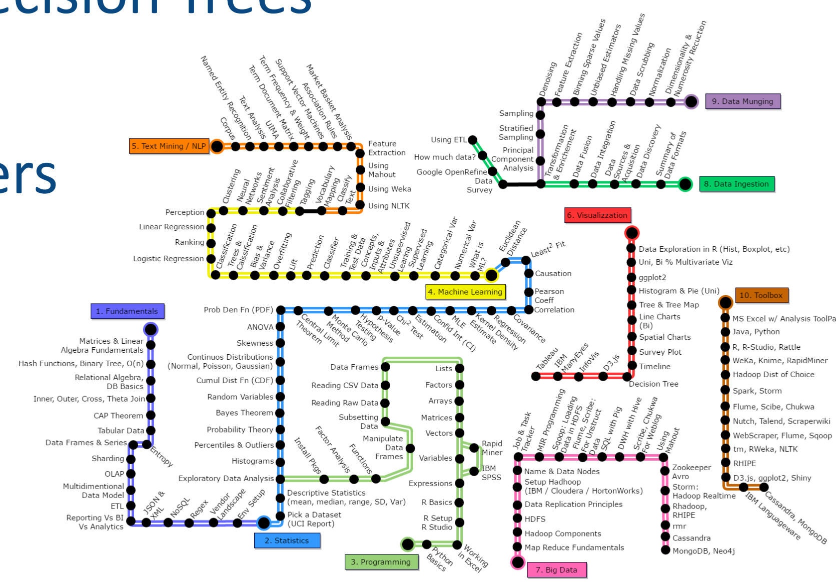
- Decision Trees are prone to overfitting when dealing with regression tasks
  - Setting *min_samples_leaf*=10 in the previous example will result in much a more reasonable model (right figure next slide)

Regularizing a Decision Tree Regressor



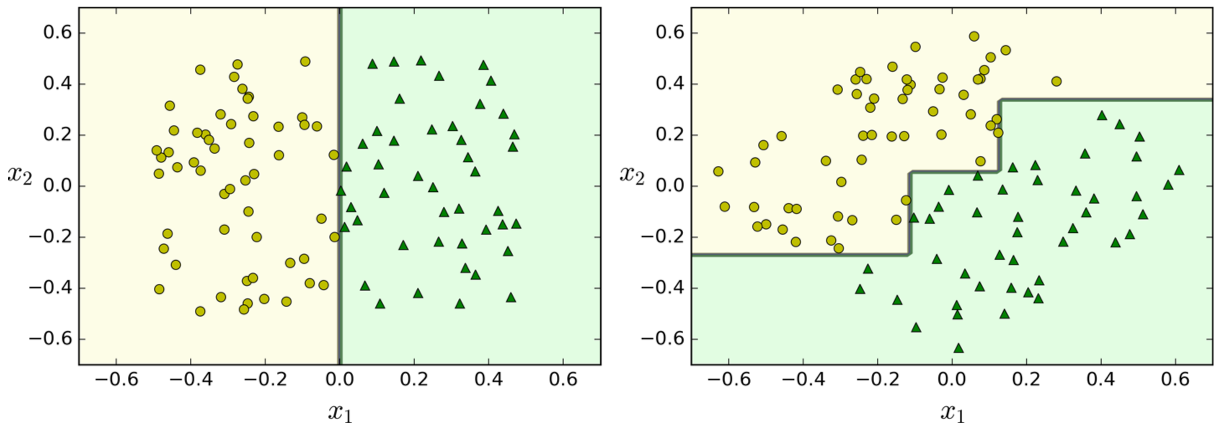
Outline

- Decision Trees
  - Training and Visualizing a Decision Tree
  - Making Predictions
  - Estimating Class Probabilities
  - Algorithms for constructing Decision Trees
  - Computational Complexity
  - Regularization Hyperparameters
- **Decision Trees Regression**
  - **Instability**



Instability (1)

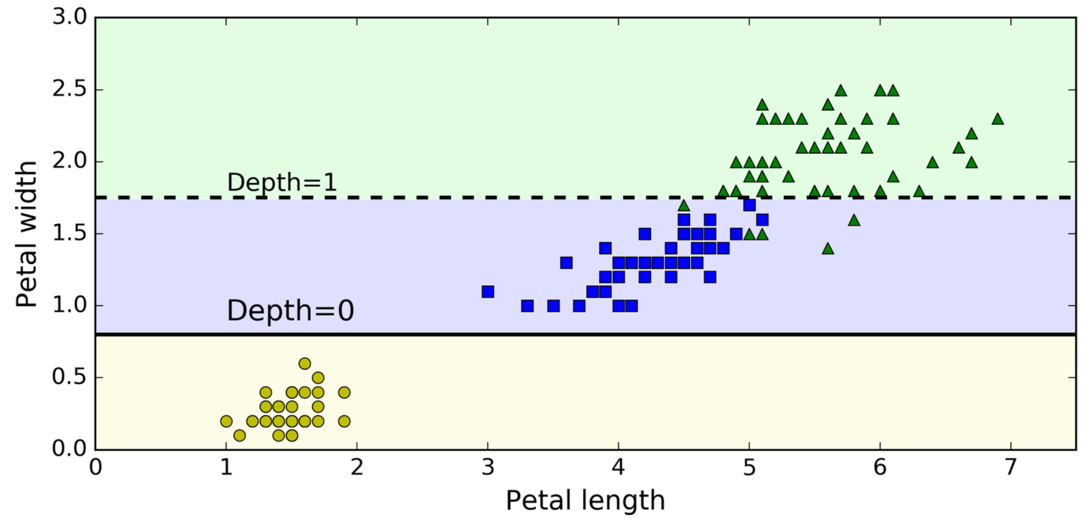
- Decision Trees have a few limitations
  - They love orthogonal decision boundaries (splits are perpendicular to an axis), making them sensitive to training set rotation
- Figure shows a simple linearly separable dataset
  - on the left, a Decision Tree can split it easily, while on the right, after a 45° dataset rotation the decision boundary looks convoluted



  - Although both fit the training set well, It is likely that the model on the right will not generalize well

Instability (2)

- The main issue with Decision Trees is that they are very sensitive to small variations in the training data
- For example, if we remove the widest Iris-Versicolor from the iris training set and train a new Decision Tree, we may get the model represented in Figure
  - since the training algorithm used by Scikit-Learn is stochastic you may get very different models even on the same training data



- Random Forests can limit this instability by averaging predictions over many trees