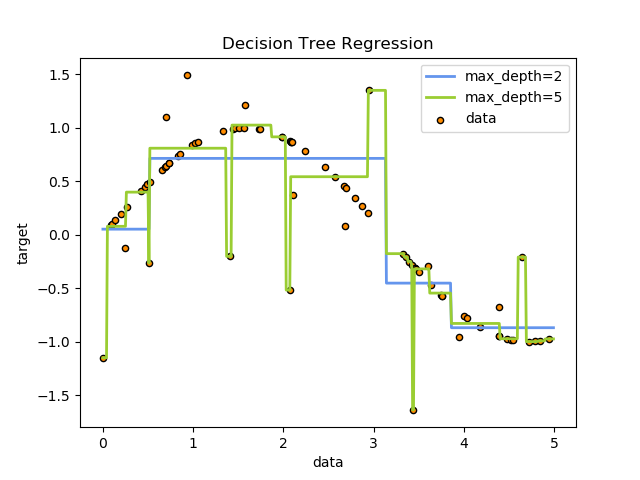
**Decision Trees (DTs)** are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

For instance, in the example below, decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.

[](http://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html)

Some advantages of decision trees are:

* Simple to understand and to interpret. Trees can be visualised.
* Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
* The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
* Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable. See algorithms for more information.
* Able to handle multi-output problems.
* Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
* Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
* Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

* Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
* Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
* The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
* There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
* Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

**1.10.1. Classification**

**DecisionTreeClassifier** is a class capable of performing multi-class classification on a dataset.

As with other classifiers, **DecisionTreeClassifier** takes as input two arrays: an array X, sparse or dense, of size [n\_samples, n\_features] holding the training samples, and an array Y of integer values, size [n\_samples], holding the class labels for the training samples:

>>>

**>>> from** **sklearn** **import** tree

**>>>** X = [[0, 0], [1, 1]]

**>>>** Y = [0, 1]

**>>>** clf = tree.DecisionTreeClassifier()

**>>>** clf = clf.fit(X, Y)

After being fitted, the model can then be used to predict the class of samples:

>>>

**>>>** clf.predict([[2., 2.]])

array([1])

Alternatively, the probability of each class can be predicted, which is the fraction of training samples of the same class in a leaf:

>>>

**>>>** clf.predict\_proba([[2., 2.]])

array([[ 0., 1.]])

**DecisionTreeClassifier** is capable of both binary (where the labels are [-1, 1]) classification and multiclass (where the labels are [0, …, K-1]) classification.

Using the Iris dataset, we can construct a tree as follows:

>>>

**>>> from** **sklearn.datasets** **import** load\_iris

**>>> from** **sklearn** **import** tree

**>>>** iris = load\_iris()

**>>>** clf = tree.DecisionTreeClassifier()

**>>>** clf = clf.fit(iris.data, iris.target)

Once trained, we can export the tree in Graphviz format using the **export\_graphviz** exporter. If you use the conda package manager, the graphviz binaries and the python package can be installed with

conda install python-graphviz

Alternatively binaries for graphviz can be downloaded from the graphviz project homepage, and the Python wrapper installed from pypi with pip install graphviz.

Below is an example graphviz export of the above tree trained on the entire iris dataset; the results are saved in an output file iris.pdf:

>>>

**>>> import** **graphviz**

**>>>** dot\_data = tree.export\_graphviz(clf, out\_file=**None**)

**>>>** graph = graphviz.Source(dot\_data)

**>>>** graph.render("iris")

The **export\_graphviz** exporter also supports a variety of aesthetic options, including coloring nodes by their class (or value for regression) and using explicit variable and class names if desired. Jupyter notebooks also render these plots inline automatically:

>>>

**>>>** dot\_data = tree.export\_graphviz(clf, out\_file=**None**,

feature\_names=iris.feature\_names,

class\_names=iris.target\_names,

filled=True, rounded=True,

special\_characters=True)

**>>>** graph = graphviz.Source(dot\_data)

**>>>** graph

After being fitted, the model can then be used to predict the class of samples:

>>>

**>>>** clf.predict(iris.data[:1, :])

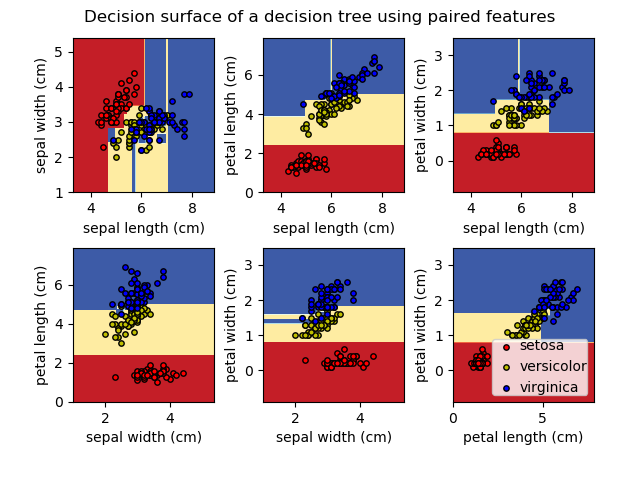
array([0])

Alternatively, the probability of each class can be predicted, which is the fraction of training samples of the same class in a leaf:

>>>

**>>>** clf.predict\_proba(iris.data[:1, :])

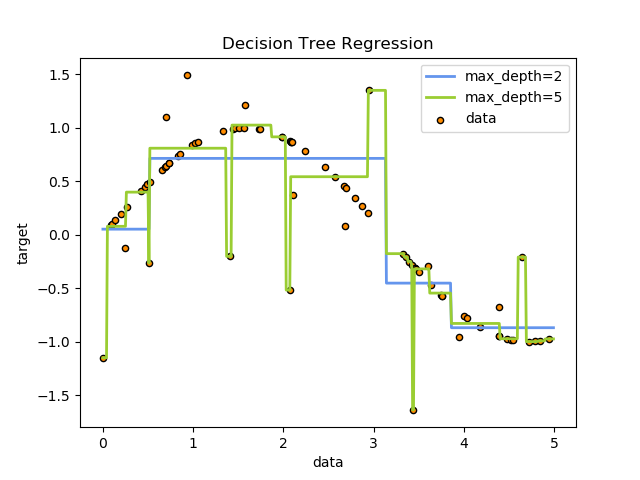
array([[ 1., 0., 0.]])

[](http://scikit-learn.org/stable/auto_examples/tree/plot_iris.html)

**Examples:**

* Plot the decision surface of a decision tree on the iris dataset

**1.10.2. Regression**

[](http://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html)