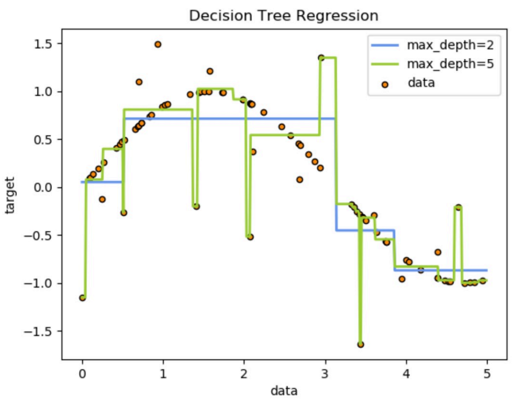
Decision Trees

Decision trees (DTs) are a non-parametric supervised learning method used for *classification* and *regression*.

- Goal: create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features



- Decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules.
- Deeper tree -> more complex decision rules and model fitting

Classification

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

`DecisionTreeClassifier` takes as input two arrays

- Array `X`, sparse or dense, of size `[n_samples, n_features]` holding the training samples
- 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)
```

Predict the class of samples:

```
>>> clf.predict([[2., 2.]])
array([1])
```

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.

Iris dataset:

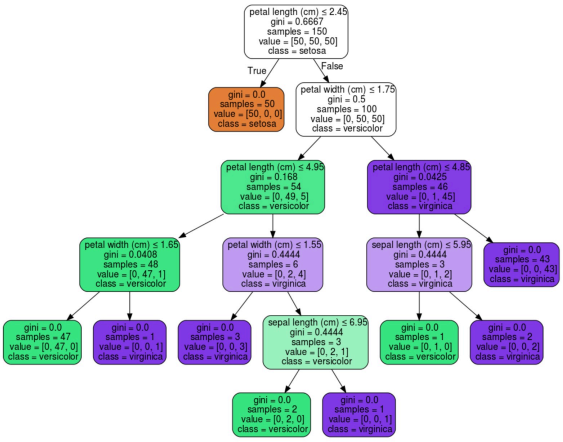
```
>>> from sklearn.datasets import load_iris
>>> from sklearn import tree
>>> iris = load_iris()
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(iris.data, iris.target)
```

Graphviz export of the above tree trained in the entire iris dataset

```
>>> 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 explicitly variable and class names if desired.

```
>>> dot_data = tree.export_grapgviz(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
```

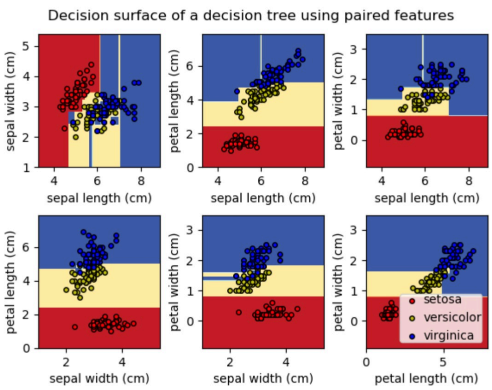


After being fitted, the model can be used to predict the class of samples of the same class in a leaf:

```
>>> clf.predict(iris.data[:1, :)
array([0])
```

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

```
>>> clf.predict_proba(iris.data[:1,:])
array([[1., 0., 0.]])
```



Parameter Tuning

`min_samples_split`: int, float, optional (default=2)

The minimum number of samples required to split an internal node:

- If int, then consider `min_samples_split` as the minimum number
  - If float, then `min_samples_split` is a percentage and ceil (`min_samples_split * n_samples`) are the minimum number of samples for each split

Entropy

Entropy controls how a DT decides where to split the data.

- Definition: measure of *impurity* in a bunch of examples

$$\boxed{H = -\sum_i p_i(\log_2 p_i)}$$

- Opposite of purity
- If all examples are same class, the entropy is 0
- If all examples are evenly split between classes, the entropy is 1.0

Information gain

Information gain = entropy(parent) - weighted averageentropy(children)

Decision tree algorithm: maximize information gain

Parameter tuning

`criterion`: string, optional (default="mini")