IT496: Introduction to Data Mining

Lecture 19

Decision Tree Classifier

(Slides are created from the book Hands-on ML by Aurelien Geron)

Arpit Rana

5th October 2023

Decision Trees

A decision tree is a representation of a function that maps a ***vector of attribute values*** to a ***single output value—a "decision."***

- Decision Trees can be used for *regression* and *classification*.
    - for binary or multiclass classification (so you don't need *one-versus-rest* or *one-versus-all*)
- They are ***more complex than linear models*** and so can better fit complex datasets.
- Many people claim that they produce ***interpretable*** models.
- ***Random Forests*** are another popular model in Machine Learning, and they contain Decision Trees.

Decision Trees

A decision tree reaches its decision by performing a sequence of tests, starting at the root and following the appropriate branch until a leaf is reached.

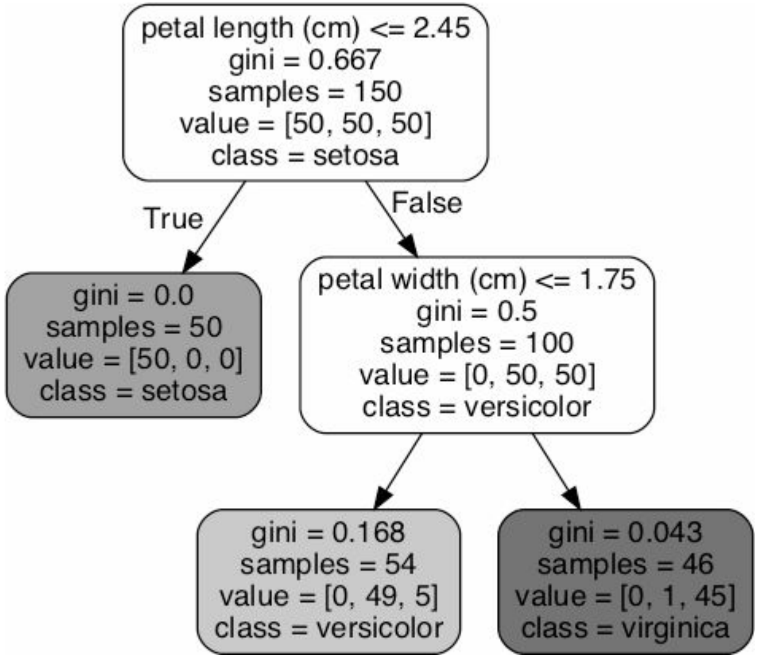
- Each *internal node* in the tree corresponds to a test of the value of one of the input attributes,

- the *branches* from the node are labeled with the possible values of the attribute, and

- the *leaf nodes* specify what value is to be returned by the function.

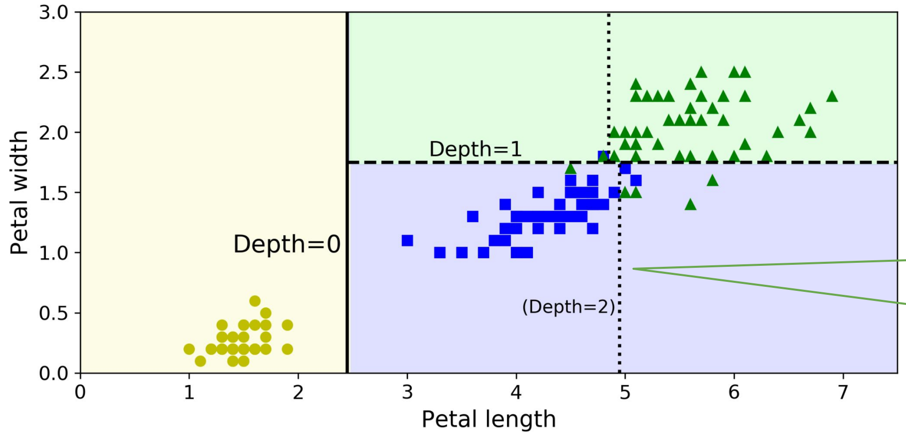
Decision Tree on the Iris Dataset

```python
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X, y)
```



Making Predictions

- What about the flower with `Petal length = 5` and `Petal width = 1.5`?
    - Start at the root: is the `Petal length <= 2.45 cm`?
    - No, so move right: is the `Petal width <= 1.75 cm`?
    - Yes, so move left: it is an *Iris-Versicolor*.



Decision Tree decision boundaries

if you set `max_depth=3`, the two depth-2 nodes would each add another decision boundary (represented by the dotted lines).

Reading the Nodes of the Tree

- `samples`: how many training examples the node applies to.
  - E.g. 100 training examples have a petal length > 2.45 cm.
- `value`: how many training examples of each class this node applies to.
  - E.g. [0, 1, 45] means 0 Iris-Sentosa, 1 Iris-Versicolor, and 45 Iris-Virginica.
- `gini`: is the *Gini impurity* of a node (with values in [0.0, 1.0]):
  - it measures how often an example would be incorrectly labeled if it was randomly labeled according to the distribution of labels for this node;
  - e.g. gini of 0 means no impurity: all examples that this node applies to belong to the same class;

Reading the Nodes of the Tree

The *Gini impurity/ Gini index* measures the impurity of X, a data partition or set of training tuples at a node, as

$$Gini(X) = 1 - \sum_{i=1}^{n} p_i^2$$

where p_i is the probability that a tuple in X belongs to the class C_i and is estimated by $|C_{i,X}|/|X|$. The sum is computed over *n* classes.

- For example, in the tree shown previously, the depth-2 left node has a `gini score` equal to $1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168$.

Learning a Decision Tree

- scikit-learn uses the CART Algorithm (Classification and Regression Tree).
  - It only *produces binary trees* (hence yes/no questions in non-leaf nodes);
  - it is *recursive* (repeats until it reaches some stopping criterion); and
  - it is *greedy* (It does not check whether or not the split will lead to the lowest possible impurity several levels down).

- There are other algorithms:
  - ID3, which can produce non-binary trees, and
  - C4.5, which is like CART but uses *entropy* in place of *Gini*, and
  - Others including ones that can directly handle nominal-valued features and missing values, which we will not study.

The CART Training Algorithm

- For each feature x_i and each value v, split the dataset into two:
  - X_{left} are the examples in X for which $x_i \leq v$
  - X_{right} are the examples in X for which $x_i > v$

  and calculate the ***CART's loss function***: $\frac{|X_{left}|}{|X|}Gini(X_{left}) + \frac{|X_{right}|}{|X|}Gini(X_{right})$

- From the above, choose the feature x_i and a value v with the lowest loss
- If a stopping criterion has been reached (e.g. maximum depth or if no split reduces impurity) then:
  - return
- Else:
  - Recursively call CART on X_{left}
  - Recursively call CART on X_{right}

Regularization Hyperparameters

- `max_depth` is a hyperparameter.
  - Increasing `max_depth` increases model's complexity and may result in overfitting.
  - If there is no constraint on tree depth, then branches will be grown until all leaves are pure.

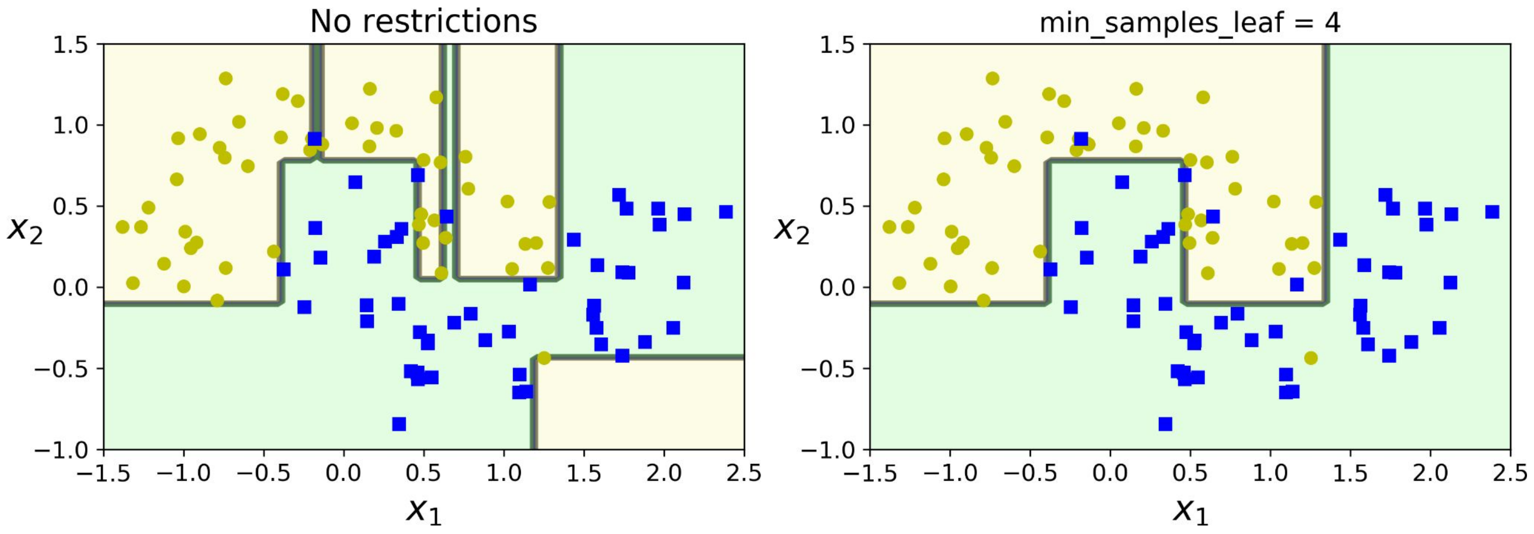
- There are other hyperparameters:
  - `min_samples_split`, the minimum number of samples a node must have before it can be split,
  - `min_samples_leaf`, the minimum number of samples a leaf node must have,
  - `max_leaf_nodes`, maximum number of leaf nodes, and
  - `max_features`, maximum number of features that are evaluated for splitting at each node.

Increasing `min_*` or reducing `max_*` hyperparameters will regularize the model.

Regularization Hyperparameters

Regularization using `min_samples_leaf`

It is quite obvious that the model on the left is overfitting, and the model on the right will probably generalize better.



Computational Complexity

- Learning is $O(mn\log_2 m)$ for the basic CART algorithm above.
  - Of course, hyperparameters such as a maximum depth can speed-up learning.
- Prediction is roughly $O(\log_2 m)$ (which is the depth of tree, assuming the tree is balanced, which it often, approximately, is). This is fast!

Regression using a Decision Tree

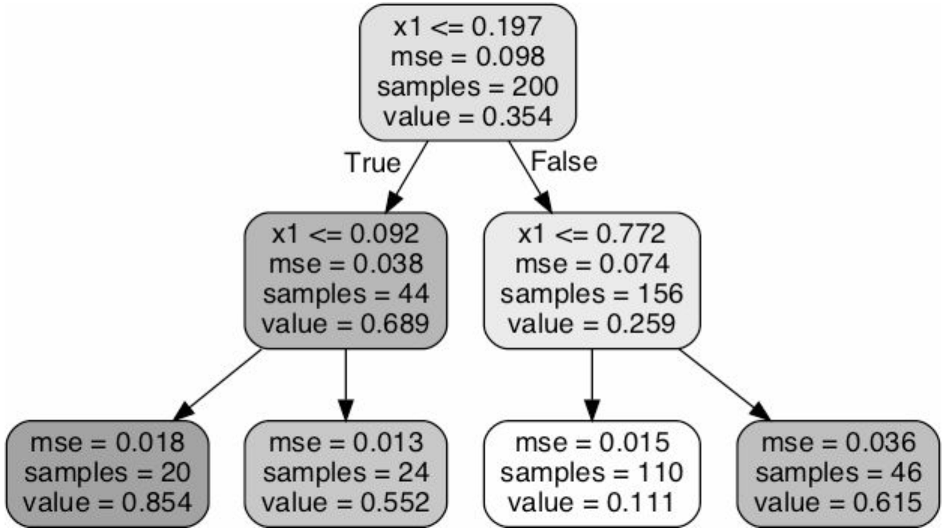
- What does the Decision Tree Regressor predict?
  - Start at the root and follow the decisions down to a leaf.
  - At any node, `value` is the prediction and is simply the mean target values of the training examples that the node applies to.

- For regression, the only difference is the loss function: in place of *Gini* it uses the ***mean squared error*** between the y-values of the training examples and their mean.

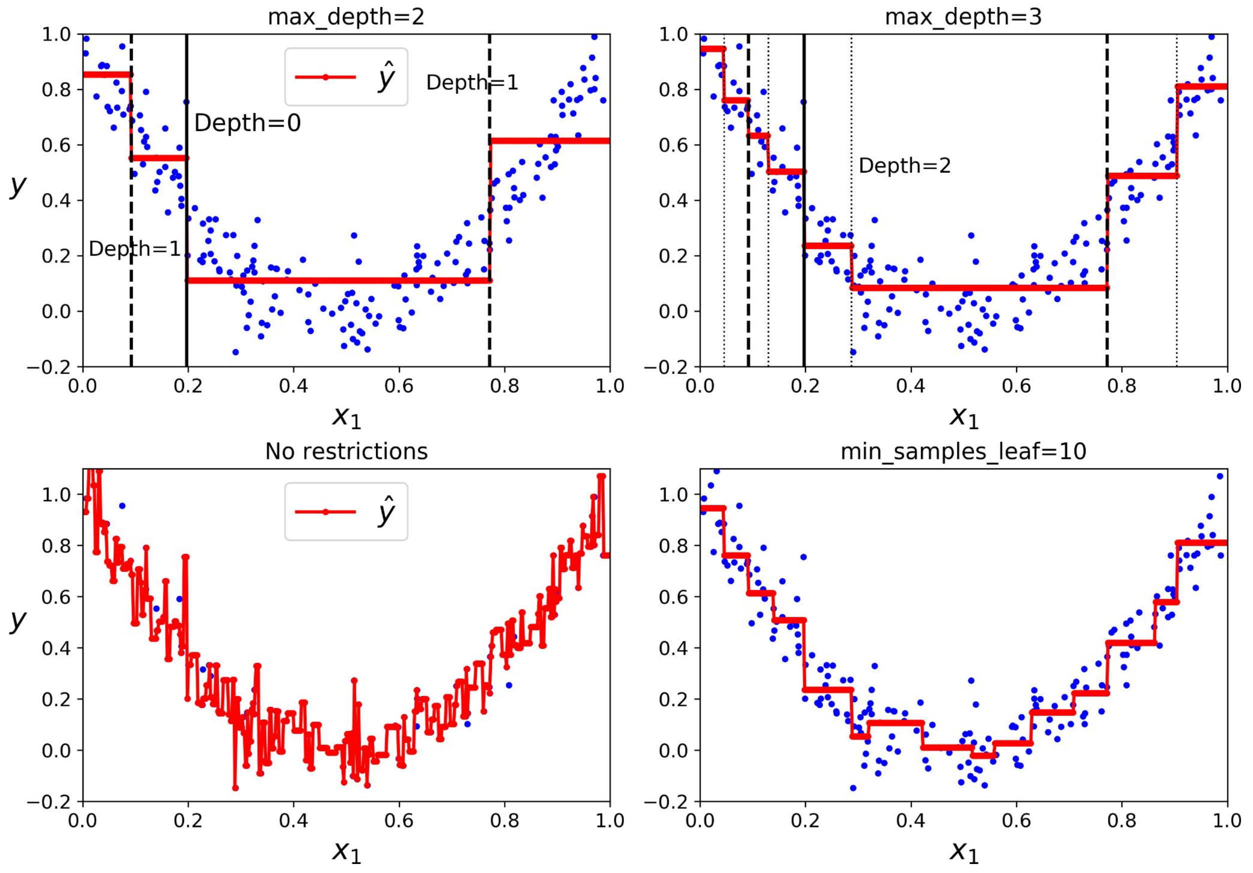
$$\frac{|X_{left}|}{|X|}MSE(X_{left}) + \frac{|X_{right}|}{|X|}MSE(X_{right})$$

Regression using a Decision Tree

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(max_depth=2)
tree_reg.fit(X, y)
```



Regression using a Decision Tree



Decision Tree: Pros and Cons

- The model is *interpretable*:
  - we can display the tree and people can see what has been learned.
- An individual prediction is *explainable*:
  - we can display the path through the tree.
- Note we do not need to scale the data before using this algorithm.
- They are very sensitive to small variations in the training data (a.k.a. *instability*). Use of PCA or Random Forests can limit this issue.

Parametric vs. Non-parametric Learning

- **Parametric learning**: the number of parameters is known prior to training and is not affected by m the number of examples in the training set.
  - E.g. Linear Regression the number of parameters is $n + 1$
  - E.g. Polynomial Regression the number of parameters is $\frac{(n+d)!}{n!d!}$

- **Non-parametric learning:** the number of parameters is not known in advance and may grow with the size of the training set.
  - E.g. For Decision Trees, in some sense, the nodes are the parameters: the structure of the model (tree) may grow to accommodate the complexity of the training data.
  - E.g. For kNN, in some sense, the neighbours are the parameters: the more training examples there are, the more different possible sets of neighbours there are.

Parametric vs. Non-parametric Learning

This does not mean that non-parametric models will have lower validation error.

- Unconstrained, they are prone to overfitting.
- So we want to impose some constraints on the CART algorithm to restrict the shape of the Decision Tree (such as maximum depth and others).
- Similarly, we avoid small values for k in kNN.

Next lecture

Dimensionality Reduction

6th October 2023