Scikit-Learn: Decision Trees

- Decision tree classifier:
  sklearn.tree.DecisionTreeClassifier(criterion=gini, splitter=best,
        max_depth=None, min_samples_split=2, min_samples_leaf=1,
        min_weight_fraction_leaf=0.0, max_features=None, random_state=None,
        max_leaf_nodes=None, min_impurity_decrease=0.0,
        min_impurity_split=None, class_weight=None, presort=False)

- Example decision tree (output by graphviz):

  ![Iris Decision Tree](image)

  * Parameters:
    * criterion (string): Function used to determine a split
      · 'gini' or 'entropy'
    * splitter (string): strategy used to choose split
      · 'best' or 'random'
    * max_depth (int)
      · The maximum depth of the tree
      · If None, nodes are expanded until all leaves are pure or
        until all leaves contain less than min_samples_split samples
Scikit-Learn: Decision Trees (2)

* `min_samples_split` (int, float)
  · 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 fraction and `ceil(min_samples_split* n_samples)` are the minimum number of samples for each split

* `min_samples_leaf` (int, float)
  · Minimum number of samples required to be at a leaf node
  · A split point at any depth will only be considered if it leaves at least `min_samples_leaf` training samples in each of the left and right branches
  · Calculated as above

* `min_weight_fraction_leaf` (float)
  · Minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node
  · Samples have equal weight when `sample_weight` is not provided

* `max_features` (int, float, string)
  · number of features to consider when looking for the best split
  · `int`: consider `max_features` features at each split
  · `float`: `max_features` is a fraction and `int(max_features*n_features)` features are considered at each split
  · `'auto'`: `max_features = sqrt(n_features)`
  · `'sqrt'`: `max_features = sqrt(n_features)`
  · `'log2'`: `max_features = log2(n_features)`
  · `None`: `max_features = n_features`

* `max_leaf_nodes` (int)
  · Grow a tree with `max_leaf_nodes` in best-first fashion
  · Best nodes are defined as relative reduction in impurity
  · `None`: unlimited number of leaf nodes
Scikit-Learn: Decision Trees (3)

* min_impurity_decrease (float)
  · Node will be split if this split induces a decrease of the impurity greater than or equal to this value
  · Weighted impurity decrease equation:
    \[
    \frac{N_t}{N} \left( \text{impurity} - \frac{N_tR}{N_t} \text{right_impurity} - \frac{N_tL}{N_t} \text{left_impurity} \right)
    \]
    where
    - \( N \) is total number of samples
    - \( N_t \) is number of samples at the current node
    - \( N_tL \) is number of samples in the left child
    - \( N_tR \) is number of samples in the right child
  · All refer to the weighted sum if sample_weight is passed

* class_weight (dictionary, list of dictionaries, 'balanced')
  · Weights associated with classes
  · If None, all assumed = 1
  · 'balanced': uses the values of \( y \) to automatically adjust weights inversely proportional to class frequencies in the input data as \( \frac{n_{samples}}{(n_{classes} \times \text{np.bincount}(y))} \) where \( y \) is a feature

− Methods:
  * apply(X[, check_input])
    · check_input: recommended not to use
    · Returns, for each datapoint \( x \) in \( X \), the index of the leaf \( x \) ends up at
  * decision_path(X[, check_input])
    · Return the decision path in the tree as a matrix where non zero elements indicates that the samples goes through the nodes

− Attributes:
  * classes_ (array, list of arrays): Class labels
  * feature_importances_ (int): inferred value of max_features
  * max_features_ (int): inferred value of max_features
  * n_classes_ (int, list): Number of classes
  * n_features_ (int): number of features when fit performed
  * n_outputs_ (int): number of outputs when fit performed
  * tree_ (tree object): The tree
– Regularization
  * Decreasing any of the $max_-$ parameters or increasing any of the $min_-$ parameters will regularize the model
from sklearn import tree

- Binary tree represented as a number of parallel arrays
  
  * Each array holds info about node \( i \)
  * Node 0 is root

- Attributes:
  
  * `node_count` (int): Number of nodes in tree
  * `capacity` (int): Current capacity of arrays, \( \geq \) `node_count`
  * `children_left` (array of ints): `children_left[i] > i` holds the node id of the left child of node \( i \)
    
    - For a leaf, value is `TREE_LEAF`
  * `children_right` (array of ints): as above
  * `feature` (array of ints): `feature[i]` holds the feature to split on
  * `threshold` (array of double): `threshold[i]` holds the threshold for the internal node \( i \)
  * `value` (array of double): Contains the constant prediction value of each node
  * `impurity` (array of double): holds the impurity (value of the splitting criterion) at node \( i \)
  * `n_node_samples` (array of ints): `n_node_samples[i]` holds the number of training samples reaching node \( i \)
  * `weighted_n_node_samples` (array of ints): `weighted_n_node_samples[i]` holds the weighted number of training samples reaching node \( i \)

To visualize a decision tree, you can use the assorted methods and attributes to manually create a textual representation.

The standard approach is to use the package `graphviz`.

This is *not* part of Python and must be installed separately.

*Graphviz* is a package for creating visualizations.

It creates images from files of data (e.g., `.dot` files)

To create a `dot` file from sklearn:

- Use `export_graphviz`

  ```python
  sklearn.tree.export_graphviz(decision_tree, out_file=None, max_depth=None, feature_names=None, class_names=None, label=all, filled=False, leaves_parallel=False, impurity=True, node_ids=False, proportion=False, rotate=False, rounded=False, special_characters=False, precision=3)
  ```

- Parameters
  * `decision_tree` (Tree): the decision tree
  * `out_file` (file object, string): output file name
  * `max_depth` (int): Max depth to generate; `None` generates whole tree
  * `feature_names` (list of strings)
  * `class_names` (list of strings, Boolean): Names of each of the target classes in ascending numerical order
    - Only relevant for classification
    - If `True`, shows a symbolic representation of the class name
  * `label` (string): Whether to show informative labels for impurity
    - Values: ’all’, ’root’, or ’none’
  * `filled` (Boolean): Paint nodes to indicate majority class for classification, extremity of values for regression, or purity of node for multi-output
  * `leaves_parallel` (Boolean): Draw all leaf nodes at the bottom of the tree.
  * `impurity` (Boolean): Show impurity at each node
  * `node_ids` (Boolean): Show ID number at each node
  * `proportion` (Boolean): Change the display of values and/or samples to be proportions and percentages respectively
Scikit-Learn: Decision Trees - Visualizing (2)

* `rotate` (Boolean): Orient tree left to right rather than top-down
* `rounded` (Boolean): Draw node boxes with rounded corners and use Helvetica fonts instead of Times-Roman
* `precision` (int): Number of digits of precision for floating point in the values of impurity, threshold and value attributes of each node
  - Returns String representation of the input tree in GraphViz dot format if `out_file` is `None`

- To convert the result to a png file from the command line, use:
  `dot -Tpng in_file_name.dot -o out_file_name.png`
Scikit-Learn: Decision Trees - The CART Algorithm

- The CART (Classification and Regression Trees) algorithm is similar to the C4.5 algorithm (the successor to ID3) with the following differences:
  1. Supports numerical target variables (regression)
  2. Does not compute rule sets

- Scikit’s implementation does not support categorical variables (currently)

- The algorithm:
  1. Split training set into two subsets based on feature \( k \) and threshold \( t_k \)
  2. \((k, t_k)\) chosen to produce the purest subsets (see below)
  3. Cost function:
     \[
     J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right}
     \]
     where
     \[
     G_{left/right} \text{ measures the impurity of the left/right subset}
     \]
     \[
     m_{left/right} \text{ the number of instances in the left/right subset}
     \]

- The algorithm is recursive, where base case is when either
  - The max depth is reached (\texttt{max\_depth}), OR
  - A split cannot be found that will reduce impurity

- Impurity
  - A node is pure if all training instances it applies to are of the same class
  - This is measured by its gini score:
    \[
    G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2
    \]
    where
    \[
    p_{i,k} = \frac{|k\_instances|}{|all\_instances|} \text{ at node } i
    \]
    - A score of zero indicates a pure node

- An alternative to the gini score is entropy:
  \[
  H_i = - \sum_{k=1}^{n} p_{i,k} \log(p_{i,k})
  \]
  where
  \[
  p_{i,k} \neq 0
  \]
Scikit-Learn: Regression Trees

- Use
  sklearn.tree.DecisionTreeRegressor(criterion=mse, splitter=best, max_depth=None,
  min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
  max_features=None, random_state=None, max_leaf_nodes=None,
  min_impurity_decrease=0.0, min_impurity_split=None, presort=False)

  - This classifier predicts a value instead of a class
  - Sample tree:

  ![Decision Tree](image)

  * Predicted value is average target value for instances associated with a node
  * Samples split so as to make the greatest number as close as possible to the predicted value
Scikit-Learn: Regression Trees (2)

- Example prediction graphs:

![Figure 6-5. Predictions of two Decision Tree regression models]

- Parameters:
  * criterion (string): Function to measure the quality of a split
    - ‘mse’ or ‘friedman_mse’ or ‘mae’
  * Rest same as DecisionTreeClassifier

- Revised CART algorithm
  - Splits now based on MSE rather than impurity
    \[
    J(k, t_k) = \frac{m_{left}}{m} MSE_{left} + \frac{m_{right}}{m} MSE_{right}
    \]
    where
    \[
    MSE_{node} = \sum_{i \in node} (\hat{y}_{node} - y_i)^2 \hat{y}_{node} = \frac{1}{m_{node}} \sum_{i \in node} (y_i)
    \]