Name: - Aditya Gavankar

Roll no: - J072

Topic: - Machine Learning **Assignment**: - 8

Sci-kit learn API – Decision Tree

- **Decision Trees** 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.
- A tree can be seen as a piecewise constant approximation.

Code:-

*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, ccp_alpha=0.0)*

Parameter:-

- **criterion*{"gini", "entropy"}, default="gini"***
  The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

- **splitter*{"best", "random"}, default="best"***
  The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

- **max_depth(*int), default=None***
  The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

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

- **min_samples_leaf(*int or float), default=1***
  The 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. This may have the effect of smoothing the model, especially in regression.
  - If int, then consider min_samples_leaf as the minimum number.
  - If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

- **min_weight_fraction_leaf(*float), default=0.0***
  The 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 or {"auto", "sqrt", "log2"}), default=None***
  The number of features to consider when looking for the best split:
  - If int, then consider max_features features at each split.
  - If float, then max_features is a fraction and int(max_features * n_features) features are considered at each split.
  - If "auto", then max_features=sqrt(n_features).
  - If "sqrt", then max_features=sqrt(n_features).
  - If "log2", then max_features=log2(n_features).
  - If None, then max_features=n_features.

  Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_features features.

- **random_state(*int, RandomState instance or None), default=None***
  Controls the randomness of the estimator. The features are always randomly permuted at each split, even if splitter is set to "best". When max_features < n_features, the algorithm will select max_features at random at each split before finding the best split among them. But the best found split may vary across different runs, even if max_features=n_features. That is the case, if the improvement of the criterion is identical for several splits and one split has to be selected at random. To obtain a deterministic behaviour during fitting, random_state has to be fixed to an integer.

- **max_leaf_nodes(*int), default=None***
  Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

- **min_impurity_decrease(*float), default=0.0***
  A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
  The weighted impurity decrease equation is the following:
  N_t / N * (impurity - N_t_R / N_t * right_impurity - N_t_L / N_t * left_impurity)
  - where N is the total number of samples, N_t is the number of samples at the current node, N_t_L is the number of samples in the left child, and N_t_R is the number of samples in the right child.
  - N, N_t, N_t_R and N_t_L all refer to the weighted sum, if sample_weight is passed.

- **min_impurity_split(*float), default=0***
  Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.

- **class_weight(*dict, list of dict or "balanced"), default=None***
  Weights associated with classes in the form {class_label: weight}. If None, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.
  Note that for multioutput (including multilabel) weights should be defined for each class of every column in its own dict. For example, for four-class multilabel classification weights should be [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] instead of [{1:1}, {2:5}, {3:1}, {4:1}].

The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)) For multi-output, the weights of each column of y will be multiplied.
Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

- **ccp_alpha(*non-negative float), default=0.0***
  Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.

Attributes:-

- **classes_*ndarray of shape (n_classes,) or list of ndarray***
  The classes labels (single output problem), or a list of arrays of class labels (multi-output problem).

- ***feature_importances_ndarray of shape (n_features,)***
  Return the feature importances.

- **max_features_*int***
  The inferred value of max_features.

- **n_classes_*int or list of int***
  The number of classes (for single output problems), or a list containing the number of classes for each output (for multi-output problems).

- **n_features_*int***
  The number of features when fit is performed.

- **n_outputs_*int***
  The number of outputs when fit is performed.

- **tree_*Tree instance***
  The underlying Tree object.

Methods:-

apply(*X*, *check_input=True*)	Return the index of the leaf that each sample is predicted as.
cost_complexity_pruning_path(*X*, *y*, *sample_weight=None*)	Compute the pruning path during Minimal Cost-Complexity Pruning.
decision_path(*X*, *check_input=True*)	Return the decision path in the tree.
fit(*X*, *y*, *sample_weight=None*, *check_input=True*, *X_idx_sorted='deprecated'*)	Build a decision tree classifier from the training set (X, y).
get_depth()	Return the depth of the decision tree.
get_n_leaves()	Return the number of leaves of the decision tree.
get_params(*deep=True*)	Get parameters for this estimator.

predict(*X*, *check_input=True*)	Predict class or regression value for X.
predict_log_proba(X)	Predict class log-probabilities of the input samples X.
predict_proba(*X*, *check_input=True*)	Predict class probabilities of the input samples X.
score(*X*, *y*, *sample_weight=None*)	Return the mean accuracy on the given test data and labels.
set_params(**params)	Set the parameters of this estimator.