Chem 277B Spring 2024 Tutorial 12

Outline

1. Decision Tree methods
2. Final Project:

- RMSE vs MAE
- Checkpoint 3
- Guidelines for final report + notebook

Decision Tree

- Documentation

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
```

```
/var/folders/k8/mg372j_55z30k1z4y_8mb0w00000gn/T/ipykernel_67183/4272825579.
py:2: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major releas
e of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and bet
ter interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54
466

  import pandas as pd
```

In [ ]:
```
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
print("Feature names:", feature_names)
print("Target names:", target_names)
```

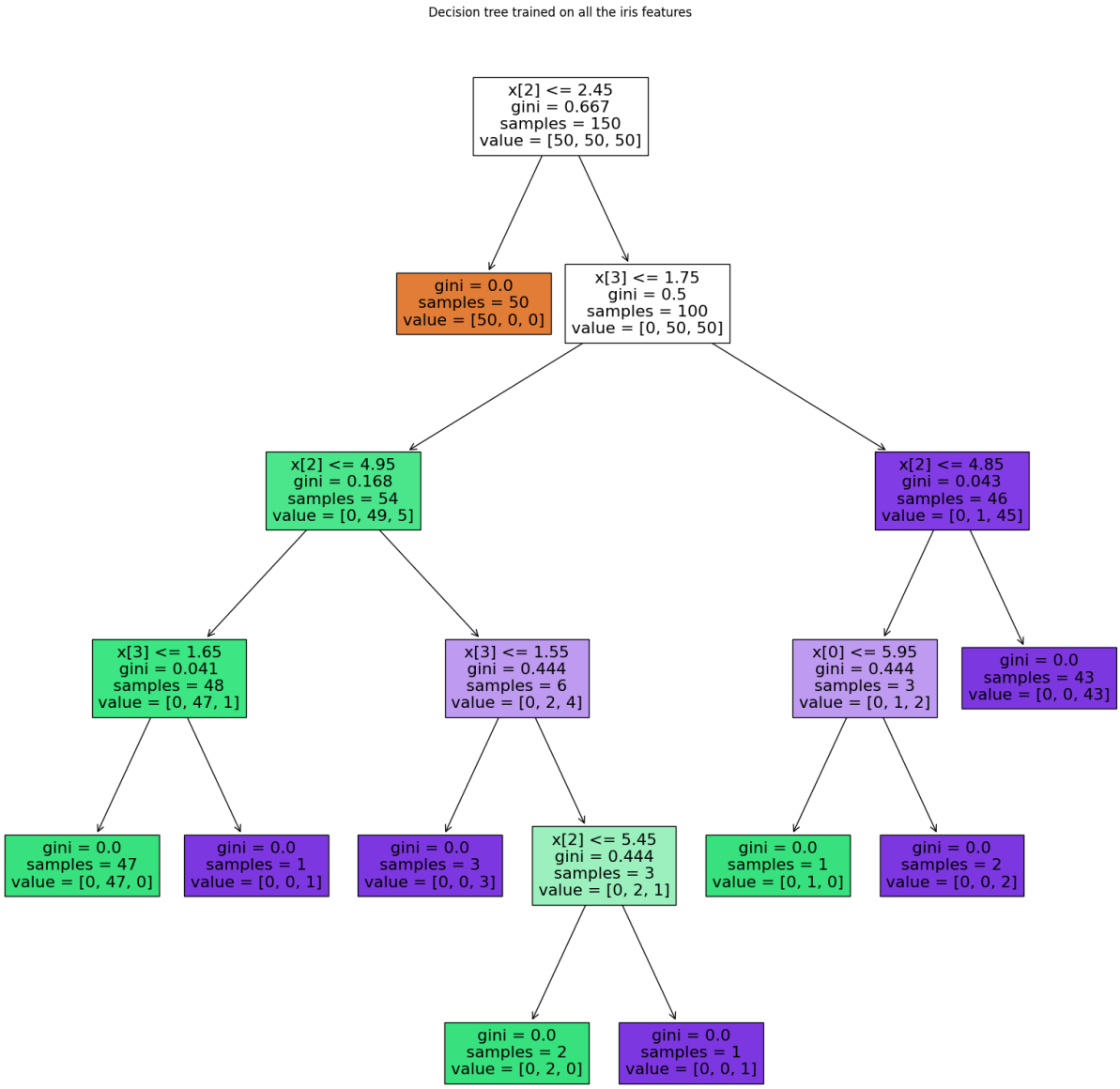
```
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (c
m)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']
```

In [ ]:
```
# define
clf = DecisionTreeClassifier()
# fit
clf.fit(X, y)

# predict
y_pred = clf.predict(X)

# accuracy
accuracy_score(y, y_pred)
```

Out[ ]: 1.0

Plot the decision process

In [ ]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(20, 20))
plot_tree(clf, filled=True)
plt.title("Decision tree trained on all the iris features")
plt.show()
```



Random Forest

- Documentation

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. This is the most frequently used method. Sometimes it is even better than complicated neural network models.

Let's play with a more challenging dataset.

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
def load_wines(path):
    df = pd.read_csv(path)
```

```
    X = df.iloc[:, :-2].values
    y = df.iloc[:, -1].values
    return X, y

X, y = load_wines("wines.csv")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [ ]:
```
dt = DecisionTreeClassifier().fit(X_train, y_train)

print("Decision Tree Accuracy (train):", accuracy_score(y_train, dt.predict(
print("Decision Tree Accuracy (test):", accuracy_score(y_test, dt.predict(X_
```

```
Decision Tree Accuracy (train): 1.0
Decision Tree Accuracy (test): 0.9444444444444444
```

In [ ]:
```
rf = RandomForestClassifier().fit(X_train, y_train)

print("Random Forest Accuracy (train):", accuracy_score(y_train, rf.predict(
print("Random Forest Accuracy (test):", accuracy_score(y_test, rf.predict(X_
```

```
Random Forest Accuracy (train): 1.0
Random Forest Accuracy (test): 0.9722222222222222
```

Play around with hyperparameters?

In [ ]:
```
help(DecisionTreeClassifier)
```

```
Help on class DecisionTreeClassifier in module sklearn.tree._classes:

class DecisionTreeClassifier(sklearn.base.ClassifierMixin, BaseDecisionTree)
 |  DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=N
one, 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_decr
ease=0.0, class_weight=None, ccp_alpha=0.0, monotonic_cst=None)
 |
 |  A decision tree classifier.
 |
 |  Read more in the :ref:`User Guide <tree>`.
 |
 |  Parameters
 |  ----------
 |  criterion : {"gini", "entropy", "log_loss"}, default="gini"
 |      The function to measure the quality of a split. Supported criteria a
re
 |      "gini" for the Gini impurity and "log_loss" and "entropy" both for t
he
 |      Shannon information gain, see :ref:`tree_mathematical_formulation`.
 |
 |  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 choos
e
 |      the best random split.
 |
 |  max_depth : int, default=None
 |      The maximum depth of the tree. If None, then nodes are expanded unti
l
 |      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.
 |
 |      .. versionchanged:: 0.18
 |         Added float values for fractions.
 |
 |  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.
 |
```

```
 |      .. versionchanged:: 0.18
 |         Added float values for fractions.
 |
 |  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 {"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
 |            `max(1, int(max_features * n_features_in_))` features are cons
idered at
 |            each split.
 |          - 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 b
est
 |      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 behavi
our
 |      during fitting, ``random_state`` has to be fixed to an integer.
 |      See :term:`Glossary <random_state>` for details.
 |
 |  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 impurit
y
 |      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 t
he
```

```
 |      left child, and ``N_t_R`` is the number of samples in the right chil
d.
 |
 |      ``N``, ``N_t``, ``N_t_R`` and ``N_t_L`` all refer to the weighted su
m,
 |      if ``sample_weight`` is passed.
 |
 |      .. versionadded:: 0.19
 |
 |  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 dat
a
 |      as ``n_samples / (n_classes * np.bincount(y))``
 |
 |      For multi-output, the weights of each column of y will be multiplie
d.
 |
 |      Note that these weights will be multiplied with sample_weight (passe
d
 |      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. S
ee
 |      :ref:`minimal_cost_complexity_pruning` for details.
 |
 |      .. versionadded:: 0.22
 |
 |  monotonic_cst : array-like of int of shape (n_features), default=None
 |      Indicates the monotonicity constraint to enforce on each feature.
 |        - 1: monotonic increase
 |        - 0: no constraint
 |        - -1: monotonic decrease
 |
 |      If monotonic_cst is None, no constraints are applied.
 |
 |      Monotonicity constraints are not supported for:
 |        - multiclass classifications (i.e. when `n_classes > 2`),
 |        - multioutput classifications (i.e. when `n_outputs_ > 1`),
 |        - classifications trained on data with missing values.
 |
```

```
|      The constraints hold over the probability of the positive class.
|
|      Read more in the :ref:`User Guide <monotonic_cst_gbdt>`.
|
|      .. versionadded:: 1.4
|
|  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,)
|      The impurity-based feature importances.
|      The higher, the more important the feature.
|      The importance of a feature is computed as the (normalized)
|      total reduction of the criterion brought by that feature.  It is als
o
|      known as the Gini importance [4]_.
|
|      Warning: impurity-based feature importances can be misleading for
|      high cardinality features (many unique values). See
|      :func:`sklearn.inspection.permutation_importance` as an alternative.
|
|  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_in_ : int
|      Number of features seen during :term:`fit`.
|
|      .. versionadded:: 0.24
|
|  feature_names_in_ : ndarray of shape (`n_features_in_`,)
|      Names of features seen during :term:`fit`. Defined only when `X`
|      has feature names that are all strings.
|
|      .. versionadded:: 1.0
|
|  n_outputs_ : int
|      The number of outputs when ``fit`` is performed.
|
|  tree_ : Tree instance
|      The underlying Tree object. Please refer to
|      ``help(sklearn.tree._tree.Tree)`` for attributes of Tree object and
|      :ref:`sphx_glr_auto_examples_tree_plot_unveil_tree_structure.py`
|      for basic usage of these attributes.
|
|  See Also
|  --------
|  DecisionTreeRegressor : A decision tree regressor.
|
```

```
 |  Notes
 |  -----
 |  The default values for the parameters controlling the size of the trees
 |  (e.g. ``max_depth``, ``min_samples_leaf``, etc.) lead to fully grown and
 |  unpruned trees which can potentially be very large on some data sets. To
 |  reduce memory consumption, the complexity and size of the trees should b
e
 |  controlled by setting those parameter values.
 |
 |  The :meth:`predict` method operates using the :func:`numpy.argmax`
 |  function on the outputs of :meth:`predict_proba`. This means that in
 |  case the highest predicted probabilities are tied, the classifier will
 |  predict the tied class with the lowest index in :term:`classes_`.
 |
 |  References
 |  ----------
 |
 |  .. [1] https://en.wikipedia.org/wiki/Decision_tree_learning
 |
 |  .. [2] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification
 |         and Regression Trees", Wadsworth, Belmont, CA, 1984.
 |
 |  .. [3] T. Hastie, R. Tibshirani and J. Friedman. "Elements of Statistica
l
 |         Learning", Springer, 2009.
 |
 |  .. [4] L. Breiman, and A. Cutler, "Random Forests",
 |         https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
 |
 |  Examples
 |  --------
 |  >>> from sklearn.datasets import load_iris
 |  >>> from sklearn.model_selection import cross_val_score
 |  >>> from sklearn.tree import DecisionTreeClassifier
 |  >>> clf = DecisionTreeClassifier(random_state=0)
 |  >>> iris = load_iris()
 |  >>> cross_val_score(clf, iris.data, iris.target, cv=10)
 |  ...                             # doctest: +SKIP
 |  ...
 |  array([ 1.     ,  0.93...,  0.86...,  0.93...,  0.93...,
 |          0.93...,  0.93...,  1.     ,  0.93...,  1.      ])
 |
 |  Method resolution order:
 |      DecisionTreeClassifier
 |      sklearn.base.ClassifierMixin
 |      BaseDecisionTree
 |      sklearn.base.MultiOutputMixin
 |      sklearn.base.BaseEstimator
 |      sklearn.utils._estimator_html_repr._HTMLDocumentationLinkMixin
 |      sklearn.utils._metadata_requests._MetadataRequester
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, criterion='gini', splitter='best', max_depth=None, min
_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_feat
```

```
ures=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.
0, class_weight=None, ccp_alpha=0.0, monotonic_cst=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  fit(self, X, y, sample_weight=None, check_input=True)
 |      Build a decision tree classifier from the training set (X, y).
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The training input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csc_matrix``.
 |
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          The target values (class labels) as integers or strings.
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Sample weights. If None, then samples are equally weighted. Spli
ts
 |          that would create child nodes with net zero or negative weight a
re
 |          ignored while searching for a split in each node. Splits are als
o
 |          ignored if they would result in any single class carrying a
 |          negative weight in either child node.
 |
 |      check_input : bool, default=True
 |          Allow to bypass several input checking.
 |          Don't use this parameter unless you know what you're doing.
 |
 |      Returns
 |      -------
 |      self : DecisionTreeClassifier
 |          Fitted estimator.
 |
 |  predict_log_proba(self, X)
 |      Predict class log-probabilities of the input samples X.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csr_matrix``.
 |
 |      Returns
 |      -------
 |      proba : ndarray of shape (n_samples, n_classes) or list of n_outputs
such arrays if n_outputs > 1
 |          The class log-probabilities of the input samples. The order of t
he
 |          classes corresponds to that in the attribute :term:`classes_`.
 |
 |  predict_proba(self, X, check_input=True)
 |      Predict class probabilities of the input samples X.
```

```
 |
 |      The predicted class probability is the fraction of samples of the sa
me
 |      class in a leaf.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csr_matrix``.
 |
 |      check_input : bool, default=True
 |          Allow to bypass several input checking.
 |          Don't use this parameter unless you know what you're doing.
 |
 |      Returns
 |      -------
 |      proba : ndarray of shape (n_samples, n_classes) or list of n_outputs
such arrays if n_outputs > 1
 |          The class probabilities of the input samples. The order of the
 |          classes corresponds to that in the attribute :term:`classes_`.
 |
 |  set_fit_request(self: sklearn.tree._classes.DecisionTreeClassifier, *, c
heck_input: Union[bool, NoneType, str] = '$UNCHANGED$', sample_weight: Union
[bool, NoneType, str] = '$UNCHANGED$') -> sklearn.tree._classes.DecisionTree
Classifier
 |      Request metadata passed to the ``fit`` method.
 |
 |      Note that this method is only relevant if
 |      ``enable_metadata_routing=True`` (see :func:`sklearn.set_config`).
 |      Please see :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      The options for each parameter are:
 |
 |      - ``True``: metadata is requested, and passed to ``fit`` if provide
d. The request is ignored if metadata is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will n
ot pass it to ``fit``.
 |
 |      - ``None``: metadata is not requested, and the meta-estimator will r
aise an error if the user provides it.
 |
 |      - ``str``: metadata should be passed to the meta-estimator with this
given alias instead of the original name.
 |
 |      The default (``sklearn.utils.metadata_routing.UNCHANGED``) retains t
he
 |      existing request. This allows you to change the request for some
 |      parameters and not others.
 |
 |      .. versionadded:: 1.3
 |
 |      .. note::
```

```
 |          This method is only relevant if this estimator is used as a
 |          sub-estimator of a meta-estimator, e.g. used inside a
 |          :class:`~sklearn.pipeline.Pipeline`. Otherwise it has no effect.
 |
 |      Parameters
 |      ----------
 |      check_input : str, True, False, or None,                    default
=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``check_input`` parameter in ``fit``.
 |
 |      sample_weight : str, True, False, or None,                    defau
lt=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``sample_weight`` parameter in ``fit``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
 |
 |  set_predict_proba_request(self: sklearn.tree._classes.DecisionTreeClassi
fier, *, check_input: Union[bool, NoneType, str] = '$UNCHANGED$') -> sklear
n.tree._classes.DecisionTreeClassifier
 |      Request metadata passed to the ``predict_proba`` method.
 |
 |      Note that this method is only relevant if
 |      ``enable_metadata_routing=True`` (see :func:`sklearn.set_config`).
 |      Please see :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      The options for each parameter are:
 |
 |      - ``True``: metadata is requested, and passed to ``predict_proba`` i
f provided. The request is ignored if metadata is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will n
ot pass it to ``predict_proba``.
 |
 |      - ``None``: metadata is not requested, and the meta-estimator will r
aise an error if the user provides it.
 |
 |      - ``str``: metadata should be passed to the meta-estimator with this
given alias instead of the original name.
 |
 |      The default (``sklearn.utils.metadata_routing.UNCHANGED``) retains t
he
 |      existing request. This allows you to change the request for some
 |      parameters and not others.
 |
 |      .. versionadded:: 1.3
 |
 |      .. note::
 |          This method is only relevant if this estimator is used as a
 |          sub-estimator of a meta-estimator, e.g. used inside a
 |          :class:`~sklearn.pipeline.Pipeline`. Otherwise it has no effect.
 |
 |      Parameters
```

```
 |      ----------
 |      check_input : str, True, False, or None,                    default
=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``check_input`` parameter in ``predict_prob
a``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
 |
 |  set_predict_request(self: sklearn.tree._classes.DecisionTreeClassifier,
*, check_input: Union[bool, NoneType, str] = '$UNCHANGED$') -> sklearn.tree.
_classes.DecisionTreeClassifier
 |      Request metadata passed to the ``predict`` method.
 |
 |      Note that this method is only relevant if
 |      ``enable_metadata_routing=True`` (see :func:`sklearn.set_config`).
 |      Please see :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      The options for each parameter are:
 |
 |      - ``True``: metadata is requested, and passed to ``predict`` if prov
ided. The request is ignored if metadata is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will n
ot pass it to ``predict``.
 |
 |      - ``None``: metadata is not requested, and the meta-estimator will r
aise an error if the user provides it.
 |
 |      - ``str``: metadata should be passed to the meta-estimator with this
given alias instead of the original name.
 |
 |      The default (``sklearn.utils.metadata_routing.UNCHANGED``) retains t
he
 |      existing request. This allows you to change the request for some
 |      parameters and not others.
 |
 |      .. versionadded:: 1.3
 |
 |      .. note::
 |          This method is only relevant if this estimator is used as a
 |          sub-estimator of a meta-estimator, e.g. used inside a
 |          :class:`~sklearn.pipeline.Pipeline`. Otherwise it has no effect.
 |
 |      Parameters
 |      ----------
 |      check_input : str, True, False, or None,                    default
=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``check_input`` parameter in ``predict``.
 |
 |      Returns
 |      -------
 |      self : object
```

```
 |          The updated object.
 |
 |  set_score_request(self: sklearn.tree._classes.DecisionTreeClassifier, *,
sample_weight: Union[bool, NoneType, str] = '$UNCHANGED$') -> sklearn.tree._
classes.DecisionTreeClassifier
 |      Request metadata passed to the ``score`` method.
 |
 |      Note that this method is only relevant if
 |      ``enable_metadata_routing=True`` (see :func:`sklearn.set_config`).
 |      Please see :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      The options for each parameter are:
 |
 |      - ``True``: metadata is requested, and passed to ``score`` if provid
ed. The request is ignored if metadata is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will n
ot pass it to ``score``.
 |
 |      - ``None``: metadata is not requested, and the meta-estimator will r
aise an error if the user provides it.
 |
 |      - ``str``: metadata should be passed to the meta-estimator with this
given alias instead of the original name.
 |
 |      The default (``sklearn.utils.metadata_routing.UNCHANGED``) retains t
he
 |      existing request. This allows you to change the request for some
 |      parameters and not others.
 |
 |      .. versionadded:: 1.3
 |
 |      .. note::
 |          This method is only relevant if this estimator is used as a
 |          sub-estimator of a meta-estimator, e.g. used inside a
 |          :class:`~sklearn.pipeline.Pipeline`. Otherwise it has no effect.
 |
 |      Parameters
 |      ----------
 |      sample_weight : str, True, False, or None,                    defau
lt=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``sample_weight`` parameter in ``score``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  __annotations__ = {'_parameter_constraints': <class 'dict'>}
 |
```

```
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.ClassifierMixin:
|
|  score(self, X, y, sample_weight=None)
|      Return the mean accuracy on the given test data and labels.
|
|      In multi-label classification, this is the subset accuracy
|      which is a harsh metric since you require for each sample that
|      each label set be correctly predicted.
|
|      Parameters
|      ----------
|      X : array-like of shape (n_samples, n_features)
|          Test samples.
|
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          True labels for `X`.
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights.
|
|      Returns
|      -------
|      score : float
|          Mean accuracy of ``self.predict(X)`` w.r.t. `y`.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.ClassifierMixin:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from BaseDecisionTree:
|
|  apply(self, X, check_input=True)
|      Return the index of the leaf that each sample is predicted as.
|
|      .. versionadded:: 0.17
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      check_input : bool, default=True
|          Allow to bypass several input checking.
|          Don't use this parameter unless you know what you're doing.
|
|      Returns
|      -------
```

```
 |      X_leaves : array-like of shape (n_samples,)
 |          For each datapoint x in X, return the index of the leaf x
 |          ends up in. Leaves are numbered within
 |          ``[0; self.tree_.node_count)``, possibly with gaps in the
 |          numbering.
 |
 |  cost_complexity_pruning_path(self, X, y, sample_weight=None)
 |      Compute the pruning path during Minimal Cost-Complexity Pruning.
 |
 |      See :ref:`minimal_cost_complexity_pruning` for details on the prunin
g
 |      process.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The training input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csc_matrix``.
 |
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          The target values (class labels) as integers or strings.
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Sample weights. If None, then samples are equally weighted. Spli
ts
 |          that would create child nodes with net zero or negative weight a
re
 |          ignored while searching for a split in each node. Splits are als
o
 |          ignored if they would result in any single class carrying a
 |          negative weight in either child node.
 |
 |      Returns
 |      -------
 |      ccp_path : :class:`~sklearn.utils.Bunch`
 |          Dictionary-like object, with the following attributes.
 |
 |          ccp_alphas : ndarray
 |              Effective alphas of subtree during pruning.
 |
 |          impurities : ndarray
 |              Sum of the impurities of the subtree leaves for the
 |              corresponding alpha value in ``ccp_alphas``.
 |
 |  decision_path(self, X, check_input=True)
 |      Return the decision path in the tree.
 |
 |      .. versionadded:: 0.18
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csr_matrix``.
```

```
 |
 |      check_input : bool, default=True
 |          Allow to bypass several input checking.
 |          Don't use this parameter unless you know what you're doing.
 |
 |      Returns
 |      -------
 |      indicator : sparse matrix of shape (n_samples, n_nodes)
 |          Return a node indicator CSR matrix where non zero elements
 |          indicates that the samples goes through the nodes.
 |
 |  get_depth(self)
 |      Return the depth of the decision tree.
 |
 |      The depth of a tree is the maximum distance between the root
 |      and any leaf.
 |
 |      Returns
 |      -------
 |      self.tree_.max_depth : int
 |          The maximum depth of the tree.
 |
 |  get_n_leaves(self)
 |      Return the number of leaves of the decision tree.
 |
 |      Returns
 |      -------
 |      self.tree_.n_leaves : int
 |          Number of leaves.
 |
 |  predict(self, X, check_input=True)
 |      Predict class or regression value for X.
 |
 |      For a classification model, the predicted class for each sample in X
is
 |      returned. For a regression model, the predicted value based on X is
 |      returned.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csr_matrix``.
 |
 |      check_input : bool, default=True
 |          Allow to bypass several input checking.
 |          Don't use this parameter unless you know what you're doing.
 |
 |      Returns
 |      -------
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          The predicted classes, or the predict values.
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from BaseDecisionTree:
```

```
|
|  feature_importances_
|      Return the feature importances.
|
|      The importance of a feature is computed as the (normalized) total
|      reduction of the criterion brought by that feature.
|      It is also known as the Gini importance.
|
|      Warning: impurity-based feature importances can be misleading for
|      high cardinality features (many unique values). See
|      :func:`sklearn.inspection.permutation_importance` as an alternative.
|
|      Returns
|      -------
|      feature_importances_ : ndarray of shape (n_features,)
|          Normalized total reduction of criteria by feature
|          (Gini importance).
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|
|  __repr__(self, N_CHAR_MAX=700)
|      Return repr(self).
|
|  __setstate__(self, state)
|
|  __sklearn_clone__(self)
|
|  get_params(self, deep=True)
|      Get parameters for this estimator.
|
|      Parameters
|      ----------
|      deep : bool, default=True
|          If True, will return the parameters for this estimator and
|          contained subobjects that are estimators.
|
|      Returns
|      -------
|      params : dict
|          Parameter names mapped to their values.
|
|  set_params(self, **params)
|      Set the parameters of this estimator.
|
|      The method works on simple estimators as well as on nested objects
|      (such as :class:`~sklearn.pipeline.Pipeline`). The latter have
|      parameters of the form ``<component>__<parameter>`` so that it's
|      possible to update each component of a nested object.
|
|      Parameters
|      ----------
|      **params : dict
|          Estimator parameters.
```

```
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.utils._metadata_requests._MetadataRequest
er:
 |
 |  get_metadata_routing(self)
 |      Get metadata routing of this object.
 |
 |      Please check :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      Returns
 |      -------
 |      routing : MetadataRequest
 |          A :class:`~sklearn.utils.metadata_routing.MetadataRequest` encap
sulating
 |          routing information.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from sklearn.utils._metadata_requests._MetadataR
equester:
 |
 |  __init_subclass__(**kwargs) from abc.ABCMeta
 |      Set the ``set_{method}_request`` methods.
 |
 |      This uses PEP-487 [1]_ to set the ``set_{method}_request`` methods.
It
 |      looks for the information available in the set default values which
are
 |      set using ``__metadata_request__*`` class attributes, or inferred
 |      from method signatures.
 |
 |      The ``__metadata_request__*`` class attributes are used when a metho
d
 |      does not explicitly accept a metadata through its arguments or if th
e
 |      developer would like to specify a request value for those metadata
 |      which are different from the default ``None``.
 |
 |      References
 |      ----------
 |      .. [1] https://www.python.org/dev/peps/pep-0487
```

In [ ]:
```
for max_depth in range(1,20):
    dt = DecisionTreeClassifier(max_depth=max_depth).fit(X_train, y_train)
    print(f'\nMax Depth: {max_depth}')
    dt_train = accuracy_score(y_train, dt.predict(X_train))
    dt_test = accuracy_score(y_test, dt.predict(X_test))
    print(f'DT Accuracy (train/test): {dt_train:.3f}/{dt_test:.3f}')
```

```
Max Depth: 1
DT Accuracy (train/test): 0.669/0.556

Max Depth: 2
DT Accuracy (train/test): 0.937/0.889

Max Depth: 3
DT Accuracy (train/test): 0.993/0.944

Max Depth: 4
DT Accuracy (train/test): 1.000/0.944

Max Depth: 5
DT Accuracy (train/test): 1.000/0.944

Max Depth: 6
DT Accuracy (train/test): 1.000/0.944

Max Depth: 7
DT Accuracy (train/test): 1.000/0.944

Max Depth: 8
DT Accuracy (train/test): 1.000/0.944

Max Depth: 9
DT Accuracy (train/test): 1.000/0.944

Max Depth: 10
DT Accuracy (train/test): 1.000/0.944

Max Depth: 11
DT Accuracy (train/test): 1.000/0.944

Max Depth: 12
DT Accuracy (train/test): 1.000/0.944

Max Depth: 13
DT Accuracy (train/test): 1.000/0.944

Max Depth: 14
DT Accuracy (train/test): 1.000/0.944

Max Depth: 15
DT Accuracy (train/test): 1.000/0.944

Max Depth: 16
DT Accuracy (train/test): 1.000/0.944

Max Depth: 17
DT Accuracy (train/test): 1.000/0.944

Max Depth: 18
DT Accuracy (train/test): 1.000/0.944

Max Depth: 19
DT Accuracy (train/test): 1.000/0.944
```

In [ ]:
```
help(RandomForestClassifier)
```

```
Help on class RandomForestClassifier in module sklearn.ensemble._forest:

class RandomForestClassifier(ForestClassifier)
 |  RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=
None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstr
ap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_st
art=False, class_weight=None, ccp_alpha=0.0, max_samples=None, monotonic_cst
=None)
 |
 |  A random forest classifier.
 |
 |  A random forest is a meta estimator that fits a number of decision tree
 |  classifiers on various sub-samples of the dataset and uses averaging to
 |  improve the predictive accuracy and control over-fitting.
 |  Trees in the forest use the best split strategy, i.e. equivalent to pass
ing
 |  `splitter="best"` to the underlying :class:`~sklearn.tree.DecisionTreeRe
gressor`.
 |  The sub-sample size is controlled with the `max_samples` parameter if
 |  `bootstrap=True` (default), otherwise the whole dataset is used to build
 |  each tree.
 |
 |  For a comparison between tree-based ensemble models see the example
 |  :ref:`sphx_glr_auto_examples_ensemble_plot_forest_hist_grad_boosting_com
parison.py`.
 |
 |  Read more in the :ref:`User Guide <forest>`.
 |
 |  Parameters
 |  ----------
 |  n_estimators : int, default=100
 |      The number of trees in the forest.
 |
 |      .. versionchanged:: 0.22
 |         The default value of ``n_estimators`` changed from 10 to 100
 |         in 0.22.
 |
 |  criterion : {"gini", "entropy", "log_loss"}, default="gini"
 |      The function to measure the quality of a split. Supported criteria a
re
 |      "gini" for the Gini impurity and "log_loss" and "entropy" both for t
he
 |      Shannon information gain, see :ref:`tree_mathematical_formulation`.
 |      Note: This parameter is tree-specific.
 |
 |  max_depth : int, default=None
 |      The maximum depth of the tree. If None, then nodes are expanded unti
l
 |      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.
 |
 |      .. versionchanged:: 0.18
 |         Added float values for fractions.
 |
 |  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.
 |
 |      .. versionchanged:: 0.18
 |         Added float values for fractions.
 |
 |  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 : {"sqrt", "log2", None}, int or float, default="sqrt"
 |      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
 |        `max(1, int(max_features * n_features_in_))` features are consider
ed at each
 |        split.
 |      - If "sqrt", then `max_features=sqrt(n_features)`.
 |      - If "log2", then `max_features=log2(n_features)`.
 |      - If None, then `max_features=n_features`.
 |
 |      .. versionchanged:: 1.1
 |          The default of `max_features` changed from `"auto"` to `"sqrt"`.
 |
 |      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.
 |
 |  max_leaf_nodes : int, default=None
 |      Grow trees 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 impurit
y
 |      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 t
he
  |      left child, and ``N_t_R`` is the number of samples in the right chil
d.
  |
  |      ``N``, ``N_t``, ``N_t_R`` and ``N_t_L`` all refer to the weighted su
m,
  |      if ``sample_weight`` is passed.
  |
  |      .. versionadded:: 0.19
  |
  |  bootstrap : bool, default=True
  |      Whether bootstrap samples are used when building trees. If False, th
e
  |      whole dataset is used to build each tree.
  |
  |  oob_score : bool or callable, default=False
  |      Whether to use out-of-bag samples to estimate the generalization sco
re.
  |      By default, :func:`~sklearn.metrics.accuracy_score` is used.
  |      Provide a callable with signature `metric(y_true, y_pred)` to use a
  |      custom metric. Only available if `bootstrap=True`.
  |
  |  n_jobs : int, default=None
  |      The number of jobs to run in parallel. :meth:`fit`, :meth:`predict`,
  |      :meth:`decision_path` and :meth:`apply` are all parallelized over th
e
  |      trees. ``None`` means 1 unless in a :obj:`joblib.parallel_backend`
  |      context. ``-1`` means using all processors. See :term:`Glossary
  |      <n_jobs>` for more details.
  |
  |  random_state : int, RandomState instance or None, default=None
  |      Controls both the randomness of the bootstrapping of the samples use
d
  |      when building trees (if ``bootstrap=True``) and the sampling of the
  |      features to consider when looking for the best split at each node
  |      (if ``max_features < n_features``).
  |      See :term:`Glossary <random_state>` for details.
  |
  |  verbose : int, default=0
  |      Controls the verbosity when fitting and predicting.
  |
  |  warm_start : bool, default=False
  |      When set to ``True``, reuse the solution of the previous call to fit
  |      and add more estimators to the ensemble, otherwise, just fit a whole
  |      new forest. See :term:`Glossary <warm_start>` and
  |      :ref:`gradient_boosting_warm_start` for details.
  |
  |  class_weight : {"balanced", "balanced_subsample"}, dict or list of dict
s,             default=None
```

```
 |      Weights associated with classes in the form ``{class_label: weight}`
`.
 |      If not given, 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 dat
a
 |      as ``n_samples / (n_classes * np.bincount(y))``
 |
 |      The "balanced_subsample" mode is the same as "balanced" except that
 |      weights are computed based on the bootstrap sample for every tree
 |      grown.
 |
 |      For multi-output, the weights of each column of y will be multiplie
d.
 |
 |      Note that these weights will be multiplied with sample_weight (passe
d
 |      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. S
ee
 |      :ref:`minimal_cost_complexity_pruning` for details.
 |
 |      .. versionadded:: 0.22
 |
 |  max_samples : int or float, default=None
 |      If bootstrap is True, the number of samples to draw from X
 |      to train each base estimator.
 |
 |      - If None (default), then draw `X.shape[0]` samples.
 |      - If int, then draw `max_samples` samples.
 |      - If float, then draw `max(round(n_samples * max_samples), 1)` sampl
es. Thus,
 |        `max_samples` should be in the interval `(0.0, 1.0]`.
 |
 |      .. versionadded:: 0.22
 |
 |  monotonic_cst : array-like of int of shape (n_features), default=None
 |      Indicates the monotonicity constraint to enforce on each feature.
 |        - 1: monotonic increase
 |        - 0: no constraint
 |        - -1: monotonic decrease
 |
 |      If monotonic_cst is None, no constraints are applied.
```

```
 |
 |      Monotonicity constraints are not supported for:
 |        - multiclass classifications (i.e. when `n_classes > 2`),
 |        - multioutput classifications (i.e. when `n_outputs_ > 1`),
 |        - classifications trained on data with missing values.
 |
 |      The constraints hold over the probability of the positive class.
 |
 |      Read more in the :ref:`User Guide <monotonic_cst_gbdt>`.
 |
 |      .. versionadded:: 1.4
 |
 |  Attributes
 |  ----------
 |  estimator_ : :class:`~sklearn.tree.DecisionTreeClassifier`
 |      The child estimator template used to create the collection of fitted
 |      sub-estimators.
 |
 |      .. versionadded:: 1.2
 |         `base_estimator_` was renamed to `estimator_`.
 |
 |  estimators_ : list of DecisionTreeClassifier
 |      The collection of fitted sub-estimators.
 |
 |  classes_ : ndarray of shape (n_classes,) or a list of such arrays
 |      The classes labels (single output problem), or a list of arrays of
 |      class labels (multi-output problem).
 |
 |  n_classes_ : int or list
 |      The number of classes (single output problem), or a list containing
the
 |      number of classes for each output (multi-output problem).
 |
 |  n_features_in_ : int
 |      Number of features seen during :term:`fit`.
 |
 |      .. versionadded:: 0.24
 |
 |  feature_names_in_ : ndarray of shape (`n_features_in_`,)
 |      Names of features seen during :term:`fit`. Defined only when `X`
 |      has feature names that are all strings.
 |
 |      .. versionadded:: 1.0
 |
 |  n_outputs_ : int
 |      The number of outputs when ``fit`` is performed.
 |
 |  feature_importances_ : ndarray of shape (n_features,)
 |      The impurity-based feature importances.
 |      The higher, the more important the feature.
 |      The importance of a feature is computed as the (normalized)
 |      total reduction of the criterion brought by that feature.  It is als
o
 |      known as the Gini importance.
 |
 |      Warning: impurity-based feature importances can be misleading for
```

```
 |      high cardinality features (many unique values). See
 |      :func:`sklearn.inspection.permutation_importance` as an alternative.
 |
 |  oob_score_ : float
 |      Score of the training dataset obtained using an out-of-bag estimate.
 |      This attribute exists only when ``oob_score`` is True.
 |
 |  oob_decision_function_ : ndarray of shape (n_samples, n_classes) or
(n_samples, n_classes, n_outputs)
 |      Decision function computed with out-of-bag estimate on the training
 |      set. If n_estimators is small it might be possible that a data point
 |      was never left out during the bootstrap. In this case,
 |      `oob_decision_function_` might contain NaN. This attribute exists
 |      only when ``oob_score`` is True.
 |
 |  estimators_samples_ : list of arrays
 |      The subset of drawn samples (i.e., the in-bag samples) for each base
 |      estimator. Each subset is defined by an array of the indices selecte
d.
 |
 |      .. versionadded:: 1.4
 |
 |  See Also
 |  --------
 |  sklearn.tree.DecisionTreeClassifier : A decision tree classifier.
 |  sklearn.ensemble.ExtraTreesClassifier : Ensemble of extremely randomized
 |      tree classifiers.
 |  sklearn.ensemble.HistGradientBoostingClassifier : A Histogram-based Grad
ient
 |      Boosting Classification Tree, very fast for big datasets (n_samples
>=
 |      10_000).
 |
 |  Notes
 |  -----
 |  The default values for the parameters controlling the size of the trees
 |  (e.g. ``max_depth``, ``min_samples_leaf``, etc.) lead to fully grown and
 |  unpruned trees which can potentially be very large on some data sets. To
 |  reduce memory consumption, the complexity and size of the trees should b
e
 |  controlled by setting those parameter values.
 |
 |  The features are always randomly permuted at each split. Therefore,
 |  the best found split may vary, even with the same training data,
 |  ``max_features=n_features`` and ``bootstrap=False``, if the improvement
 |  of the criterion is identical for several splits enumerated during the
 |  search of the best split. To obtain a deterministic behaviour during
 |  fitting, ``random_state`` has to be fixed.
 |
 |  References
 |  ----------
 |  .. [1] L. Breiman, "Random Forests", Machine Learning, 45(1), 5-32, 200
1.
 |
 |  Examples
 |  --------
```

```
 |  >>> from sklearn.ensemble import RandomForestClassifier
 |  >>> from sklearn.datasets import make_classification
 |  >>> X, y = make_classification(n_samples=1000, n_features=4,
 |  ...                            n_informative=2, n_redundant=0,
 |  ...                            random_state=0, shuffle=False)
 |  >>> clf = RandomForestClassifier(max_depth=2, random_state=0)
 |  >>> clf.fit(X, y)
 |  RandomForestClassifier(...)
 |  >>> print(clf.predict([[0, 0, 0, 0]]))
 |  [1]
 |
 |  Method resolution order:
 |      RandomForestClassifier
 |      ForestClassifier
 |      sklearn.base.ClassifierMixin
 |      BaseForest
 |      sklearn.base.MultiOutputMixin
 |      sklearn.ensemble._base.BaseEnsemble
 |      sklearn.base.MetaEstimatorMixin
 |      sklearn.base.BaseEstimator
 |      sklearn.utils._estimator_html_repr._HTMLDocumentationLinkMixin
 |      sklearn.utils._metadata_requests._MetadataRequester
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, n_estimators=100, *, criterion='gini', max_depth=None, mi
n_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_fea
tures='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=Tru
e, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=Fa
lse, class_weight=None, ccp_alpha=0.0, max_samples=None, monotonic_cst=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  set_fit_request(self: sklearn.ensemble._forest.RandomForestClassifier,
*, sample_weight: Union[bool, NoneType, str] = '$UNCHANGED$') -> sklearn.ens
emble._forest.RandomForestClassifier
 |      Request metadata passed to the ``fit`` method.
 |
 |      Note that this method is only relevant if
 |      ``enable_metadata_routing=True`` (see :func:`sklearn.set_config`).
 |      Please see :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      The options for each parameter are:
 |
 |      - ``True``: metadata is requested, and passed to ``fit`` if provide
d. The request is ignored if metadata is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will n
ot pass it to ``fit``.
 |
 |      - ``None``: metadata is not requested, and the meta-estimator will r
aise an error if the user provides it.
 |
 |      - ``str``: metadata should be passed to the meta-estimator with this
given alias instead of the original name.
```

```
 |
 |      The default (``sklearn.utils.metadata_routing.UNCHANGED``) retains t
he
 |      existing request. This allows you to change the request for some
 |      parameters and not others.
 |
 |      .. versionadded:: 1.3
 |
 |      .. note::
 |          This method is only relevant if this estimator is used as a
 |          sub-estimator of a meta-estimator, e.g. used inside a
 |          :class:`~sklearn.pipeline.Pipeline`. Otherwise it has no effect.
 |
 |      Parameters
 |      ----------
 |      sample_weight : str, True, False, or None,                     defau
lt=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``sample_weight`` parameter in ``fit``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
 |
 |  set_score_request(self: sklearn.ensemble._forest.RandomForestClassifier,
*, sample_weight: Union[bool, NoneType, str] = '$UNCHANGED$') -> sklearn.ens
emble._forest.RandomForestClassifier
 |      Request metadata passed to the ``score`` method.
 |
 |      Note that this method is only relevant if
 |      ``enable_metadata_routing=True`` (see :func:`sklearn.set_config`).
 |      Please see :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      The options for each parameter are:
 |
 |      - ``True``: metadata is requested, and passed to ``score`` if provid
ed. The request is ignored if metadata is not provided.
 |
 |      - ``False``: metadata is not requested and the meta-estimator will n
ot pass it to ``score``.
 |
 |      - ``None``: metadata is not requested, and the meta-estimator will r
aise an error if the user provides it.
 |
 |      - ``str``: metadata should be passed to the meta-estimator with this
given alias instead of the original name.
 |
 |      The default (``sklearn.utils.metadata_routing.UNCHANGED``) retains t
he
 |      existing request. This allows you to change the request for some
 |      parameters and not others.
 |
 |      .. versionadded:: 1.3
 |
 |      .. note::
```

```
 |          This method is only relevant if this estimator is used as a
 |          sub-estimator of a meta-estimator, e.g. used inside a
 |          :class:`~sklearn.pipeline.Pipeline`. Otherwise it has no effect.
 |
 |      Parameters
 |      ----------
 |      sample_weight : str, True, False, or None,                     defau
lt=sklearn.utils.metadata_routing.UNCHANGED
 |          Metadata routing for ``sample_weight`` parameter in ``score``.
 |
 |      Returns
 |      -------
 |      self : object
 |          The updated object.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  __annotations__ = {'_parameter_constraints': <class 'dict'>}
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from ForestClassifier:
 |
 |  predict(self, X)
 |      Predict class for X.
 |
 |      The predicted class of an input sample is a vote by the trees in
 |      the forest, weighted by their probability estimates. That is,
 |      the predicted class is the one with highest mean probability
 |      estimate across the trees.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, its dtype will be converted to
 |          ``dtype=np.float32``. If a sparse matrix is provided, it will be
 |          converted into a sparse ``csr_matrix``.
 |
 |      Returns
 |      -------
 |      y : ndarray of shape (n_samples,) or (n_samples, n_outputs)
 |          The predicted classes.
 |
 |  predict_log_proba(self, X)
 |      Predict class log-probabilities for X.
 |
 |      The predicted class log-probabilities of an input sample is computed
as
 |      the log of the mean predicted class probabilities of the trees in th
e
 |      forest.
 |
 |      Parameters
 |      ----------
```

```
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, its dtype will be converted to
 |          ``dtype=np.float32``. If a sparse matrix is provided, it will be
 |          converted into a sparse ``csr_matrix``.
 |
 |      Returns
 |      -------
 |      p : ndarray of shape (n_samples, n_classes), or a list of such array
s
 |          The class probabilities of the input samples. The order of the
 |          classes corresponds to that in the attribute :term:`classes_`.
 |
 |  predict_proba(self, X)
 |      Predict class probabilities for X.
 |
 |      The predicted class probabilities of an input sample are computed as
 |      the mean predicted class probabilities of the trees in the forest.
 |      The class probability of a single tree is the fraction of samples of
 |      the same class in a leaf.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, its dtype will be converted to
 |          ``dtype=np.float32``. If a sparse matrix is provided, it will be
 |          converted into a sparse ``csr_matrix``.
 |
 |      Returns
 |      -------
 |      p : ndarray of shape (n_samples, n_classes), or a list of such array
s
 |          The class probabilities of the input samples. The order of the
 |          classes corresponds to that in the attribute :term:`classes_`.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.ClassifierMixin:
 |
 |  score(self, X, y, sample_weight=None)
 |      Return the mean accuracy on the given test data and labels.
 |
 |      In multi-label classification, this is the subset accuracy
 |      which is a harsh metric since you require for each sample that
 |      each label set be correctly predicted.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Test samples.
 |
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          True labels for `X`.
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Sample weights.
 |
 |      Returns
```

```
|      -------
|      score : float
|          Mean accuracy of ``self.predict(X)`` w.r.t. `y`.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.ClassifierMixin:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from BaseForest:
|
|  apply(self, X)
|      Apply trees in the forest to X, return leaf indices.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, its dtype will be converted to
|          ``dtype=np.float32``. If a sparse matrix is provided, it will be
|          converted into a sparse ``csr_matrix``.
|
|      Returns
|      -------
|      X_leaves : ndarray of shape (n_samples, n_estimators)
|          For each datapoint x in X and for each tree in the forest,
|          return the index of the leaf x ends up in.
|
|  decision_path(self, X)
|      Return the decision path in the forest.
|
|      .. versionadded:: 0.18
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, its dtype will be converted to
|          ``dtype=np.float32``. If a sparse matrix is provided, it will be
|          converted into a sparse ``csr_matrix``.
|
|      Returns
|      -------
|      indicator : sparse matrix of shape (n_samples, n_nodes)
|          Return a node indicator matrix where non zero elements indicates
|          that the samples goes through the nodes. The matrix is of CSR
|          format.
|
|      n_nodes_ptr : ndarray of shape (n_estimators + 1,)
|          The columns from indicator[n_nodes_ptr[i]:n_nodes_ptr[i+1]]
|          gives the indicator value for the i-th estimator.
|
|  fit(self, X, y, sample_weight=None)
```

```
 |      Build a forest of trees from the training set (X, y).
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The training input samples. Internally, its dtype will be conver
ted
 |          to ``dtype=np.float32``. If a sparse matrix is provided, it will
be
 |          converted into a sparse ``csc_matrix``.
 |
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          The target values (class labels in classification, real numbers
in
 |          regression).
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Sample weights. If None, then samples are equally weighted. Spli
ts
 |          that would create child nodes with net zero or negative weight a
re
 |          ignored while searching for a split in each node. In the case of
 |          classification, splits are also ignored if they would result in
any
 |          single class carrying a negative weight in either child node.
 |
 |      Returns
 |      -------
 |      self : object
 |          Fitted estimator.
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from BaseForest:
 |
 |  estimators_samples_
 |      The subset of drawn samples for each base estimator.
 |
 |      Returns a dynamically generated list of indices identifying
 |      the samples used for fitting each member of the ensemble, i.e.,
 |      the in-bag samples.
 |
 |      Note: the list is re-created at each call to the property in order
 |      to reduce the object memory footprint by not storing the sampling
 |      data. Thus fetching the property may be slower than expected.
 |
 |  feature_importances_
 |      The impurity-based feature importances.
 |
 |      The higher, the more important the feature.
 |      The importance of a feature is computed as the (normalized)
 |      total reduction of the criterion brought by that feature.  It is als
o
 |      known as the Gini importance.
 |
 |      Warning: impurity-based feature importances can be misleading for
 |      high cardinality features (many unique values). See
```

```
 |      :func:`sklearn.inspection.permutation_importance` as an alternative.
 |
 |      Returns
 |      -------
 |      feature_importances_ : ndarray of shape (n_features,)
 |          The values of this array sum to 1, unless all trees are single n
ode
 |          trees consisting of only the root node, in which case it will be
an
 |          array of zeros.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.ensemble._base.BaseEnsemble:
 |
 |  __getitem__(self, index)
 |      Return the index'th estimator in the ensemble.
 |
 |  __iter__(self)
 |      Return iterator over estimators in the ensemble.
 |
 |  __len__(self)
 |      Return the number of estimators in the ensemble.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  __sklearn_clone__(self)
 |
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |
 |      Parameters
 |      ----------
 |      deep : bool, default=True
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |
 |      Returns
 |      -------
 |      params : dict
 |          Parameter names mapped to their values.
 |
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |
 |      The method works on simple estimators as well as on nested objects
 |      (such as :class:`~sklearn.pipeline.Pipeline`). The latter have
 |      parameters of the form ``<component>__<parameter>`` so that it's
 |      possible to update each component of a nested object.
```

```
 |
 |      Parameters
 |      ----------
 |      **params : dict
 |          Estimator parameters.
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.utils._metadata_requests._MetadataRequest
er:
 |
 |  get_metadata_routing(self)
 |      Get metadata routing of this object.
 |
 |      Please check :ref:`User Guide <metadata_routing>` on how the routing
 |      mechanism works.
 |
 |      Returns
 |      -------
 |      routing : MetadataRequest
 |          A :class:`~sklearn.utils.metadata_routing.MetadataRequest` encap
sulating
 |          routing information.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from sklearn.utils._metadata_requests._MetadataR
equester:
 |
 |  __init_subclass__(**kwargs) from abc.ABCMeta
 |      Set the ``set_{method}_request`` methods.
 |
 |      This uses PEP-487 [1]_ to set the ``set_{method}_request`` methods.
It
 |      looks for the information available in the set default values which
are
 |      set using ``__metadata_request__*`` class attributes, or inferred
 |      from method signatures.
 |
 |      The ``__metadata_request__*`` class attributes are used when a metho
d
 |      does not explicitly accept a metadata through its arguments or if th
e
 |      developer would like to specify a request value for those metadata
 |      which are different from the default ``None``.
 |
 |      References
 |      ----------
 |      .. [1] https://www.python.org/dev/peps/pep-0487
```

In [ ]:
```
for num in [1,2,5,10,100]:
    rf = RandomForestClassifier(n_estimators=num).fit(X_train, y_train)
```

```
    print(f'\nNumber of estimators: {num}')
    rf_train = accuracy_score(y_train, rf.predict(X_train))
    rf_test = accuracy_score(y_test, rf.predict(X_test))
    print(f'RF Accuracy (train/test): {rf_train:.3f}/{rf_test:.3f}')
```

```
Number of estimators: 1
RF Accuracy (train/test): 0.993/0.972

Number of estimators: 2
RF Accuracy (train/test): 0.972/0.944

Number of estimators: 5
RF Accuracy (train/test): 0.993/0.944

Number of estimators: 10
RF Accuracy (train/test): 1.000/0.972

Number of estimators: 100
RF Accuracy (train/test): 1.000/0.972
```

ANI project: RMSE vs MAE

- You should aim for an RMSE < 3 kcal/mol, or MAE < 2 kcal/mol on the test set.
- Compared to MAE, RMSE is usually numerically larger and more sensitive to outliers.
- Be aware of the energy unit conversions!

$$1\ \text{Hartree} = 627.5094738898777\ \text{kcal/mol}$$

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_i (\hat{y}_i - y_i)^2}$$

$$\text{MAE} = \frac{1}{N}\sum_i |\hat{y}_i - y_i|$$

In [ ]:
```
def rmse(y_pred, y_true):
    return np.sqrt(np.mean((y_pred - y_true) ** 2))
def mae(y_pred, y_true):
    return np.mean(np.abs(y_pred - y_true))
```

In [ ]:
```
y_true = np.random.random(1000) * 20 - 10
y_pred = y_true + np.random.randn(1000) * 2

print('RMSE:',rmse(y_true, y_pred))
print('MAE:',mae(y_true, y_pred))
```

```
RMSE: 1.995830654993427
MAE: 1.5699117910938598
```

In [ ]:
```

```