Mudcard

- **why is only the second column of predict_proba being used to find the best critical probability?**
  - You can use either column but be careful of how you convert the critical probability to a predicted class
  - If you use the first column, that's class 0 predicted probability. If it is larger than 50%, you need to predict class 0.
  - If you use the second column, that's class 1 predicted probability. If it is larger than 50%, you need to predcit class 1.
- **For decision trees and random forest, will we need to do the steps that we did during the quiz? Or is that what python will do for us?**
  - slearn optimizes the decision tree for you and it also aggregates the votes of trees if you use a random forest

Supervised ML algorithms

By the end of this week, you will be able to

- Summarize how decision trees, random forests, and support vector machines work
- Describe how the predictions of these techniques behave in classification and regression
- Describe which hyper-parameters should be tuned

A decision tree in regression

In [1]:
```
import numpy as np
from sklearn.ensemble import RandomForestRegressor
np.random.seed(10)
def true_fun(X):
    return np.cos(1.5 * np.pi * X)

n_samples = 30

X = np.random.rand(n_samples)
y = true_fun(X) + np.random.randn(n_samples) * 0.1

X_new = np.linspace(0, 1, 1000)

reg = RandomForestRegressor(n_estimators=1,max_depth=1)
reg.fit(X[:, np.newaxis],y)
y_new = reg.predict(X_new[:, np.newaxis])
```

In [2]:
```
help(RandomForestRegressor)
```

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

class RandomForestRegressor(ForestRegressor)
 |  RandomForestRegressor(n_estimators=100, *, criterion='squared_error', ma
x_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_l
eaf=0.0, max_features=1.0, max_leaf_nodes=None, min_impurity_decrease=0.0, b
ootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, w
arm_start=False, ccp_alpha=0.0, max_samples=None)
 |
 |  A random forest regressor.
 |
 |  A random forest is a meta estimator that fits a number of classifying
 |  decision trees on various sub-samples of the dataset and uses averaging
 |  to improve the predictive accuracy and control over-fitting.
 |  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 : {"squared_error", "absolute_error", "friedman_mse", "poisso
n"},             default="squared_error"
 |      The function to measure the quality of a split. Supported criteria
 |      are "squared_error" for the mean squared error, which is equal to
 |      variance reduction as feature selection criterion and minimizes the
L2
 |      loss using the mean of each terminal node, "friedman_mse", which use
s
 |      mean squared error with Friedman's improvement score for potential
 |      splits, "absolute_error" for the mean absolute error, which minimize
s
 |      the L1 loss using the median of each terminal node, and "poisson" wh
ich
 |      uses reduction in Poisson deviance to find splits.
 |      Training using "absolute_error" is significantly slower
 |      than when using "squared_error".
 |
 |      .. versionadded:: 0.18
 |         Mean Absolute Error (MAE) criterion.
 |
 |      .. versionadded:: 1.0
 |         Poisson criterion.
 |
```

```
 |  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=1.0
 |      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 or 1.0, then `max_features=n_features`.
 |
 |      .. note::
 |          The default of 1.0 is equivalent to bagged trees and more
 |          randomness can be achieved by setting smaller values, e.g. 0.3.
 |
 |      .. versionchanged:: 1.1
 |          The default of `max_features` changed from `"auto"` to 1.0.
 |
```

```
  |      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.r2_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.
 |
 |  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
 |
 |  Attributes
 |  ----------
 |  estimator_ : :class:`~sklearn.tree.DecisionTreeRegressor`
 |      The child estimator template used to create the collection of fitted
 |      sub-estimators.
 |
 |      .. versionadded:: 1.2
 |         `base_estimator_` was renamed to `estimator_`.
 |
 |  base_estimator_ : DecisionTreeRegressor
 |      The child estimator template used to create the collection of fitted
 |      sub-estimators.
 |
 |      .. deprecated:: 1.2
 |          `base_estimator_` is deprecated and will be removed in 1.4.
 |          Use `estimator_` instead.
 |
 |  estimators_ : list of DecisionTreeRegressor
 |      The collection of fitted sub-estimators.
 |
 |  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.
 |
 |  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.
 |
 |  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_prediction_ : ndarray of shape (n_samples,) or (n_samples, n_output
s)
 |      Prediction computed with out-of-bag estimate on the training set.
 |      This attribute exists only when ``oob_score`` is True.
 |
 |  See Also
 |  --------
 |  sklearn.tree.DecisionTreeRegressor : A decision tree regressor.
 |  sklearn.ensemble.ExtraTreesRegressor : Ensemble of extremely randomized
 |      tree regressors.
 |  sklearn.ensemble.HistGradientBoostingRegressor : A Histogram-based Gradi
ent
 |      Boosting Regression 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.
 |
 |  The default value ``max_features=1.0`` uses ``n_features``
 |  rather than ``n_features / 3``. The latter was originally suggested in
 |  [1], whereas the former was more recently justified empirically in [2].
 |
 |  References
 |  ----------
 |  .. [1] L. Breiman, "Random Forests", Machine Learning, 45(1), 5-32, 200
1.
 |
 |  .. [2] P. Geurts, D. Ernst., and L. Wehenkel, "Extremely randomized
 |         trees", Machine Learning, 63(1), 3-42, 2006.
 |
 |  Examples
 |  --------
 |  >>> from sklearn.ensemble import RandomForestRegressor
 |  >>> from sklearn.datasets import make_regression
 |  >>> X, y = make_regression(n_features=4, n_informative=2,
 |  ...                        random_state=0, shuffle=False)
 |  >>> regr = RandomForestRegressor(max_depth=2, random_state=0)
 |  >>> regr.fit(X, y)
 |  RandomForestRegressor(...)
 |  >>> print(regr.predict([[0, 0, 0, 0]]))
 |  [-8.32987858]
 |
 |  Method resolution order:
 |      RandomForestRegressor
 |      ForestRegressor
 |      sklearn.base.RegressorMixin
 |      BaseForest
 |      sklearn.base.MultiOutputMixin
 |      sklearn.ensemble._base.BaseEnsemble
 |      sklearn.base.MetaEstimatorMixin
 |      sklearn.base.BaseEstimator
 |      sklearn.utils._metadata_requests._MetadataRequester
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, n_estimators=100, *, criterion='squared_error', max_depth
=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.
0, max_features=1.0, 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, ccp_alpha=0.0, max_samples=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  set_fit_request(self: sklearn.ensemble._forest.RandomForestRegressor, *,
sample_weight: Union[bool, NoneType, str] = '$UNCHANGED$') -> sklearn.ensemb
le._forest.RandomForestRegressor
 |      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:`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.RandomForestRegressor,
*, sample_weight: Union[bool, NoneType, str] = '$UNCHANGED$') -> sklearn.ens
emble._forest.RandomForestRegressor
 |      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:`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 ForestRegressor:
 |
 |  predict(self, X)
 |      Predict regression target for X.
 |
 |      The predicted regression target of an input sample is computed as th
e
 |      mean predicted regression targets of the trees in the 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
 |      -------
 |      y : ndarray of shape (n_samples,) or (n_samples, n_outputs)
```

```
 |          The predicted values.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.RegressorMixin:
 |
 |  score(self, X, y, sample_weight=None)
 |      Return the coefficient of determination of the prediction.
 |
 |      The coefficient of determination :math:`R^2` is defined as
 |      :math:`(1 - \frac{u}{v})`, where :math:`u` is the residual
 |      sum of squares ``((y_true - y_pred)** 2).sum()`` and :math:`v`
 |      is the total sum of squares ``((y_true - y_true.mean()) ** 2).sum()`
`.
 |      The best possible score is 1.0 and it can be negative (because the
 |      model can be arbitrarily worse). A constant model that always predic
ts
 |      the expected value of `y`, disregarding the input features, would ge
t
 |      a :math:`R^2` score of 0.0.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Test samples. For some estimators this may be a precomputed
 |          kernel matrix or a list of generic objects instead with shape
 |          ``(n_samples, n_samples_fitted)``, where ``n_samples_fitted``
 |          is the number of samples used in the fitting for the estimator.
 |
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          True values for `X`.
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Sample weights.
 |
 |      Returns
 |      -------
 |      score : float
 |          :math:`R^2` of ``self.predict(X)`` w.r.t. `y`.
 |
 |      Notes
 |      -----
 |      The :math:`R^2` score used when calling ``score`` on a regressor use
s
 |      ``multioutput='uniform_average'`` from version 0.23 to keep consiste
nt
 |      with default value of :func:`~sklearn.metrics.r2_score`.
 |      This influences the ``score`` method of all the multioutput
 |      regressors (except for
 |      :class:`~sklearn.multioutput.MultiOutputRegressor`).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.RegressorMixin:
 |
 |  __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:
 |
 |  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.
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from sklearn.ensemble._base.BaseEnsemble:
 |
 |  base_estimator_
 |      Estimator used to grow the ensemble.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |      Helper for pickle.
 |
 |  __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:`~utils.metadata_routing.MetadataRequest` encapsulating
 |          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 [3]:

```
############################################################
# HUGE thanks to Drew Solomon and Yifei Song (DSI alumni)
# for preparing the visualizations in this lecture!
############################################################
# check out helper_functions.ipynb for more details
%run ./helper_functions.ipynb

hyperparameters = {
    'n_estimators': [1, 3, 10, 30],
    'max_depth': [1, 2, 3, 10, 30]
}

vis(X, y, RandomForestRegressor, hyperparameters, X_new)
```

```
interactive(children=(SelectionSlider(description='n_estimators', options=
(1, 3, 10, 30), value=1), SelectionS…
```

How to avoid overfitting with random forests?

- tune some (or all) of following hyperparameters:
  - max_depth
  - max_features
- With sklearn random forests, **do not tune n_estimators**!
  - the larger this value is, the better the forest will be
  - set n_estimators to maybe 100 while tuning hyperparameters
  - increase it if necessary once the best hyperparameters are found

ML algo	suitable for large datasets?	behaviour wrt outliers	non-linear?	params to tune	smooth predictions	easy to interpret?
linear regression	yes	linear extrapolation	no	l1 and/or l2 reg	yes	yes
logistic regression	yes	scales with distance from the decision boundary	no	l1 and/or l2 reg	yes	yes
random forest regression	so so	constant	yes	max_features, max_depth	no	so so
random forest classification	tbd	tbd	tbd	tbd	tbd	tbd
SVM rbf regression	tbd	tbd	tbd	tbd	tbd	tbd
SVM rbf classification	tbd	tbd	tbd	tbd	tbd	tbd

A random forest in classification

In [4]:
```
from sklearn.datasets import make_moons
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import ParameterGrid

# create the data
X,y = make_moons(noise=0.2, random_state=1,n_samples=200)
# set the hyperparameters
clf = RandomForestClassifier(n_estimators=1,max_depth=3,random_state=0)
# fit the model
clf.fit(X,y)
# predict new data
#y_new = clf.predict(X_new)
```

```
# predict probabilities
#y_new = clf.predict_proba(X_new)
```

Out[4]:

```
▾                    RandomForestClassifier
RandomForestClassifier(max_depth=3, n_estimators=1, random_state=0)
```

In [5]:

```
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)
 |
 |  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.
 |  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
  |
  |  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_`.
  |
  |  base_estimator_ : DecisionTreeClassifier
  |      The child estimator template used to create the collection of fitted
  |      sub-estimators.
```

```
 |
 |      .. deprecated:: 1.2
 |          `base_estimator_` is deprecated and will be removed in 1.4.
 |          Use `estimator_` instead.
 |
 |  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.
 |
 |  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._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)
 |      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:`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:`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:
 |
 |  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.
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from sklearn.ensemble._base.BaseEnsemble:
 |
 |  base_estimator_
 |      Estimator used to grow the ensemble.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |      Helper for pickle.
 |
```

```
 |  __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:`~utils.metadata_routing.MetadataRequest` encapsulating
 |          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 [6]:
```
# initialize RandomForestClassifier
ML_algo = RandomForestClassifier(random_state=42)

# set RF parameter grid
hyperparameters = {
    'n_estimators': [1, 3, 10, 30],
    'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
}

plot_clf_contour(hyperparameters, X, y)
```

```
interactive(children=(SelectionSlider(description='n_estimators', options=
(1, 3, 10, 30), value=1), SelectionS…
FigureWidget({
    'data': [{'colorbar': {'title': {'text': 'predicted probability'}},
              'colorsca…
```

ML algo	suitable for large datasets?	behaviour wrt outliers	non-linear?	params to tune	smooth predictions	easy to interpret?
linear regression	yes	linear extrapolation	no	l1 and/or l2 reg	yes	yes
logistic regression	yes	scales with distance from the decision boundary	no	l1 and/or l2 reg	yes	yes
random forest	so so	constant	yes	max_features, max_depth	no	so so

ML algo	suitable for large datasets?	behaviour wrt outliers	non-linear?	params to tune	smooth predictions	easy to interpret?
regression						
random forest classification	so so	step-like, difficult to tell	yes	max_features, max_depth	no	so so
SVM rbf regression	tbd	tbd	tbd	tbd	tbd	tbd
SVM rbf classification	tbd	tbd	tbd	tbd	tbd	tbd

Quiz 1

Support Vector Machine

- very versatile technique, it comes in lots of flavors/types, read more about it here
- SVM classifier motivation
  - points in n dimensional space with class 0 and 1
  - we want to find the (n-1) dimensional hyperplane that best separates the points
  - this hyperplane is our (linear) decision boundary
- we cover SVMs with radial basis functions (rbf)
  - we apply a kernel function (a non-linear transformation) to the data points
  - the kernel function basically "smears" the points
  - gaussian rbf kernel: $\exp(-\gamma(|x-x'|)^2)$ where $\gamma > 0$

SVR

In [7]:
```
import numpy as np
from sklearn.svm import SVR
np.random.seed(10)
def true_fun(X):
    return np.cos(1.5 * np.pi * X)

n_samples = 30

X = np.random.rand(n_samples)
y = true_fun(X) + np.random.randn(n_samples) * 0.1

X_new = np.linspace(-0.5, 1.5, 2000)

reg = SVR(gamma = 1, C = 1)
reg.fit(X[:, np.newaxis],y)
y_new = reg.predict(X_new[:, np.newaxis])
```

In [8]:
```
help(SVR)
```

```
Help on class SVR in module sklearn.svm._classes:

class SVR(sklearn.base.RegressorMixin, sklearn.svm._base.BaseLibSVM)
 |  SVR(*, kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=0.001, C=1.
0, epsilon=0.1, shrinking=True, cache_size=200, verbose=False, max_iter=-1)
 |
 |  Epsilon-Support Vector Regression.
 |
 |  The free parameters in the model are C and epsilon.
 |
 |  The implementation is based on libsvm. The fit time complexity
 |  is more than quadratic with the number of samples which makes it hard
 |  to scale to datasets with more than a couple of 10000 samples. For large
 |  datasets consider using :class:`~sklearn.svm.LinearSVR` or
 |  :class:`~sklearn.linear_model.SGDRegressor` instead, possibly after a
 |  :class:`~sklearn.kernel_approximation.Nystroem` transformer or
 |  other :ref:`kernel_approximation`.
 |
 |  Read more in the :ref:`User Guide <svm_regression>`.
 |
 |  Parameters
 |  ----------
 |  kernel : {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callabl
e,          default='rbf'
 |       Specifies the kernel type to be used in the algorithm.
 |       If none is given, 'rbf' will be used. If a callable is given it is
 |       used to precompute the kernel matrix.
 |
 |  degree : int, default=3
 |      Degree of the polynomial kernel function ('poly').
 |      Must be non-negative. Ignored by all other kernels.
 |
 |  gamma : {'scale', 'auto'} or float, default='scale'
 |      Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
 |
 |      - if ``gamma='scale'`` (default) is passed then it uses
 |        1 / (n_features * X.var()) as value of gamma,
 |      - if 'auto', uses 1 / n_features
 |      - if float, must be non-negative.
 |
 |      .. versionchanged:: 0.22
 |         The default value of ``gamma`` changed from 'auto' to 'scale'.
 |
 |  coef0 : float, default=0.0
 |      Independent term in kernel function.
 |      It is only significant in 'poly' and 'sigmoid'.
 |
 |  tol : float, default=1e-3
 |      Tolerance for stopping criterion.
 |
 |  C : float, default=1.0
 |      Regularization parameter. The strength of the regularization is
 |      inversely proportional to C. Must be strictly positive.
 |      The penalty is a squared l2 penalty.
 |
 |  epsilon : float, default=0.1
```

```
|       Epsilon in the epsilon-SVR model. It specifies the epsilon-tube
|       within which no penalty is associated in the training loss function
|       with points predicted within a distance epsilon from the actual
|       value. Must be non-negative.
|
|  shrinking : bool, default=True
|      Whether to use the shrinking heuristic.
|      See the :ref:`User Guide <shrinking_svm>`.
|
|  cache_size : float, default=200
|      Specify the size of the kernel cache (in MB).
|
|  verbose : bool, default=False
|      Enable verbose output. Note that this setting takes advantage of a
|      per-process runtime setting in libsvm that, if enabled, may not work
|      properly in a multithreaded context.
|
|  max_iter : int, default=-1
|      Hard limit on iterations within solver, or -1 for no limit.
|
|  Attributes
|  ----------
|  class_weight_ : ndarray of shape (n_classes,)
|      Multipliers of parameter C for each class.
|      Computed based on the ``class_weight`` parameter.
|
|      .. deprecated:: 1.2
|          `class_weight_` was deprecated in version 1.2 and will be remove
d in 1.4.
|
|  coef_ : ndarray of shape (1, n_features)
|      Weights assigned to the features (coefficients in the primal
|      problem). This is only available in the case of a linear kernel.
|
|      `coef_` is readonly property derived from `dual_coef_` and
|      `support_vectors_`.
|
|  dual_coef_ : ndarray of shape (1, n_SV)
|      Coefficients of the support vector in the decision function.
|
|  fit_status_ : int
|      0 if correctly fitted, 1 otherwise (will raise warning)
|
|  intercept_ : ndarray of shape (1,)
|      Constants in decision function.
|
|  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_iter_ : int
 |      Number of iterations run by the optimization routine to fit the mode
l.
 |
 |      .. versionadded:: 1.1
 |
 |  n_support_ : ndarray of shape (1,), dtype=int32
 |      Number of support vectors.
 |
 |  shape_fit_ : tuple of int of shape (n_dimensions_of_X,)
 |      Array dimensions of training vector ``X``.
 |
 |  support_ : ndarray of shape (n_SV,)
 |      Indices of support vectors.
 |
 |  support_vectors_ : ndarray of shape (n_SV, n_features)
 |      Support vectors.
 |
 |  See Also
 |  --------
 |  NuSVR : Support Vector Machine for regression implemented using libsvm
 |      using a parameter to control the number of support vectors.
 |
 |  LinearSVR : Scalable Linear Support Vector Machine for regression
 |      implemented using liblinear.
 |
 |  References
 |  ----------
 |  .. [1] `LIBSVM: A Library for Support Vector Machines
 |      <http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf>`_
 |
 |  .. [2] `Platt, John (1999). "Probabilistic Outputs for Support Vector
 |      Machines and Comparisons to Regularized Likelihood Methods"
 |      <https://citeseerx.ist.psu.edu/doc_view/pid/42e5ed832d4310ce4378c44d
05570439df28a393>`_
 |
 |  Examples
 |  --------
 |  >>> from sklearn.svm import SVR
 |  >>> from sklearn.pipeline import make_pipeline
 |  >>> from sklearn.preprocessing import StandardScaler
 |  >>> import numpy as np
 |  >>> n_samples, n_features = 10, 5
 |  >>> rng = np.random.RandomState(0)
 |  >>> y = rng.randn(n_samples)
 |  >>> X = rng.randn(n_samples, n_features)
 |  >>> regr = make_pipeline(StandardScaler(), SVR(C=1.0, epsilon=0.2))
 |  >>> regr.fit(X, y)
 |  Pipeline(steps=[('standardscaler', StandardScaler()),
 |                  ('svr', SVR(epsilon=0.2))])
 |
 |  Method resolution order:
 |      SVR
 |      sklearn.base.RegressorMixin
 |      sklearn.svm._base.BaseLibSVM
```

```
 |      sklearn.base.BaseEstimator
 |      sklearn.utils._metadata_requests._MetadataRequester
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=
0.001, C=1.0, epsilon=0.1, shrinking=True, cache_size=200, verbose=False, ma
x_iter=-1)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  set_fit_request(self: sklearn.svm._classes.SVR, *, sample_weight: Union
[bool, NoneType, str] = '$UNCHANGED$') -> sklearn.svm._classes.SVR
 |      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:`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.svm._classes.SVR, *, sample_weight: Unio
n[bool, NoneType, str] = '$UNCHANGED$') -> sklearn.svm._classes.SVR
 |      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:`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.
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties defined here:
 |
 |  class_weight_
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
```

```
 |
 |  __annotations__ = {'_parameter_constraints': <class 'dict'>}
 |
 |  unused_param = 'random_state'
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.RegressorMixin:
 |
 |  score(self, X, y, sample_weight=None)
 |      Return the coefficient of determination of the prediction.
 |
 |      The coefficient of determination :math:`R^2` is defined as
 |      :math:`(1 - \frac{u}{v})`, where :math:`u` is the residual
 |      sum of squares ``((y_true - y_pred)** 2).sum()`` and :math:`v`
 |      is the total sum of squares ``((y_true - y_true.mean()) ** 2).sum()`
`.
 |      The best possible score is 1.0 and it can be negative (because the
 |      model can be arbitrarily worse). A constant model that always predic
ts
 |      the expected value of `y`, disregarding the input features, would ge
t
 |      a :math:`R^2` score of 0.0.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Test samples. For some estimators this may be a precomputed
 |          kernel matrix or a list of generic objects instead with shape
 |          ``(n_samples, n_samples_fitted)``, where ``n_samples_fitted``
 |          is the number of samples used in the fitting for the estimator.
 |
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          True values for `X`.
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Sample weights.
 |
 |      Returns
 |      -------
 |      score : float
 |          :math:`R^2` of ``self.predict(X)`` w.r.t. `y`.
 |
 |      Notes
 |      -----
 |      The :math:`R^2` score used when calling ``score`` on a regressor use
s
 |      ``multioutput='uniform_average'`` from version 0.23 to keep consiste
nt
 |      with default value of :func:`~sklearn.metrics.r2_score`.
 |      This influences the ``score`` method of all the multioutput
 |      regressors (except for
 |      :class:`~sklearn.multioutput.MultiOutputRegressor`).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.RegressorMixin:
 |
```

```
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.svm._base.BaseLibSVM:
 |
 |  fit(self, X, y, sample_weight=None)
 |      Fit the SVM model according to the given training data.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
or (n_samples, n_samples)
 |          Training vectors, where `n_samples` is the number of samples
 |          and `n_features` is the number of features.
 |          For kernel="precomputed", the expected shape of X is
 |          (n_samples, n_samples).
 |
 |      y : array-like of shape (n_samples,)
 |          Target values (class labels in classification, real numbers in
 |          regression).
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Per-sample weights. Rescale C per sample. Higher weights
 |          force the classifier to put more emphasis on these points.
 |
 |      Returns
 |      -------
 |      self : object
 |          Fitted estimator.
 |
 |      Notes
 |      -----
 |      If X and y are not C-ordered and contiguous arrays of np.float64 and
 |      X is not a scipy.sparse.csr_matrix, X and/or y may be copied.
 |
 |      If X is a dense array, then the other methods will not support spars
e
 |      matrices as input.
 |
 |  predict(self, X)
 |      Perform regression on samples in X.
 |
 |      For an one-class model, +1 (inlier) or -1 (outlier) is returned.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          For kernel="precomputed", the expected shape of X is
 |          (n_samples_test, n_samples_train).
 |
 |      Returns
 |      -------
```

```
|      y_pred : ndarray of shape (n_samples,)
|          The predicted values.
|
|  ----------------------------------------------------------------------
|  Readonly properties inherited from sklearn.svm._base.BaseLibSVM:
|
|  coef_
|      Weights assigned to the features when `kernel="linear"`.
|
|      Returns
|      -------
|      ndarray of shape (n_features, n_classes)
|
|  n_support_
|      Number of support vectors for each class.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|      Helper for pickle.
|
|  __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:`~utils.metadata_routing.MetadataRequest` encapsulating
 |          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 [9]:
```
hyperparameters = {
    'gamma': [1e-3, 1e-1, 1e1, 1e3, 1e5],
    'C': [1e-1, 1e0, 1e1]
}

vis(X, y, SVR, hyperparameters, X_new)
```

```
interactive(children=(SelectionSlider(description='gamma', options=(0.001,
0.1, 10.0, 1000.0, 100000.0), value…
```

Quiz 2

Let's measure how long it takes to fit a linear regression, random forest regression, and SVR as a function of `n_samples` using our toy regression dataset.

Check this stackoverflow post to figure out how to measure the execution time of a couple of lines of code.

Set n_estimators to 10 and max_depth to 3 in the random forest.

Set the gamma and C parameters to 1 in SVR.

Fit models with n_samples = 1000, 2000, 3000, 4000, 5000. Measure how long it takes to fit each model.

Plot the run time as a function of n_samples for the three models. You might need to adjust the y axis range to check some of the statements.

Which of these statements are true?

- The random forest run-time scales linearly with n_samples.
- The linear regression model is the fastest to fit.
- The SVR run-time scales worse than linear. (I.e., if we double n_sample, the fit time more than doubles.)

In [ ]:

ML algo	suitable for large datasets?	behaviour wrt outliers	non-linear?	params to tune	smooth predictions	easy to interpret?
linear regression	yes	linear extrapolation	no	l1 and/or l2 reg	yes	yes
logistic regression	yes	scales with distance from the decision boundary	no	l1 and/or l2 reg	yes	yes
random forest regression	so so	constant	yes	max_features, max_depth	no	so so
random forest classification	so so	step-like, difficult to tell	yes	max_features, max_depth	no	so so
SVM rbf regression	no	non-linear extrapolation	yes	C, gamma	yes	so so

ML algo	suitable for large datasets?	behaviour wrt outliers	non-linear?	params to tune	smooth predictions	easy to interpret?
SVM rbf classification	tbd	tbd	tbd	tbd	tbd	tbd

SVC

In [10]:
```
from sklearn.datasets import make_moons
import numpy as np
from sklearn.svm import SVC

# create the data
X,y = make_moons(noise=0.2, random_state=1,n_samples=200)
# set the hyperparameters
clf = SVC(gamma = 1, C = 1, probability=True)
# fit the model
clf.fit(X,y)
# predict new data
#y_new = clf.predict(X_new)
# predict probabilities
#y_new = clf.predict_proba(X_new)
```

Out[10]:
```
SVC(C=1, gamma=1, probability=True)
```

In [11]:
```
help(SVC)
```

```
Help on class SVC in module sklearn.svm._classes:

class SVC(sklearn.svm._base.BaseSVC)
 |  SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinkin
g=True, probability=False, tol=0.001, cache_size=200, class_weight=None, ver
bose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, ra
ndom_state=None)
 |
 |  C-Support Vector Classification.
 |
 |  The implementation is based on libsvm. The fit time scales at least
 |  quadratically with the number of samples and may be impractical
 |  beyond tens of thousands of samples. For large datasets
 |  consider using :class:`~sklearn.svm.LinearSVC` or
 |  :class:`~sklearn.linear_model.SGDClassifier` instead, possibly after a
 |  :class:`~sklearn.kernel_approximation.Nystroem` transformer or
 |  other :ref:`kernel_approximation`.
 |
 |  The multiclass support is handled according to a one-vs-one scheme.
 |
 |  For details on the precise mathematical formulation of the provided
 |  kernel functions and how `gamma`, `coef0` and `degree` affect each
 |  other, see the corresponding section in the narrative documentation:
 |  :ref:`svm_kernels`.
 |
 |  Read more in the :ref:`User Guide <svm_classification>`.
 |
 |  Parameters
 |  ----------
 |  C : float, default=1.0
 |      Regularization parameter. The strength of the regularization is
 |      inversely proportional to C. Must be strictly positive. The penalty
 |      is a squared l2 penalty.
 |
 |  kernel : {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callabl
e,          default='rbf'
 |      Specifies the kernel type to be used in the algorithm.
 |      If none is given, 'rbf' will be used. If a callable is given it is
 |      used to pre-compute the kernel matrix from data matrices; that matri
x
 |      should be an array of shape ``(n_samples, n_samples)``.
 |
 |  degree : int, default=3
 |      Degree of the polynomial kernel function ('poly').
 |      Must be non-negative. Ignored by all other kernels.
 |
 |  gamma : {'scale', 'auto'} or float, default='scale'
 |      Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
 |
 |      - if ``gamma='scale'`` (default) is passed then it uses
 |        1 / (n_features * X.var()) as value of gamma,
 |      - if 'auto', uses 1 / n_features
 |      - if float, must be non-negative.
 |
 |      .. versionchanged:: 0.22
 |         The default value of ``gamma`` changed from 'auto' to 'scale'.
```

```
  |
  |  coef0 : float, default=0.0
  |      Independent term in kernel function.
  |      It is only significant in 'poly' and 'sigmoid'.
  |
  |  shrinking : bool, default=True
  |      Whether to use the shrinking heuristic.
  |      See the :ref:`User Guide <shrinking_svm>`.
  |
  |  probability : bool, default=False
  |      Whether to enable probability estimates. This must be enabled prior
  |      to calling `fit`, will slow down that method as it internally uses
  |      5-fold cross-validation, and `predict_proba` may be inconsistent wit
h
  |      `predict`. Read more in the :ref:`User Guide <scores_probabilities>
`.
  |
  |  tol : float, default=1e-3
  |      Tolerance for stopping criterion.
  |
  |  cache_size : float, default=200
  |      Specify the size of the kernel cache (in MB).
  |
  |  class_weight : dict or 'balanced', default=None
  |      Set the parameter C of class i to class_weight[i]*C for
  |      SVC. If not given, all classes are supposed to have
  |      weight one.
  |      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))``.
  |
  |  verbose : bool, default=False
  |      Enable verbose output. Note that this setting takes advantage of a
  |      per-process runtime setting in libsvm that, if enabled, may not work
  |      properly in a multithreaded context.
  |
  |  max_iter : int, default=-1
  |      Hard limit on iterations within solver, or -1 for no limit.
  |
  |  decision_function_shape : {'ovo', 'ovr'}, default='ovr'
  |      Whether to return a one-vs-rest ('ovr') decision function of shape
  |      (n_samples, n_classes) as all other classifiers, or the original
  |      one-vs-one ('ovo') decision function of libsvm which has shape
  |      (n_samples, n_classes * (n_classes - 1) / 2). However, note that
  |      internally, one-vs-one ('ovo') is always used as a multi-class strat
egy
  |      to train models; an ovr matrix is only constructed from the ovo matr
ix.
  |      The parameter is ignored for binary classification.
  |
  |      .. versionchanged:: 0.19
  |          decision_function_shape is 'ovr' by default.
  |
  |      .. versionadded:: 0.17
  |         *decision_function_shape='ovr'* is recommended.
```

```
  |
  |      .. versionchanged:: 0.17
  |         Deprecated *decision_function_shape='ovo' and None*.
  |
  |  break_ties : bool, default=False
  |      If true, ``decision_function_shape='ovr'``, and number of classes >
2,
  |      :term:`predict` will break ties according to the confidence values o
f
  |      :term:`decision_function`; otherwise the first class among the tied
  |      classes is returned. Please note that breaking ties comes at a
  |      relatively high computational cost compared to a simple predict.
  |
  |      .. versionadded:: 0.22
  |
  |  random_state : int, RandomState instance or None, default=None
  |      Controls the pseudo random number generation for shuffling the data
for
  |      probability estimates. Ignored when `probability` is False.
  |      Pass an int for reproducible output across multiple function calls.
  |      See :term:`Glossary <random_state>`.
  |
  |  Attributes
  |  ----------
  |  class_weight_ : ndarray of shape (n_classes,)
  |      Multipliers of parameter C for each class.
  |      Computed based on the ``class_weight`` parameter.
  |
  |  classes_ : ndarray of shape (n_classes,)
  |      The classes labels.
  |
  |  coef_ : ndarray of shape (n_classes * (n_classes - 1) / 2, n_features)
  |      Weights assigned to the features (coefficients in the primal
  |      problem). This is only available in the case of a linear kernel.
  |
  |      `coef_` is a readonly property derived from `dual_coef_` and
  |      `support_vectors_`.
  |
  |  dual_coef_ : ndarray of shape (n_classes -1, n_SV)
  |      Dual coefficients of the support vector in the decision
  |      function (see :ref:`sgd_mathematical_formulation`), multiplied by
  |      their targets.
  |      For multiclass, coefficient for all 1-vs-1 classifiers.
  |      The layout of the coefficients in the multiclass case is somewhat
  |      non-trivial. See the :ref:`multi-class section of the User Guide
  |      <svm_multi_class>` for details.
  |
  |  fit_status_ : int
  |      0 if correctly fitted, 1 otherwise (will raise warning)
  |
  |  intercept_ : ndarray of shape (n_classes * (n_classes - 1) / 2,)
  |      Constants in decision function.
  |
  |  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_iter_ : ndarray of shape (n_classes * (n_classes - 1) // 2,)
  |      Number of iterations run by the optimization routine to fit the mode
l.
  |      The shape of this attribute depends on the number of models optimize
d
  |      which in turn depends on the number of classes.
  |
  |      .. versionadded:: 1.1
  |
  |  support_ : ndarray of shape (n_SV)
  |      Indices of support vectors.
  |
  |  support_vectors_ : ndarray of shape (n_SV, n_features)
  |      Support vectors.
  |
  |  n_support_ : ndarray of shape (n_classes,), dtype=int32
  |      Number of support vectors for each class.
  |
  |  probA_ : ndarray of shape (n_classes * (n_classes - 1) / 2)
  |  probB_ : ndarray of shape (n_classes * (n_classes - 1) / 2)
  |      If `probability=True`, it corresponds to the parameters learned in
  |      Platt scaling to produce probability estimates from decision values.
  |      If `probability=False`, it's an empty array. Platt scaling uses the
  |      logistic function
  |      ``1 / (1 + exp(decision_value * probA_ + probB_))``
  |      where ``probA_`` and ``probB_`` are learned from the dataset [2]_. F
or
  |      more information on the multiclass case and training procedure see
  |      section 8 of [1]_.
  |
  |  shape_fit_ : tuple of int of shape (n_dimensions_of_X,)
  |      Array dimensions of training vector ``X``.
  |
  |  See Also
  |  --------
  |  SVR : Support Vector Machine for Regression implemented using libsvm.
  |
  |  LinearSVC : Scalable Linear Support Vector Machine for classification
  |      implemented using liblinear. Check the See Also section of
  |      LinearSVC for more comparison element.
  |
  |  References
  |  ----------
  |  .. [1] `LIBSVM: A Library for Support Vector Machines
  |      <http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf>`_
  |
  |  .. [2] `Platt, John (1999). "Probabilistic Outputs for Support Vector
  |      Machines and Comparisons to Regularized Likelihood Methods"
```

```
 |      <https://citeseerx.ist.psu.edu/doc_view/pid/42e5ed832d4310ce4378c44d
05570439df28a393>`_
 |
 |  Examples
 |  --------
 |  >>> import numpy as np
 |  >>> from sklearn.pipeline import make_pipeline
 |  >>> from sklearn.preprocessing import StandardScaler
 |  >>> X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
 |  >>> y = np.array([1, 1, 2, 2])
 |  >>> from sklearn.svm import SVC
 |  >>> clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
 |  >>> clf.fit(X, y)
 |  Pipeline(steps=[('standardscaler', StandardScaler()),
 |                  ('svc', SVC(gamma='auto'))])
 |
 |  >>> print(clf.predict([[-0.8, -1]]))
 |  [1]
 |
 |  Method resolution order:
 |      SVC
 |      sklearn.svm._base.BaseSVC
 |      sklearn.base.ClassifierMixin
 |      sklearn.svm._base.BaseLibSVM
 |      sklearn.base.BaseEstimator
 |      sklearn.utils._metadata_requests._MetadataRequester
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.
0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weigh
t=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_tie
s=False, random_state=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  set_fit_request(self: sklearn.svm._classes.SVC, *, sample_weight: Union
[bool, NoneType, str] = '$UNCHANGED$') -> sklearn.svm._classes.SVC
 |      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:`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.svm._classes.SVC, *, sample_weight: Unio
n[bool, NoneType, str] = '$UNCHANGED$') -> sklearn.svm._classes.SVC
 |      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:`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__ = {}
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.svm._base.BaseSVC:
 |
 |  decision_function(self, X)
 |      Evaluate the decision function for the samples in X.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          The input samples.
 |
 |      Returns
 |      -------
 |      X : ndarray of shape (n_samples, n_classes * (n_classes-1) / 2)
 |          Returns the decision function of the sample for each class
 |          in the model.
 |          If decision_function_shape='ovr', the shape is (n_samples,
 |          n_classes).
 |
 |      Notes
 |      -----
 |      If decision_function_shape='ovo', the function values are proportion
al
 |      to the distance of the samples X to the separating hyperplane. If th
e
 |      exact distances are required, divide the function values by the norm
of
 |      the weight vector (``coef_``). See also `this question
 |      <https://stats.stackexchange.com/questions/14876/
 |      interpreting-distance-from-hyperplane-in-svm>`_ for further details.
 |      If decision_function_shape='ovr', the decision function is a monoton
ic
 |      transformation of ovo decision function.
```

```
 |
 |  predict(self, X)
 |      Perform classification on samples in X.
 |
 |      For an one-class model, +1 or -1 is returned.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features) or
(n_samples_test, n_samples_train)
 |          For kernel="precomputed", the expected shape of X is
 |          (n_samples_test, n_samples_train).
 |
 |      Returns
 |      -------
 |      y_pred : ndarray of shape (n_samples,)
 |          Class labels for samples in X.
 |
 |  predict_log_proba(self, X)
 |      Compute log probabilities of possible outcomes for samples in X.
 |
 |      The model need to have probability information computed at training
 |      time: fit with attribute `probability` set to True.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features) or
(n_samples_test, n_samples_train)
 |          For kernel="precomputed", the expected shape of X is
 |          (n_samples_test, n_samples_train).
 |
 |      Returns
 |      -------
 |      T : ndarray of shape (n_samples, n_classes)
 |          Returns the log-probabilities of the sample for each class in
 |          the model. The columns correspond to the classes in sorted
 |          order, as they appear in the attribute :term:`classes_`.
 |
 |      Notes
 |      -----
 |      The probability model is created using cross validation, so
 |      the results can be slightly different than those obtained by
 |      predict. Also, it will produce meaningless results on very small
 |      datasets.
 |
 |  predict_proba(self, X)
 |      Compute probabilities of possible outcomes for samples in X.
 |
 |      The model need to have probability information computed at training
 |      time: fit with attribute `probability` set to True.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          For kernel="precomputed", the expected shape of X is
 |          (n_samples_test, n_samples_train).
```

```
|
|      Returns
|      -------
|      T : ndarray of shape (n_samples, n_classes)
|          Returns the probability of the sample for each class in
|          the model. The columns correspond to the classes in sorted
|          order, as they appear in the attribute :term:`classes_`.
|
|      Notes
|      -----
|      The probability model is created using cross validation, so
|      the results can be slightly different than those obtained by
|      predict. Also, it will produce meaningless results on very small
|      datasets.
|
|  ----------------------------------------------------------------------
|  Readonly properties inherited from sklearn.svm._base.BaseSVC:
|
|  probA_
|      Parameter learned in Platt scaling when `probability=True`.
|
|      Returns
|      -------
|      ndarray of shape  (n_classes * (n_classes - 1) / 2)
|
|  probB_
|      Parameter learned in Platt scaling when `probability=True`.
|
|      Returns
|      -------
|      ndarray of shape  (n_classes * (n_classes - 1) / 2)
|
|  ----------------------------------------------------------------------
|  Data and other attributes inherited from sklearn.svm._base.BaseSVC:
|
|  unused_param = 'nu'
|
|  ----------------------------------------------------------------------
|  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 sklearn.svm._base.BaseLibSVM:
 |
 |  fit(self, X, y, sample_weight=None)
 |      Fit the SVM model according to the given training data.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
or (n_samples, n_samples)
 |          Training vectors, where `n_samples` is the number of samples
 |          and `n_features` is the number of features.
 |          For kernel="precomputed", the expected shape of X is
 |          (n_samples, n_samples).
 |
 |      y : array-like of shape (n_samples,)
 |          Target values (class labels in classification, real numbers in
 |          regression).
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Per-sample weights. Rescale C per sample. Higher weights
 |          force the classifier to put more emphasis on these points.
 |
 |      Returns
 |      -------
 |      self : object
 |          Fitted estimator.
 |
 |      Notes
 |      -----
 |      If X and y are not C-ordered and contiguous arrays of np.float64 and
 |      X is not a scipy.sparse.csr_matrix, X and/or y may be copied.
 |
 |      If X is a dense array, then the other methods will not support spars
e
 |      matrices as input.
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties inherited from sklearn.svm._base.BaseLibSVM:
 |
```

```
|  coef_
|      Weights assigned to the features when `kernel="linear"`.
|
|      Returns
|      -------
|      ndarray of shape (n_features, n_classes)
|
|  n_support_
|      Number of support vectors for each class.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|      Helper for pickle.
|
|  __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:`~utils.metadata_routing.MetadataRequest` encapsulating
 |          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 [12]:

```python
# initialize RandomForestClassifier
ML_algo = SVC(probability=True)

# SVC parameter grid
hyperparameters = {
    'gamma': [1e-3, 1e-1, 1e1, 1e3, 1e5],
    'C': [1e-1, 1e0, 1e1]
}

plot_clf_contour(hyperparameters, X, y)
```

```
interactive(children=(SelectionSlider(description='gamma', options=(0.001,
0.1, 10.0, 1000.0, 100000.0), value…
```

```
FigureWidget({
    'data': [{'colorbar': {'title': {'text': 'predicted probability'}},
              'colorsca…
```

ML algo	suitable for large datasets?	behaviour wrt outliers	non-linear?	params to tune	smooth predictions	easy to interpret?
linear regression	yes	linear extrapolation	no	l1 and/or l2 reg	yes	yes
logistic regression	yes	scales with distance from the decision boundary	no	l1 and/or l2 reg	yes	yes
random forest regression	so so	constant	yes	max_features, max_depth	no	so so
random forest classification	so so	step-like, difficult to tell	yes	max_features, max_depth	no	so so
SVM rbf regression	no	non-linear extrapolation	yes	C, gamma	yes	so so
SVM rbf classification	no	50-50	yes	C, gamma	yes	so so

Quiz 3

Bias variance trade off

Which gamma value gives the best trade off between high bias and high variance? Work through the steps to answer the question.

- Use random_state = 42 where-ever necessary.
- Split X, y into X_train, X_val, y_train, y_val such that 70% of the points are in train.
- Fit SVC models with C = 1, and gamma = 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3 on the training set.
- Measure the validation accuracy for each gamma.
- Which gamma value gives the highest validation accuracy?

In [ ]:

Mud card