In [9]:
```
import numpy as np
import seaborn as sns
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
```

In [ ]:
```
```

In [5]:
```
data=sns.load_dataset('iris')
```

In [7]:
```
data
```

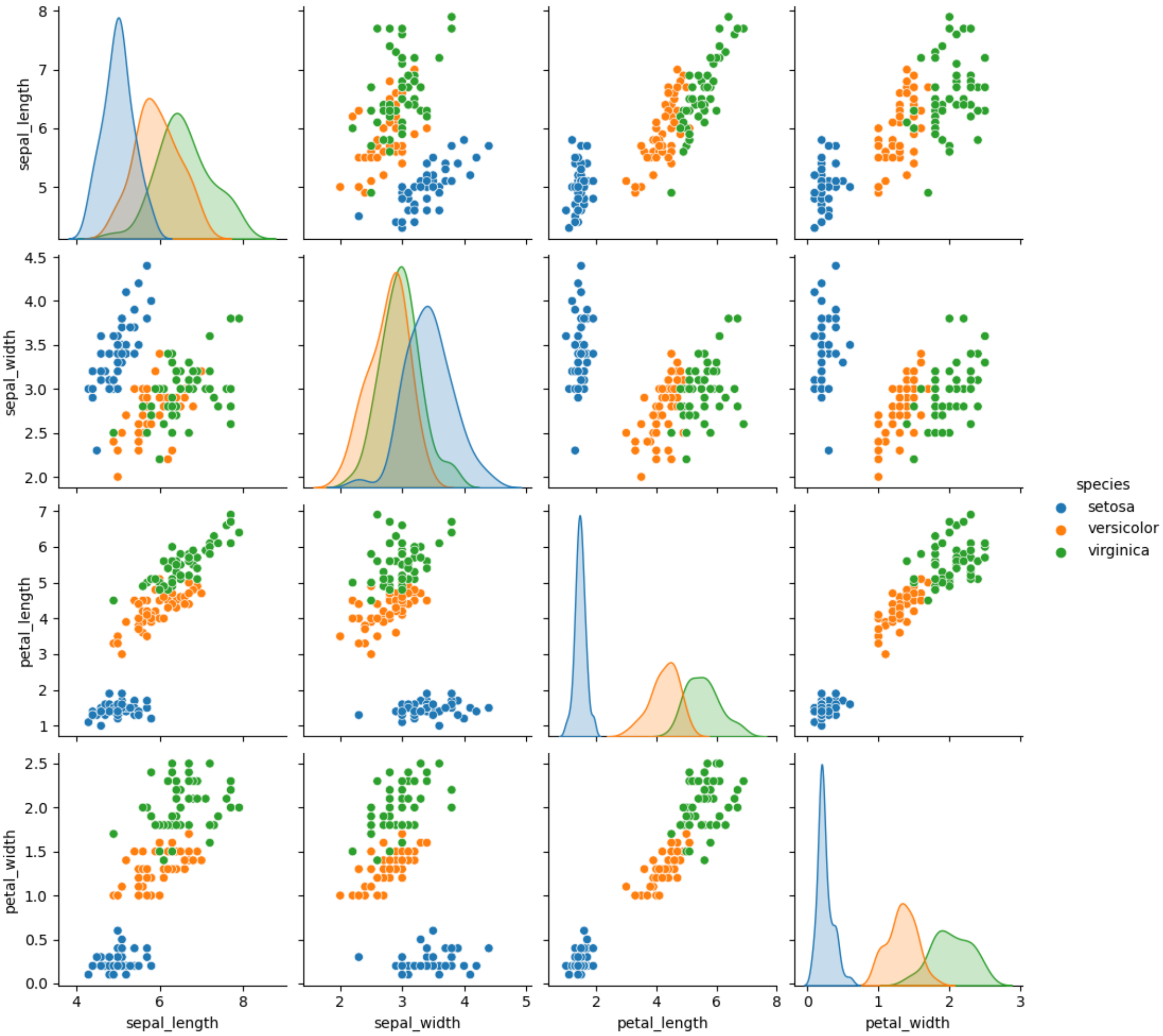
Out[7]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [9]:
```
sns.pairplot(data,hue='species')
```

Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x1f665db29b0>
```



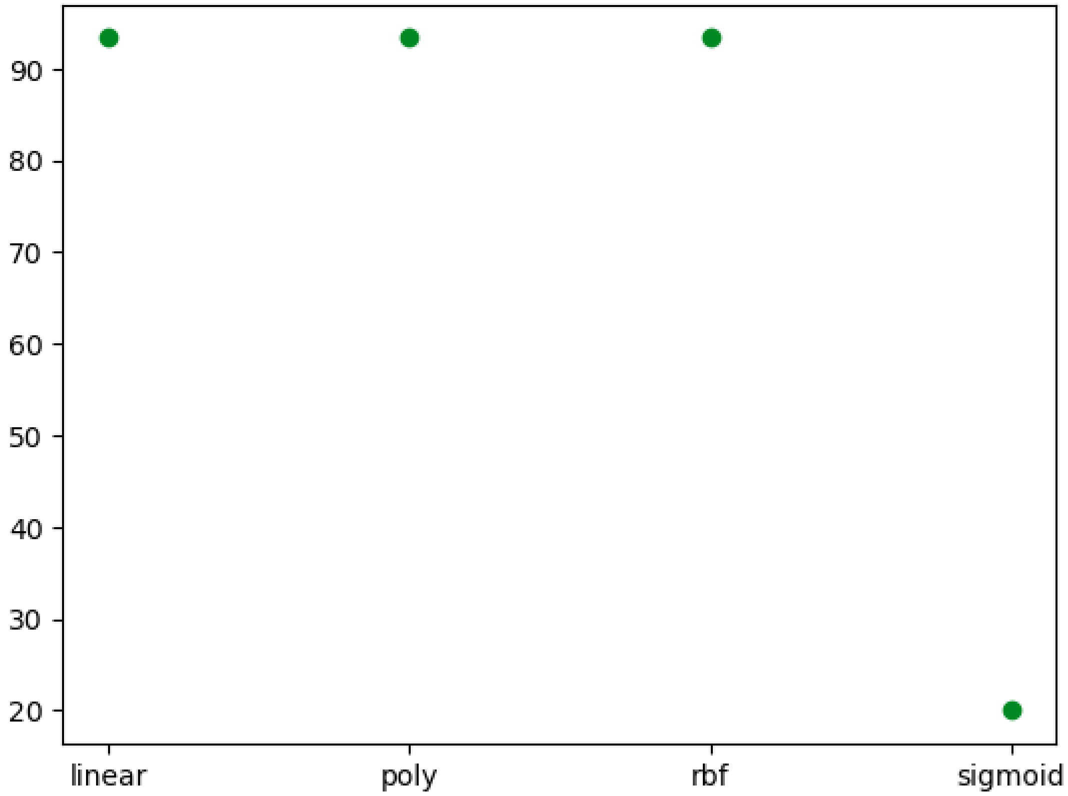
In [15]:
```
x=data[['sepal_length','sepal_width','petal_length','petal_width']]
y=data['species']
```

In [16]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

In [ ]:
```
#svc(linear', 'poly', 'rbf', 'sigmoid)
```

In [74]:
```
for i in ('linear', 'poly', 'rbf', 'sigmoid'):
    dtree=SVC(kernel=i)
    dtree.fit(x_train,y_train)
    pred=dtree.predict(x_test)
    pred
    mt=confusion_matrix(y_test,pred)
    print(mt)
    x=mt[0][0]+mt[1][1]+mt[2][2]
    #y=mt[0][0]+mt[0][1]+mt[0][2]+mt[1][0]+mt[1][1]+mt[1][2]+mt[2][0]+mt[2][1]+mt[2
    y=np.sum(mt)
    xy=x*100/y
    print(x*100/y)
    print(plt.scatter(i,xy,color='#008922'))
```

```
[[ 9  0  0]
 [ 0 15  3]
 [ 0  0 18]]
93.33333333333333
<matplotlib.collections.PathCollection object at 0x000001F66AFFE8C0>
[[ 9  0  0]
 [ 0 15  3]
 [ 0  0 18]]
93.33333333333333
<matplotlib.collections.PathCollection object at 0x000001F66AFFEB90>
[[ 9  0  0]
 [ 0 15  3]
 [ 0  0 18]]
93.33333333333333
<matplotlib.collections.PathCollection object at 0x000001F66AFFF280>
[[ 9  0  0]
 [18  0  0]
 [18  0  0]]
20.0
<matplotlib.collections.PathCollection object at 0x000001F66AFFF6D0>
```



In [59]:
```
#pred=dtree.predict(x_test)
```

In [60]:
```
#pred
```

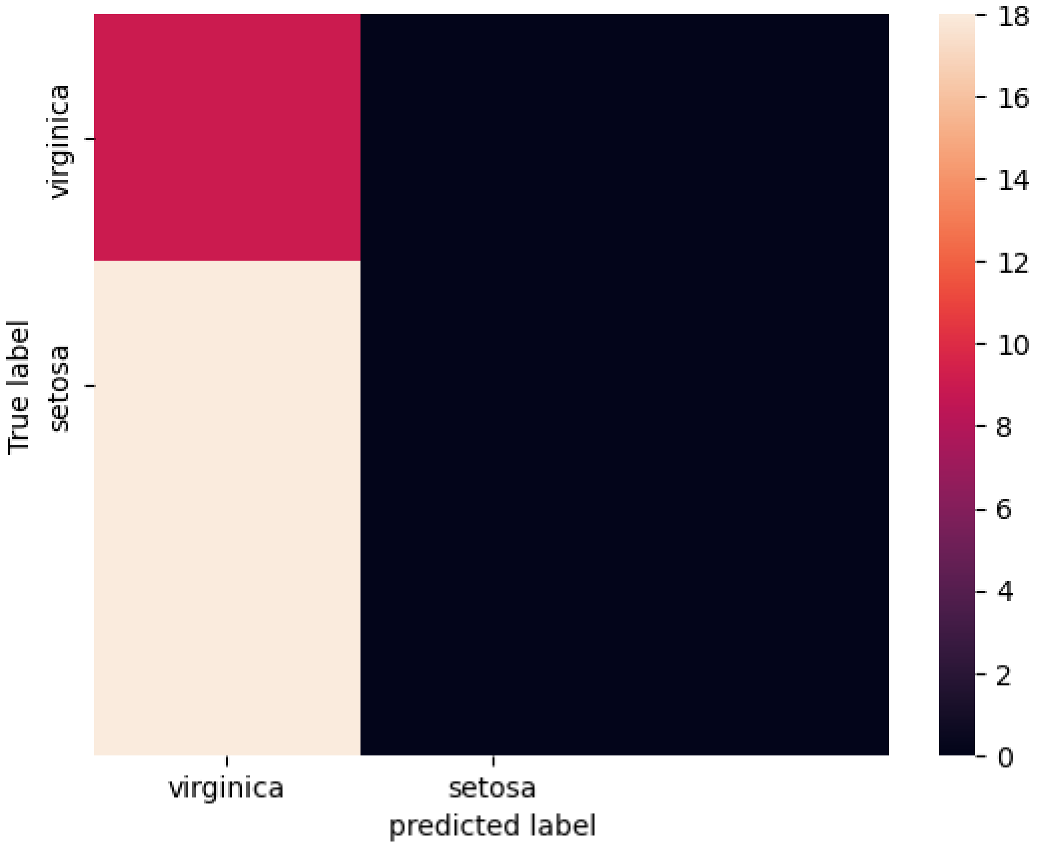
Out[60]:
```
array(['setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa'], dtype=object)
```

In [61]:
```
confusion_matrix(y_test,pred)
```

```
Out[61]: array([[ 9,  0,  0],
                [18,  0,  0],
                [18,  0,  0]], dtype=int64)
```

```
In [62]: plt.figure()
         sns.heatmap(confusion_matrix(y_test,pred),xticklabels=['virginica','setosa'],ytickl
         plt.xlabel('predicted label')
         plt.ylabel('True label')
```

```
Out[62]: Text(50.722222222222214, 0.5, 'True label')
```




```
In [63]: #(42*100)/45
         (9*100)/45
```

```
Out[63]: 20.0
```

```
In [38]: 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, shrinking=Tru
e, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False,
max_iter=-1, decision_function_shape='ovr', break_ties=False, random_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 callable,
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 matrix
 |      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 with
 |      `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 data
 |      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 strategy
 |      to train models; an ovr matrix is only constructed from the ovo matrix.
 |      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 of
 |      :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 model.
 |      The shape of this attribute depends on the number of models optimized
 |      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]_. For
 |      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/42e5ed832d4310ce4378c44d055704
39df28a393>`_
 |
 |  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
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, *, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shr
inking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verb
ose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_st
ate=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  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 proportional
 |      to the distance of the samples X to the separating hyperplane. If the
 |      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 monotonic
 |      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_samp
les_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)`` wrt. `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 sparse
 |      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)
 |
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  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.
```

In [4]:

```
p=0
for i in range(1,11):
    d=i*i
    p=p+(1/d)
print(p)
```

```
1.5497677311665408
```

In [103…

```
p1=0
for i in range(2,11):
```

```
        d=np.math.factorial(i)
        p1=p1+((-1)**i)/d
print(1+p1)
```

1.3678794642857144

In [84]:
```
print((-1)**2)
```

1

In [ ]:
```

```

In [ ]:
```

```