Support Vectors Classifier

class sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, 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)

```
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]
```

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'}, default='rbf'* -** *Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used.*
- **Degree:*int, default=3* -** *Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.*
- **Gamma:*{'scale', 'auto'} or float, default='scale'* -** *Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.*
- **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.*
- ***Probobaility:bool,default=false* -** *Whether to enable probability estimates.*
- **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.*
- **Verbose:*bool, default=False* -** Enable verbose output.
- **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, one-vs-one ('ovo') is always used as multi-class strategy. The parameter is ignored for binary classification.*
- **break_ties:*bool, default=False* -** *If true, decision_function_shape='ovr', and number of classes > 2, predict will break ties according to the confidence values of decision_function; otherwise the first class among the tied classes is returned.*
- **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.*

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.*
- **dual_coef_ : ndarray of shape (n_classes -1, n_SV) -** *Dual coefficients of the support vector in the decision function multiplied by their targets. For multiclass, coefficient for all 1-vs-1 classifiers.*
- **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.*
- **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.*
- **shape_fit_ : tuple of int of shape (n_dimensions_of_X,) -** *Array dimensions of training vector X.*

How sklearn handles SVMs:

- Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.
- The advantages of support vector machines are:

1. Effective in high dimensional spaces.
2. Still effective in cases where number of dimensions is greater than the number of samples.
3. Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
4. Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

- The disadvantages of support vector machines include:

1. If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
2. SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

- The support vector machines in scikit-learn support both dense (numpy.ndarray and convertible to that by numpy.asarray) and sparse (any scipy.sparse) sample vectors as input.
- However, to use an SVM to make predictions for sparse data, it must have been fit on such data.
- For optimal performance, use C-ordered numpy.ndarray (dense) or scipy.sparse.csr_matrix (sparse) with dtype=float64.