**Support vector machines**

**SVMs** are a set of supervised learning methods used for [classification](https://scikit-learn.org/stable/modules/svm.html#svm-classification), [regression](https://scikit-learn.org/stable/modules/svm.html#svm-regression) and [outliers detection](https://scikit-learn.org/stable/modules/svm.html#svm-outlier-detection).

The advantages of support vector machines are:

* Effective in high dimensional spaces.
* Still effective in cases where number of dimensions is greater than the number of samples.
* Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
* Versatile: different [Kernel functions](https://scikit-learn.org/stable/modules/svm.html#svm-kernels) 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:

* If the number of features is much greater than the number of samples, avoid over-fitting in choosing [Kernel functions](https://scikit-learn.org/stable/modules/svm.html#svm-kernels) and regularization term is crucial.
* SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see [Scores and probabilities](https://scikit-learn.org/stable/modules/svm.html#scores-probabilities), below).

[**SVC**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) and **[NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \o "sklearn.svm.NuSVC)** are similar methods, but accept slightly different sets of parameters and have different mathematical formulations (see section [Mathematical formulation](https://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation)). On the other hand, **[LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \o "sklearn.svm.LinearSVC)** is another implementation of Support Vector Classification for the case of a linear kernel. Note that **[LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \o "sklearn.svm.LinearSVC)** does not accept keyword kernel, as this is assumed to be linear. It also lacks some of the members of [**SVC**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) and **[NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \o "sklearn.svm.NuSVC)**, like support\_.

As other classifiers, [**SVC**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC), **[NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \o "sklearn.svm.NuSVC)** and **[LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \o "sklearn.svm.LinearSVC)** take as input two arrays: an array X of size [n\_samples, n\_features]holding the training samples, and an array y of class labels (strings or integers), size [n\_samples]

SVMs decision function depends on some subset of the training data, called the support vectors. Some properties of these support vectors can be found in members support\_vectors\_, support\_ and n\_support:

Note: The LinearSVC class regularizes the bias term, so you should center the training set first by subtracting its mean. This is automatic if you scale the data using the StandardScaler. Moreover, make sure you set the loss hyperparameter to "hinge", as it is not the default value. Finally, for better performance you should set the dual hyperparameter to False, unless there are more features than training instances (we will discuss duality later in the chapter).

**>>>** *# get support vectors*

**>>>** clf.support\_vectors\_

array([[0., 0.],

[1., 1.]])

**>>>** *# get indices of support vectors*

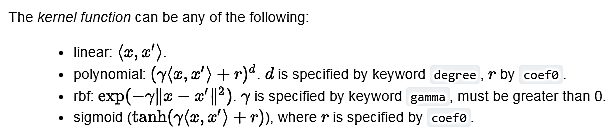
**>>>** clf.support\_

array([0, 1]...)

**>>>** *# get number of support vectors for each class*

**>>>** clf.n\_support\_

array([1, 1]...)



**Polynomial Kernel**

Adding polynomial features is simple to implement and can work great with all sorts of Machine Learning algorithms (not just SVMs), but at a low polynomial degree it cannot deal with very complex datasets, and with a high polynomial degree it creates a huge number of features, making the model too slow. Fortunately, when using SVMs you can apply an almost miraculous mathematical technique called the kernel trick (it is explained in a moment). It makes it possible to get the same result as if you added many polynomial features, even with very high degree polynomials, without actually having to add them. So there is no combinatorial explosion of the number of features since you don’t actually add any features. This trick is implemented by the SVC class.

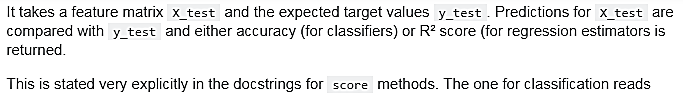
**Parameters:**

* 1. **C : (float)** Penalty parameter C of the error term
  2. **kernel : *string, optional (default=’rbf’)***
  3. **degree : *int, optional (default=3)*** Degree of the polynomial kernel function (‘poly’). Ignored by all other kernels.
  4. **gamma : *float, optional (default=’auto’)*** Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’
  5. **probability : *boolean, optional (default=False)*** Whether to enable probability estimates. This must be enabled prior to calling [**fit**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.fit), and will slow down that method.
  6. **max\_iter : *int, optional (default=-1)*** Hard limit on iterations within solver, or -1 for no limit.
  7. **coef0 : *float, optional (default=0.0)*** Independent term in kernel function. It is only significant in ‘poly’ and ‘sigmoid’.

**Methods**

|  |  |
| --- | --- |
| [**decision\_function**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.decision_function)(self, X) | Evaluates the decision function for the samples in X. |
| [**fit**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.fit)(self, X, y[, sample\_weight]) | Fit the SVM model according to the given training data. |
| [**get\_params**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.get_params)(self[, deep]) | Get parameters for this estimator. |
| [**predict**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.predict)(self, X) | Perform classification on samples in X. |
| [**score**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.score)(self, X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| [**set\_params**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.set_params)(self, \\*\\*params) | Set the parameters of this estimator. |

**About Score method in every predictor:**



**For SVR**

**epsilon : *float, optional (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.

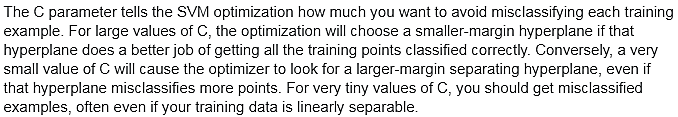
**In Scikit learn, under SVM, there are multiple algos.**

1. **sklearn.svm.SVC : SVM Classifier**
2. **sklearn.svm.NuSVC**
3. **sklearn.svm.LearnSVC**
4. **sklearn.svm.SVR : Regressor**
5. **sklearn.svm.NuSVR**
6. **sklearn.svm.LearnSVR**

**Multi-Class Classification**

[**SVC**](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) and **[NuSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html" \l "sklearn.svm.NuSVC" \o "sklearn.svm.NuSVC)** implement the “one-against-one” approach (Knerr et al., 1990) for multi- class classification. If n\_class is the number of classes, then n\_class \* (n\_class - 1) / 2 classifiers are constructed and each one trains data from two classes. To provide a consistent interface with other classifiers, the decision\_function\_shape option allows to monotically transform the results of the “one-against-one” classifiers to a decision function of shape (n\_samples, n\_classes).

On the other hand, **[LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html" \l "sklearn.svm.LinearSVC" \o "sklearn.svm.LinearSVC)** implements “one-vs-the-rest” multi-class strategy, thus training n\_class models. If there are only two classes, only one model is trained.



**SVM Regressor:**

As we mentioned earlier, the SVM algorithm is quite versatile: not only does it support linear and nonlinear classification, but it also supports linear and nonlinear regression. The trick is to reverse the objective: instead of trying to fit the largest possible street between two classes while limiting margin violations, SVM Regression tries to fit as many instances as possible on the street while limiting margin violations (i.e., instances off the street). The width of the street is controlled by a hyperparameter ϵ. Figure 5-10 shows two linear SVM Regression models trained on some random linear data, one with a large margin (ϵ = 1.5) and the other with a small margin (ϵ = 0.5).

So basically in classifier, it is tried no of violations (points inside street) is as less as possible. In regression, it is tried more the number of points inside street, better. Also model is not affected if new points are added inside this street. Now this line, with equal width on both sides, is linear or nonlinear depends on type of kernel used. With nonlinear kernel this street becomes nonlinear.

**Scaling in SVM or any Algo in ML:**

