Implementación del algoritmo SVM en scikit-learn

El clasificador SVM permite configurar el tipo de kernel a usar. Dependiendo del mismo obtenemos fronteras de decisión distintos.

Class	Purpose	Hyperparameters
`sklearn.svm.SVC`	The LIBSVM implementation for binary and multiclass linear and kernel classification	C, kernel, degree, gamma
`sklearn.svm.NuSVC`	same as above	nu, kernel, degree, gamma
`sklearn.svm.OneClassSVM`	Unsupervised detection of outliers	nu, kernel, degree, gamma
`sklearn.svm.LinearSVC`	Based on LIBLINEAR, it is a binary and multiclass linear classifier	Penalty, loss, C

Entre todos los parámetros podemos destacar:

kernel: SVM se puede ajustar a una serie de funciones kernel linear, poly, rbf, sigmoid. El más utilizado es rbf.

gamma: Este es un coeficiente para 'rbf', 'poly' y 'sigmoid'; Los valores altos tienden a ajustarse mejor a los datos

Como un ejemplo para la clasificación básica y regresión utilizando SVC y SVR dentro del módulo sklearn.svm de Scikit-learn, trabajaremos con los conjuntos de datos Iris y Boston.

Entonces, podemos entrenar nuestro conjunto de datos mediante la clase SVC con la función kernel RBF (C y gamma fueron elegidos sobre la base de otros ejemplos conocidos) y probamos los resultados utilizando la función **cross_val_score** :

```
#cargamos dataset

from sklearn import datasets
iris = datasets.load_iris()
X, y = iris.data, iris.target

from sklearn.svm import SVC
from sklearn.cross_validation import cross_val_score
import numpy as np

classifier = SVC(kernel='rbf', C=1.0, gamma=0.7, random_state=101)

scores = cross_val_score(classifier, X, y, cv=20, scoring='accuracy')

print 'Accuracy: %0.3f' % np.mean(scores)
```

Output: Accuracy: 0.969

Paso a paso

1.Instanciar objeto svm mediante la inicialización del parámetro C

```
svm = LinearSVC(C=0.1)
```

2.Entrenar nuestro modelo

```
svm.fit(X_train, y_train)
```

```
LinearSVC(C=0.1, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)
```

3.Predecir nuestro modelo

```
print(svm.predict(X_train))
print(y_train)
```

```
[1 5 3 ..., 9 3 5]
[1 5 3 ..., 9 3 5]
```

4.Evaluar nuestro modelo sobre los datos de entrenamiento y sobre los datos de test de forma separada

```
svm.score(X_train, y_train)
```

```
0.99628804751299183
```

```
svm.score(X_test, y_test)
```

```
0.94666666666666666
```