Práctica 3: Máquina Vector Soporte

noviembre de 2020

Está práctica está enfocada a la aplicación práctica de la Máquina Vector Soporte (SVM) a partir de librerías ya definidas y que nos facilitarán la asimilación de los conceptos sin perdernos en las líneas de la implementación.

Al realizar la práctica sobre el uso de una librería, tenemos que elegir un lenguaje que, en este caso, ha sido Python. Python posee la librería de SVM y un conjunto más que nos va a permitir representaciones gráficas interesantes y aclaratorias. La librería SVM a usar se encuentra dentro del paquete `scikit-learn` que contiene bastantes algoritmos de aprendizaje automático.

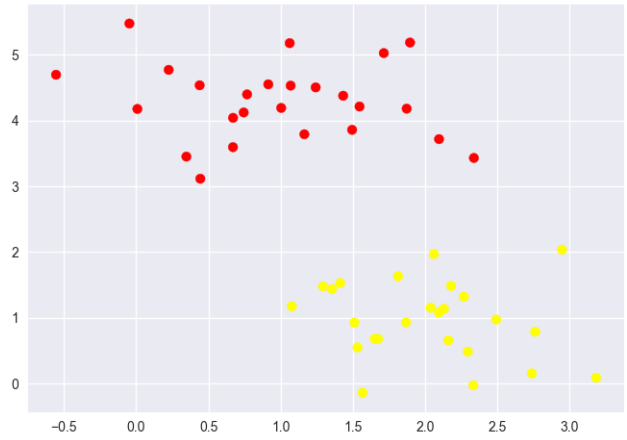
1. Arrancando

Comenzamos importando las clases:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# use seaborn plotting defaults
import seaborn as sns; sns.set()
```

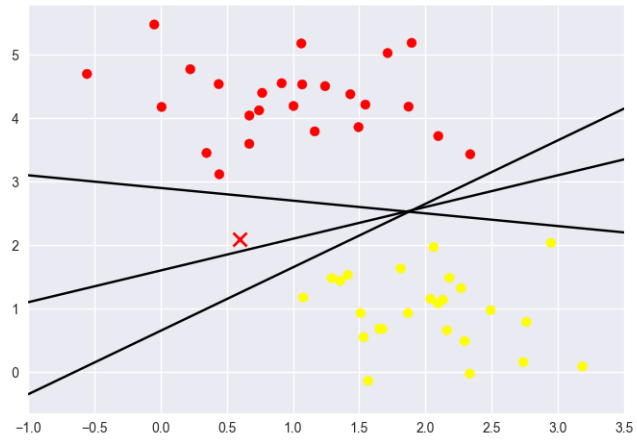
Vamos a considerar un problema de clasificación, en los que las dos clases están perfectamente separadas.

```
from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples=50, centers=2,
                  random_state=0, cluster_std=0.60)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
plt.show()
```



Un clasificador lineal intentaría dibujar una linea recta separando las dos clases de datos, creando así un clasificador. Como hemos visto en teoría se podrían hacer más de una línea de separación:

```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plt.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10)

for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:
    plt.plot(xfit, m * xfit + b, '-k')

plt.xlim(-1, 3.5);
```

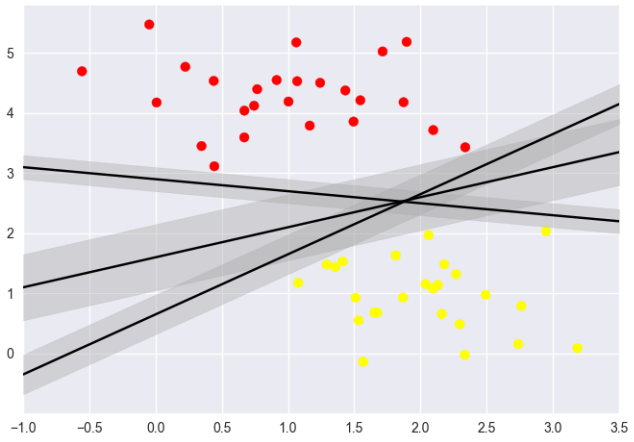


Hay 3 líneas que serían 3 clasificadores que funcionarían perfectamente para los datos de entrenamiento pero darían resultados distintos para el datos de evaluación (la "X"). Tenemos que encontrar la mejor de ellas.

2. Maximizar el margen

Vamos a pintar los márgenes de cada una de las líneas clasificadoras hasta el punto más cercano de uno de las clases:

```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
    yfit = m * xfit + b
    plt.plot(xfit, yfit, '-k')
    plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
                     color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5);
plt.show()
```



El clasificador ideal es el que maximiza este margen.

3. Creando la SVM

Vamos a hallar una máquina SVM que nos resuelva este problema y usaremos el `Scikit-Learn's support vector classifier` citado anteriormente, con el conjunto de datos actual.

Vamos a configurar la máquina con un kernel lineal y un Ç"muy grande.

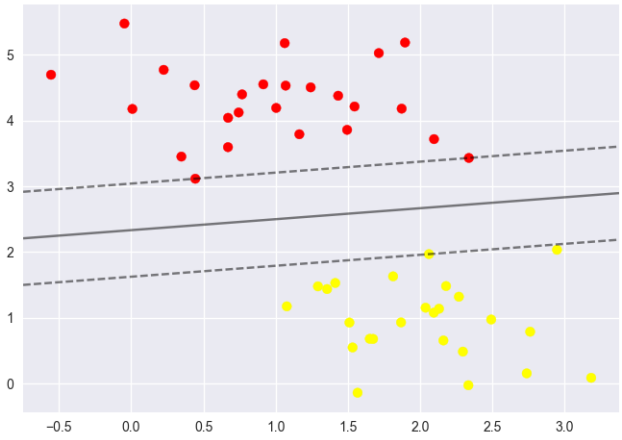
```
from sklearn.svm import SVC # "Support vector classifier"
model = SVC(kernel='linear', C=1E10)
model.fit(X, y)
```

```
SVC(C=10000000000.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

Para ver el resultado, vamos a crear una función que nos dibuje las superficies de decisión:

```
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
```

```
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(model)
plt.show()
```



Esa es la línea que maximiza los márgenes entre los dos conjuntos de puntos. Como se ve, un pequeño conjunto de puntos son los que tocan el margen: los que están rodeados por un círculo negro. Éstos son los *vectores soporte*. Imprimimos los vectores soporte:

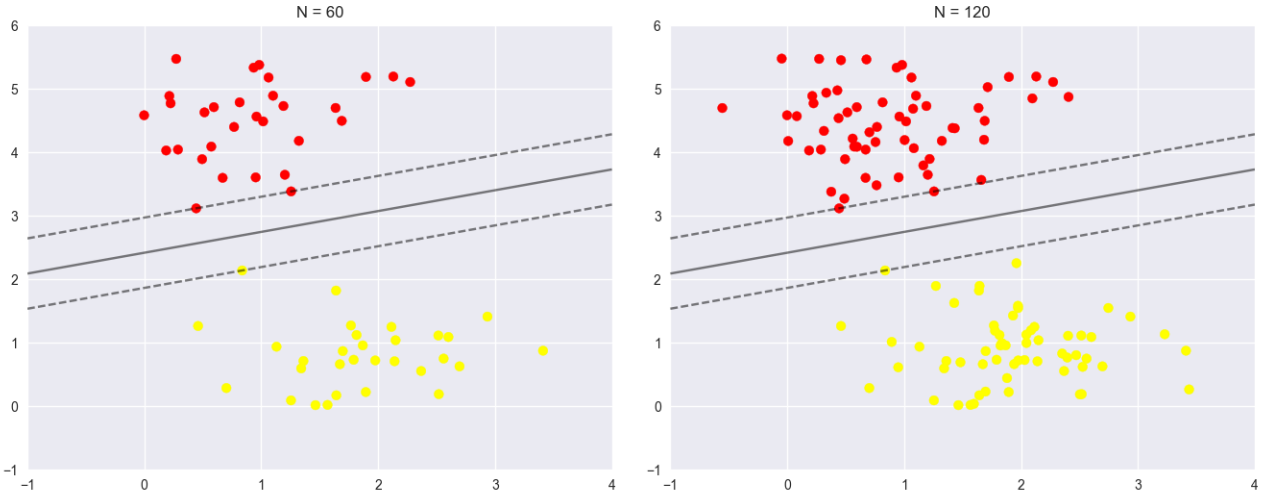
```
print model.support_vectors_

array([[ 0.44359863,  3.11530945],
       [ 2.33812285,  3.43116792],
       [ 2.06156753,  1.96918596]])
```

El buen funcionamiento de la SVM depende de básicamente de la posición de los vectores soporte; cualquier punto que esté más atrás de éstos no influye en los resultados. Técnicamente, los puntos interiores no contribuyen a la construcción del modelo, por lo que ni la cantidad ni la posición son importantes.

Para verlo gráficamente, pintamos dos gráficas, con 60 y 120 puntos respectivamente:

```
def plot_svm(N=10, ax=None):
    X, y = make_blobs(n_samples=200, centers=2,
                      random_state=0, cluster_std=0.60)
    X = X[:N]
    y = y[:N]
    model = SVC(kernel='linear', C=1E10)
    model.fit(X, y)

    ax = ax or plt.gca()
    ax.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
    ax.set_xlim(-1, 4)
    ax.set_ylim(-1, 6)
```

```
    plot_svc_decision_function(model, ax)

fig, ax = plt.subplots(1, 2, figsize=(16, 6))
fig.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)
for axi, N in zip(ax, [60, 120]):
    plot_svm(N, axi)
    axi.set_title('N = {0}'.format(N))
plt.show()
```



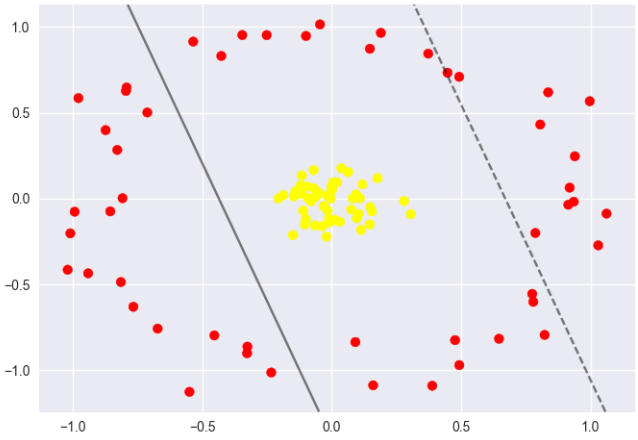
En la figura de la izquierda tenemos el modelo de entrenamiento de 60 puntos y en la derecha de 120, pero el modelo no cambia: los vectores soporte del modelo son los mismos. Esto es uno de los puntos fuertes de las máquinas SVM

4. Kernel SVM

Donde las máquinas SVM son verdaderamente potentes son cuando se combinan con funciones kernel no lineales. Vamos a ver como proyectar en un espacio de mayor dimensionalidad y poder emplear un clasificador lineal.

Vamos a ver un ejemplo de un dataset no-lineal con una función kernel lineal:

```
from sklearn.datasets.samples_generator import make_circles
X, y = make_circles(100, factor=.1, noise=.1)

clf = SVC(kernel='linear').fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf, plot_support=False);
plt.show()
```



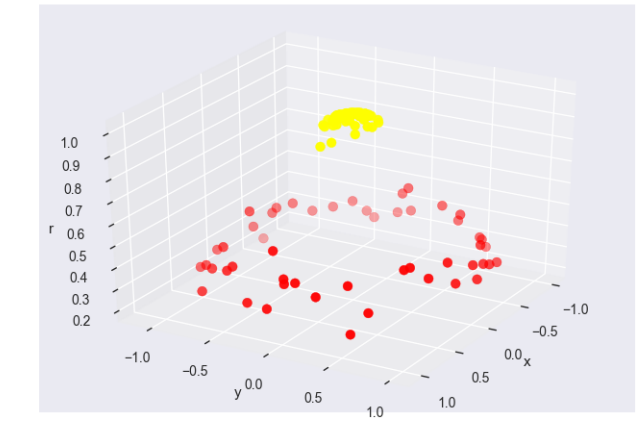
Claramente, esta SVM no es capaz de clasificar correctamente los datos y vamos a necesitar una función kernel distinta y propyectar los datos a un espacio de características de mayor dimensionalidad.

Por ejemplo, usando la función "r" de base radial y de centro en el centro del conjunto de datos, le vamos a añadir una columna con el resultado de esa función:

```
r = np.exp(-(X ** 2).sum(1))
```

Visualizamos esa dimensión extra usando una representación 3-dimensional:

```
from mpl_toolkits import mplot3d

def plot_3D(elev=30, azim=30, X=X, y=y):
    ax = plt.subplot(projection='3d')
    ax.scatter3D(X[:, 0], X[:, 1], r, c=y, s=50, cmap='autumn')
    ax.view_init(elev=elev, azim=azim)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('r')
plot_3D()
plt.show()
```



Como se puede ver, esta nueva dimensión hace trivial la separación lineal de las clases

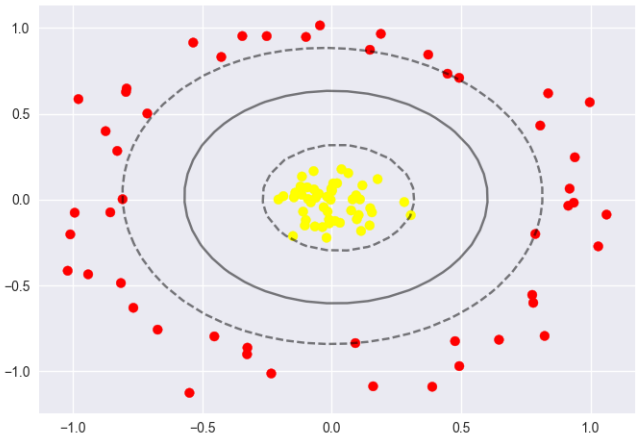
trazando, por ejemplo, un plano de ecuación $r = 0.7$

Un punto importante de esta proyección es elegir bien la posición del centro, ya que si no lo hacemos la división puede no ser tan clara y no resultar linealmente separables. En general, esa elección es en sí un problema y lo ideal es encontrar una forma automatizada de encontrar la mejor función kernel posible.

Una posible estrategia es aplicar esta función base centrada en cada uno de los puntos del dataset, y que sea el algoritmo de la SVM la que decida cuál es la mejor opción. Esto puede provocar que tengamos que realizar una cantidad de proyecciones inmensa sobre un número muy grande de datos, por lo que el cálculo se podría hacer inviable. Hay formas de evitar esto[1], pero se sale del propósito de esta práctica.

En `Scikit-Learn`, podemos crear máquinas SVM con otros kernels, simplemente cambiando el parámetro de llamada por un "`RBF`":

```
clf = SVC(kernel='rbf', C=1E6)
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
            s=300, lw=1, facecolors='none');
plt.show()
```



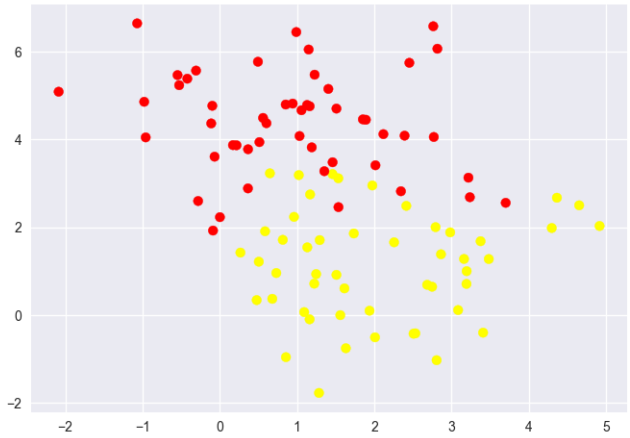
Usando este kernel, aprendemos un contorno de superficie no lineal y que se adapta a los datos. Esta técnica es muy usada para convertir en lineales los datos no lineales.

5. Márgenes blandos

Hasta ahora hemos estado trabajando con dataset muy limpios y claramente separables. En la realidad, los datos suelen solapar, obteniendo algo parecido a esto:

```
X, y = make_blobs(n_samples=100, centers=2,
                  random_state=0, cluster_std=1.2)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
```

[1] `https://en.wikipedia.org/wiki/Kernel_trick`

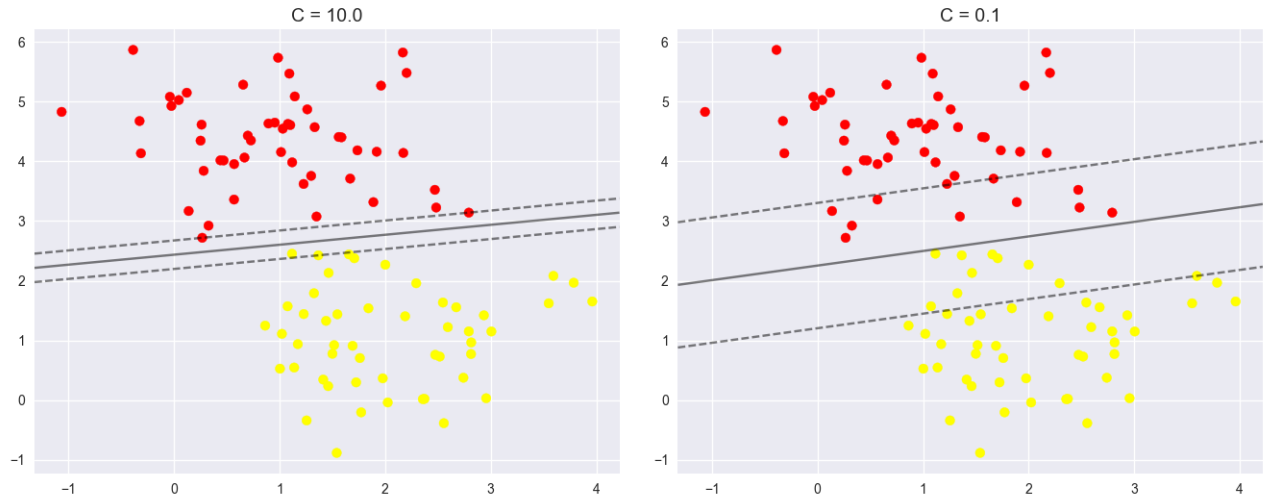


Para tratar esto, la SVM debe "ablandar" los márgenes, es decir, permitir que haya algunos puntos que crucen los márgenes de forma que nos permita una mejor tarea de clasificación global.

El parámetro que controla esta dureza del margen es `C`. Para valores muy altos, el margen se vuelve rígido y tendríamos el caso anterior y para valores muy pequeños puede haber demasiados puntos mal clasificados.

La gráfica siguiente da una imagen de cómo afecta la elección de `C` en el resultado final.

```
X, y = make_blobs(n_samples=100, centers=2,
                  random_state=0, cluster_std=0.8)

fig, ax = plt.subplots(1, 2, figsize=(16, 6))
fig.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)

for axi, C in zip(ax, [10.0, 0.1]):
    model = SVC(kernel='linear', C=C).fit(X, y)
    axi.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
    plot_svc_decision_function(model, axi)
    axi.scatter(model.support_vectors_[:, 0],
                model.support_vectors_[:, 1],
                s=300, lw=1, facecolors='none');
    axi.set_title('C = {0:.1f}'.format(C), size=14)
plt.show()
```



El valor óptimo de C va a depender del dataset y debería ser adaptado usando `cross-validation`[2] o algún procedimiento similar.

6. Ejemplo: Face Recognition

Como un buen ejemplo de la efectividad de una SVM vamos a hacer una aproximación al problema del reconocimiento facial. Usaremos el dataset `Labeled Faces` (de Wild) que consiste en varios miles de fotografías de personajes públicos y se puede cagar directamente desde el paquete `Scikit-Learn`

```
from sklearn.datasets import fetch_lfw_people
faces = fetch_lfw_people(min_faces_per_person=60)
print(faces.target_names)
print(faces.images.shape)


['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi' 'Tony Blair']
(1348, 62, 47)
```

Vamos a pintar algunas de las imágenes para que veamos de lo que estamos hablando realmente:

```
fig, ax = plt.subplots(3, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='bone')
    axi.set(xticks=[], yticks=[],
            xlabel=faces.target_names[faces.target[i]])
plt.show()
```

[2] Práctica 3

Cada imagen contiene [62x47] (casi 3.000 pixeles). Podríamos proceder usando cada uno de los píxeles como una característica, pero a menudo es más efectivo utilizar algún tipo de pre-procesamiento para extraer las características más importantes. Aquí vamos a usar el *PCA*, o Principal Component Analysis[3], y extraemos las 150 mejores características con las que vamos a crear nuestra máquina SVM.

```
from sklearn.svm import SVC
from sklearn.decomposition import RandomizedPCA
from sklearn.pipeline import make_pipeline

pca = RandomizedPCA(n_components=150, whiten=True, random_state=42)
svc = SVC(kernel='rbf', class_weight='balanced')
model = make_pipeline(pca, svc)
```

For the sake of testing our classifier output, we will split the data into a training and testing set:

```
from sklearn.cross_validation import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(faces.data, faces.target,
                                                random_state=42)
```

Finalmente, podemos usar una búsqueda (basada en cross-validation) para explorar la mejor combinación de parámetros. Ajustaremos C y γ para hallar el mejor modelo.s

```
from sklearn.grid_search import GridSearchCV
param_grid = {'svc__C': [1, 5, 10, 50],
              'svc__gamma': [0.0001, 0.0005, 0.001, 0.005]}
grid = GridSearchCV(model, param_grid)

%time grid.fit(Xtrain, ytrain)
print(grid.best_params_)

CPU times: user 47.8 s, sys: 4.08 s, total: 51.8 s
Wall time: 26 s
{'svc__gamma': 0.001, 'svc__C': 10}
```

Los valores óptimos recaen sobre el medio de nuestro grid; si cayesen sobre los bordes, deberíamos de extender el grid para asegurarnos que sean un mínimo verdadero.

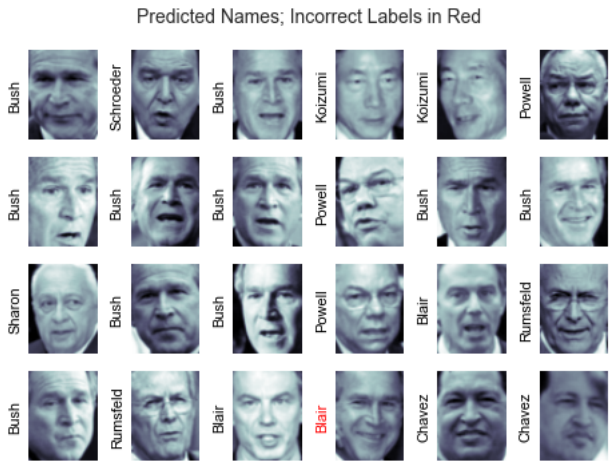
Una vez construida la máquina, podemos predecir el valor de imágenes futuras con el test-set, de imágenes nuevas:

```
model = grid.best_estimator_
yfit = model.predict(Xtest)
```

y ahora representamos un conjunto pequeño de resultados:

[3] `https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html`

```
fig, ax = plt.subplots(4, 6)
for i, axi in enumerate(ax.flat):
    axi.imshow(Xtest[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[yfit[i]].split()[-1],
                   color='black' if yfit[i] == ytest[i] else 'red')
fig.suptitle('Predicted Names; Incorrect Labels in Red', size=14);
plt.show()
```



En este pequeño ejemplo, nuestro clasificador únicamente ha cometido un error (en la cara Bush, como Blair).

Podemos tener una mejor idea de la calidad de nuestro modelo si ejecutamos el informe de clasificación:

```
from sklearn.metrics import classification_report
print(classification_report(ytest, yfit,
                            target_names=faces.target_names))
```

```
                   precision    recall  f1-score   support

     Ariel Sharon       0.65      0.73      0.69        15
     Colin Powell       0.81      0.87      0.84        68
  Donald Rumsfeld       0.75      0.87      0.81        31
    George W Bush       0.93      0.83      0.88       126
Gerhard Schroeder       0.86      0.78      0.82        23
      Hugo Chavez       0.93      0.70      0.80        20
Junichiro Koizumi       0.80      1.00      0.89        12
       Tony Blair       0.83      0.93      0.88        42

      avg / total       0.85      0.85      0.85       337
```

También deberíamos de visualizar la matriz de confusión entre estas clases:

```
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(ytest, yfit)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=faces.target_names,
            yticklabels=faces.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
plt.show()
```

y nos ayudará a tener una estimación de cómo y cuáles de las etiquetas están funcionando.

7. Fuente

Esta práctica está extraida de In-Depth: Support Vector Machines