```
# Conjunto de datos
import pandas as pd
import numpy as np
# https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.da
df = pd.read_csv('https://bit.ly/3gob0mX', header=None)
```

```
df.head(2)
```

	0	1	2	3	4	5	6	7	8	9	...	22	23	24	
0	842302	M	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.6	20
1	842517	M	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.8	19

2 rows × 32 columns

```
# Predictoras y objetivo
X = df.loc[:, 2:].values
y = df.loc[:, 1].values
```

```
# Codificación de etiquetas
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
le.classes_, le.transform(['M','B']) # clases y ejemplo
```

```
(array(['B', 'M'], dtype=object), array([1, 0]))
```

```
# Entrenamiento y pruebas
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    stratify=y, random_state=1)
X_train.shape, X_test.shape
```

```
((455, 30), (114, 30))
```

```
# Primer pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline

pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=2),
                        LogisticRegression(random_state=1))
pipe_lr.fit(X_train, y_train)
y_pred = pipe_lr.predict(X_test)
print('Exactitud en test = %.3f' % pipe_lr.score(X_test, y_test))
```

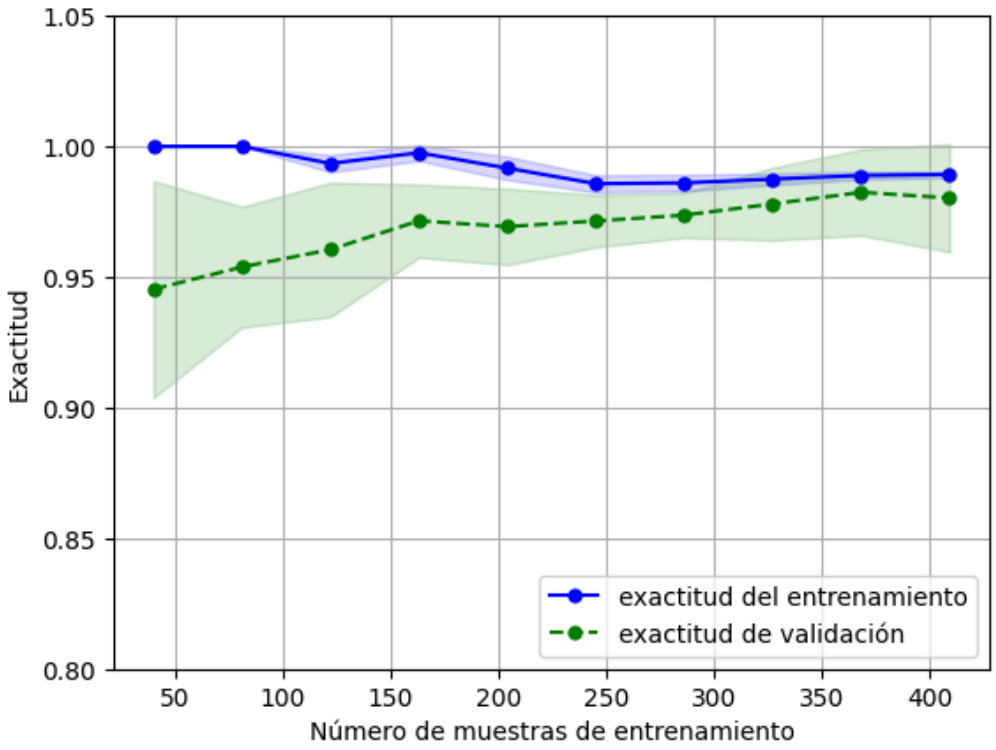
```
Exactitud en test = 0.956
```

```
# Validación cruzada en sklearn
from sklearn.model_selection import cross_val_score

scores = cross_val_score(estimator=pipe_lr,
                         X=X_train, y=y_train,
                         cv=10, n_jobs=1) # njobs=-1 => indica usar todos los núcleos dispor
print('Puntajes de exactitud de validación cruzada : %s'%(scores))
print('Exactitud de validación cruzada : %.3f +/- %.3f'%(np.mean(scores),np.std(scores)))
```

```
Puntajes de exactitud de validación cruzada : [0.93478261 0.93478261 0.95652174 0.95652174 0
 0.97777778 0.93333333 0.95555556 0.95555556]
Exactitud de validación cruzada : 0.950 +/- 0.014
```

```
# Curvas de aprendizaje
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

pipe_lr = make_pipeline(StandardScaler(),
                        LogisticRegression(penalty='l2', random_state=1))

train_sizes,train_scores,test_scores=learning_curve(estimator=pipe_lr,
                                                    X=X_train, y=y_train,
                                                    train_sizes=np.linspace(0.1,1,10),
                                                    cv=10, n_jobs=1)
```

```
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5,
         label='exactitud del entrenamiento')
plt.fill_between(train_sizes, train_mean+train_std, train_mean-train_std,
                 alpha=0.15, color='blue')

plt.plot(train_sizes, test_mean, color='green', marker='o', markersize=5,
         linestyle='--', label='exactitud de validación')
plt.fill_between(train_sizes, test_mean+test_std, test_mean-test_std,
                 alpha=0.15, color='green')

plt.grid()
plt.xlabel('Número de muestras de entrenamiento')
plt.ylabel('Exactitud')
plt.legend(loc='lower right')
plt.ylim([0.8, 1.05])
plt.show()
```



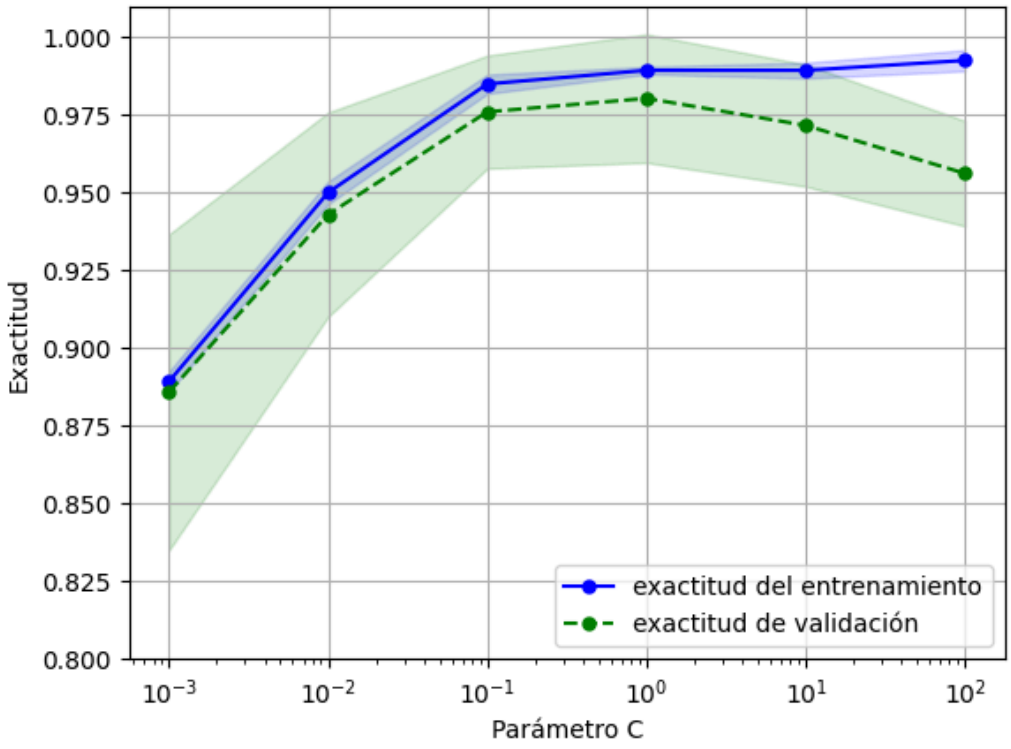

```
# Sobre/subajuste & curvas de validación
from sklearn.model_selection import validation_curve
param_range = [10**i for i in range(-3,3)]
train_scores,test_scores=validation_curve(estimator=pipe_lr,
                                          X=X_train, y=y_train,
                                          param_name='logisticregression__C',
                                          param_range=param_range,
                                          cv=10, n_jobs=10)
```

```
param_range = [10**i for i in range(-3,3)]
param_range
```

```
[0.001, 0.01, 0.1, 1, 10, 100]
```

```
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.plot(param_range, train_mean, color='blue', marker='o', markersize=5,
         label='exactitud del entrenamiento')
plt.fill_between(param_range, train_mean+train_std, train_mean-train_std,
                 alpha=0.15, color='blue')

plt.plot(param_range, test_mean, color='green', marker='o', markersize=5,
         linestyle='--', label='exactitud de validación')
plt.fill_between(param_range, test_mean+test_std, test_mean-test_std,
                 alpha=0.15, color='green')

plt.grid()
plt.xscale('log')
plt.xlabel('Parámetro C')
```

```
plt.ylabel('Exactitud')
plt.legend(loc='lower right')
plt.ylim([0.8, 1.01])
plt.show()
```




```
# Ajuste de hiperparámetros con búsqueda de malla => grid search
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

pipe_svc = make_pipeline(StandardScaler(),
                         SVC(random_state=1))

param_range = [10**i for i in range(-4,5)]
param_grid_svc = [{'svc__C':param_range,'svc__kernel':['linear']},
                  {'svc__C':param_range,'svc__gamma':param_range,
                   'svc__kernel':['rbf']}]

gs = GridSearchCV(estimator=pipe_svc, param_grid=param_grid_svc,
                  scoring='accuracy', cv=10, n_jobs=-1)

gs = gs.fit(X_train, y_train)
```

```
print(gs.best_score_)
print(gs.best_params_)
```

```
0.9846859903381642
{'svc__C': 100, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}
```

```
# exactitud del mejor estimador
clf = gs.best_estimator_
clf.fit(X_train, y_train)
print('Exactitud en test : %.3f' % clf.score(X_test, y_test))
```

```
Exactitud en test : 0.974
```

```
# Comparación de búsqueda aleatoria y de malla para estimar hiperparámetros
# https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html#
```

```
# Selección de algoritmos con validación cruzada anidada
gs_svc = GridSearchCV(estimator=pipe_svc, param_grid=param_grid_svc,
                      scoring='accuracy', cv=2)

scores = cross_val_score(gs_svc, X_train, y_train,
                         scoring='accuracy', cv=5)
```

```
print('Exactitud NCV con SVC : %.3f +/- %.3f'%(np.mean(scores),np.std(scores)))
```

```
Exactitud NCV con SVC : 0.974 +/- 0.015
```

```
from sklearn.tree import DecisionTreeClassifier

gs_dtc = GridSearchCV(estimator=DecisionTreeClassifier(random_state=0),
                  param_grid=[{'max_depth':[1,2,3,4,5,6,7,None]}],
                  scoring='accuracy', cv=2)

scores = cross_val_score(gs_dtc, X_train, y_train,
                         scoring='accuracy', cv=5)
```

```
print('Exactitud de NCV con DT : %.3f +/- %.3f'%(np.mean(scores),np.std(scores)))
```

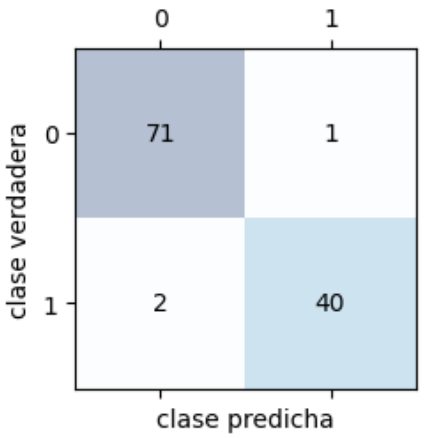
```
Exactitud de NCV con DT : 0.934 +/- 0.016
```

```
# Experimento propio: usar pipe_lr
```

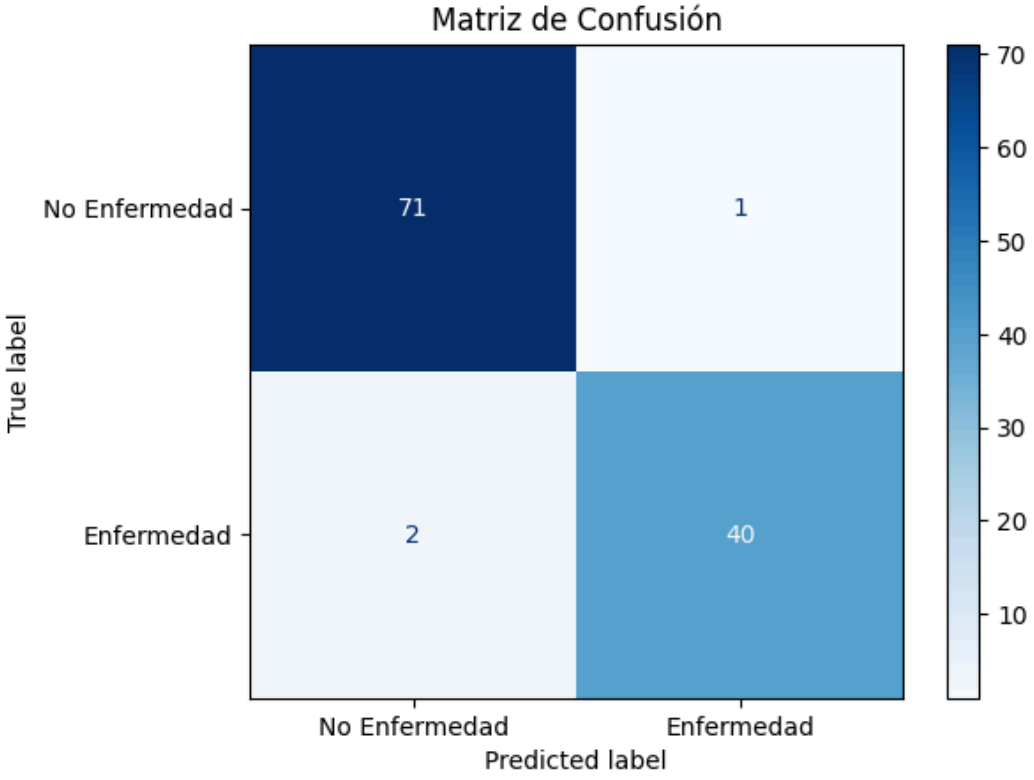
```
# Métricas de rendimiento
# Matriz de confusión
from sklearn.metrics import confusion_matrix

pipe_svc.fit(X_train, y_train)
y_pred = pipe_svc.predict(X_test)
confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(confmat)
```

```
[[71  1]
 [ 2 40]]
```

```
confmat = np.array([[71,1],[2,40]])
fig, ax = plt.subplots(figsize=(2.5, 2.5))
ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confmat.shape[0]):
  for j in range(confmat.shape[1]):
    ax.text(x=j, y=i, s=confmat[i,j],
            va='center', ha='center')
plt.xlabel('clase predicha')
plt.ylabel('clase verdadera')
plt.show()
```




```
# Con sklearn
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=confmat, display_labels=['No Enfermedad', 'Er
disp.plot(cmap=plt.cm.Blues)  # Puedes cambiar el mapa de color
plt.title('Matriz de Confusión')
plt.show()
```



```
# Precisión y sensibilidad
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score, f1_score

print('Precisión : %.3f' % precision_score(y_true=y_test, y_pred=y_pred))
print('   Recall : %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
print('       F1 : %.3f' % f1_score(y_true=y_test, y_pred=y_pred))
```

```
Precisión : 0.976
   Recall : 0.952
       F1 : 0.964
```

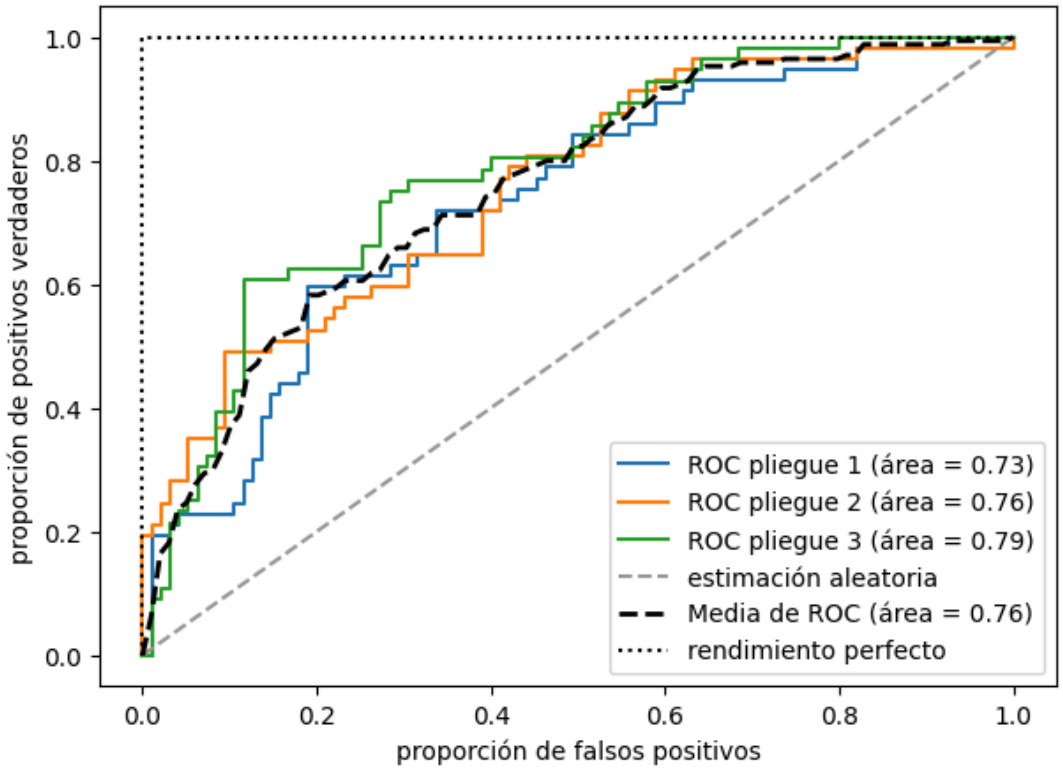
```
print(gs.best_score_)
print(gs.best_params_)
```

```
0.9846859903381642
{'svc__C': 100, 'svc__gamma': 0.001, 'svc__kernel': 'rbf'}
```

```
# Gráfica de ROC (Receiver Operating Characteristic, o Característica Operativa del Receptor
from sklearn.metrics import roc_curve, auc
from numpy import interp
from sklearn.model_selection import StratifiedKFold

pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=2),
                        LogisticRegression(penalty='l2',random_state=1,
                                           C=100.0))

X_train2 = X_train[:, [4, 14]]

cv = list(StratifiedKFold(n_splits=3).split(X_train, y_train))
```

```
fig = plt.figure(figsize=(7, 5))

mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)
all_tpr = []

for i, (train, test) in enumerate(cv):
  probas = pipe_lr.fit(X_train2[train],
                       y_train[train]).predict_proba(X_train2[test])
  fpr, tpr, threshold = roc_curve(y_train[test], probas[:,1], pos_label=1)
  mean_tpr += interp(mean_fpr, fpr, tpr)
  mean_tpr[0] = 0.0
  roc_auc = auc(fpr, tpr)
  plt.plot(fpr, tpr, label='ROC pliegue %d (área = %0.2f)' % (i+1, roc_auc))

plt.plot([0,1], [0,1], linestyle='--', color=[0.6,0.6,0.6],
         label='estimación aleatoria')

mean_tpr /= len(cv)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, 'k--',
         label='Media de ROC (área = %.2f)'%mean_auc, lw=2)
plt.plot([0,0,1], [0,1,1], linestyle=':', color='black',
```

```
         label='rendimiento perfecto')

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('proporción de falsos positivos')
plt.ylabel('proporción de positivos verdaderos')
plt.legend(loc='lower right')
plt.show()
```



1

1

1

1

1

1

1

1

1

1