Clase9 IMA539

Alejandro Ferreira Vergara

April 17, 2023

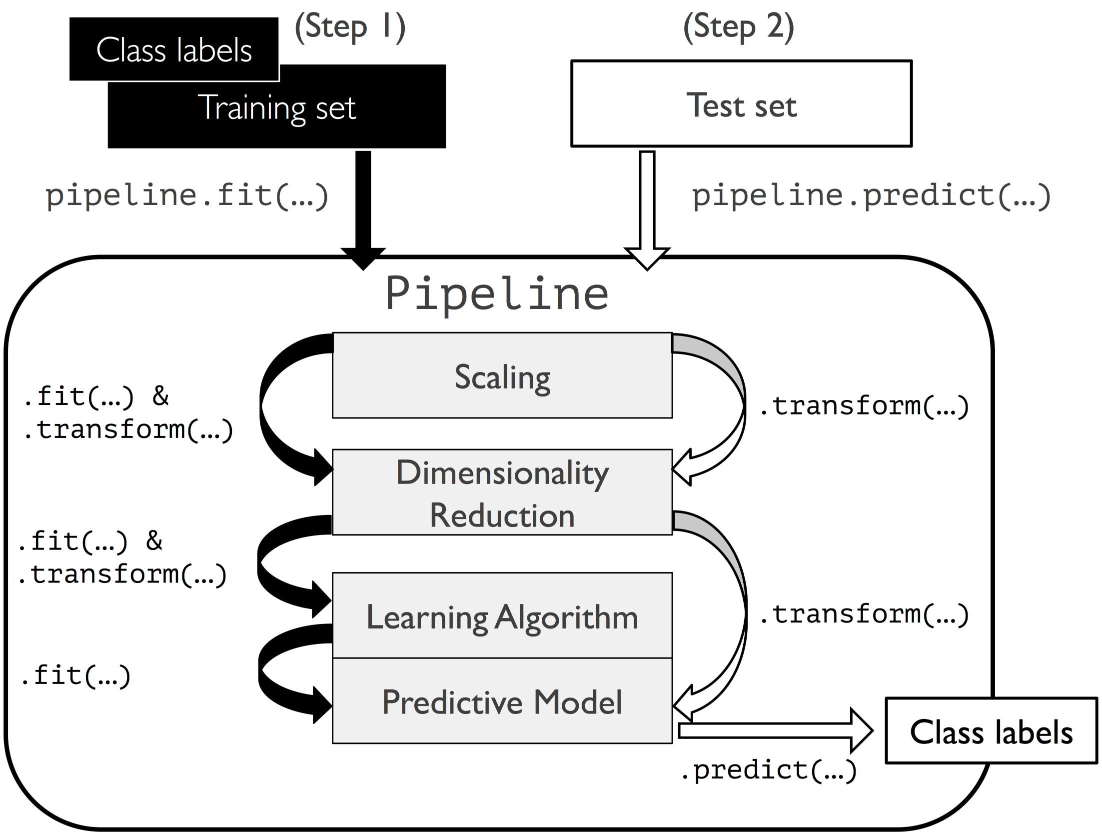
1 Métricas de Rendimiento y Ajuste de Hiperparámetros

- **Validación Cruzada**
- **Curva de Aprendizaje y Curva de Validación**
- **Métricas de Clasificación**
- **Curva ROC**
- **Datos Desbalanceados**

2 Pipeline de Scikit-Learn

[1]:
```python
from IPython.display import Image

Image(filename=r'clase9/9_1.png', width=500)
```

[1]:



[ ]:
```python
import pandas as pd

df = pd.read_csv('https://archive.ics.uci.edu/ml/'
                 'machine-learning-databases'
                 '/breast-cancer-wisconsin/wdbc.data', header=None)

df.head()
```

[ ]:
```python
df.shape
```

[ ]:
```python
df.iloc[:,1].value_counts()
```

[ ]:
```python
df.iloc[:,1:].describe(include='all')
```

[ ]:
```python
from sklearn.preprocessing import LabelEncoder

X = df.loc[:, 2:].values
y = df.loc[:, 1].values
le = LabelEncoder()
y = le.fit_transform(y)
```

```
le.classes_
```

[ ]:
```
le.transform(['M', 'B'])
```

[ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.
 ↪20,stratify=y,random_state=1)
```

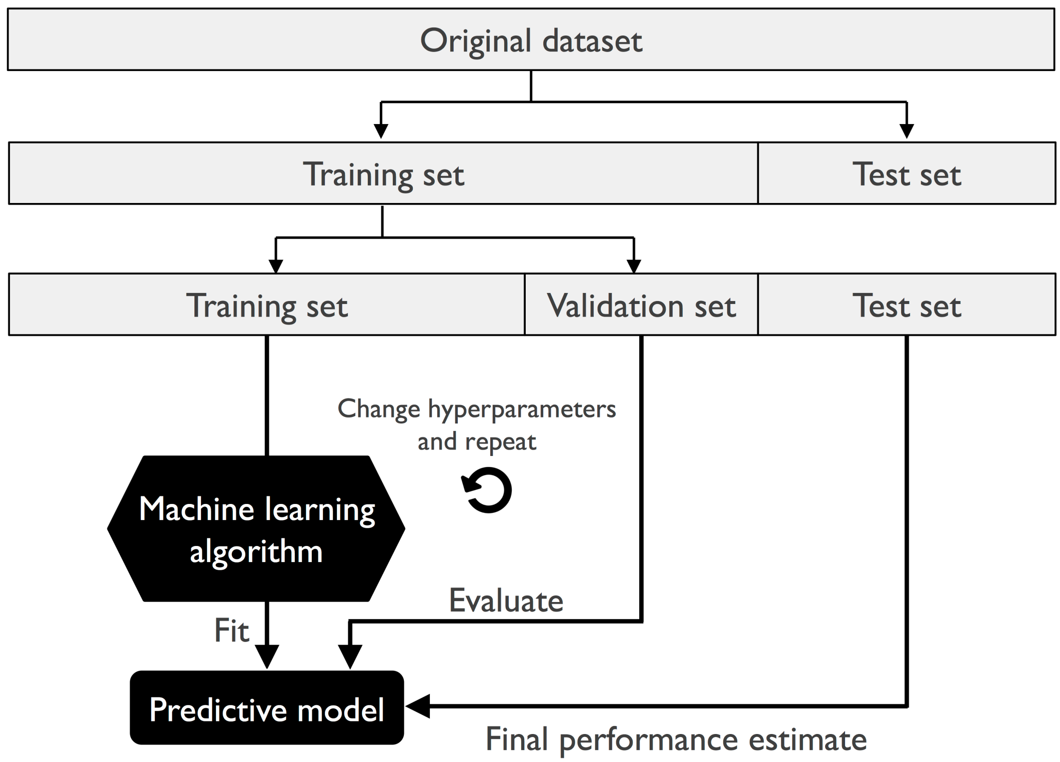
[ ]:
```
print(len(X_train))
print(len(X_test))
```

[ ]:
```
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('Test Accuracy: %.3f' % pipe_lr.score(X_test, y_test))
```

3 Cross Validation

[2]:
```
Image(filename=r'clase9/9_2.png', width=500)
```

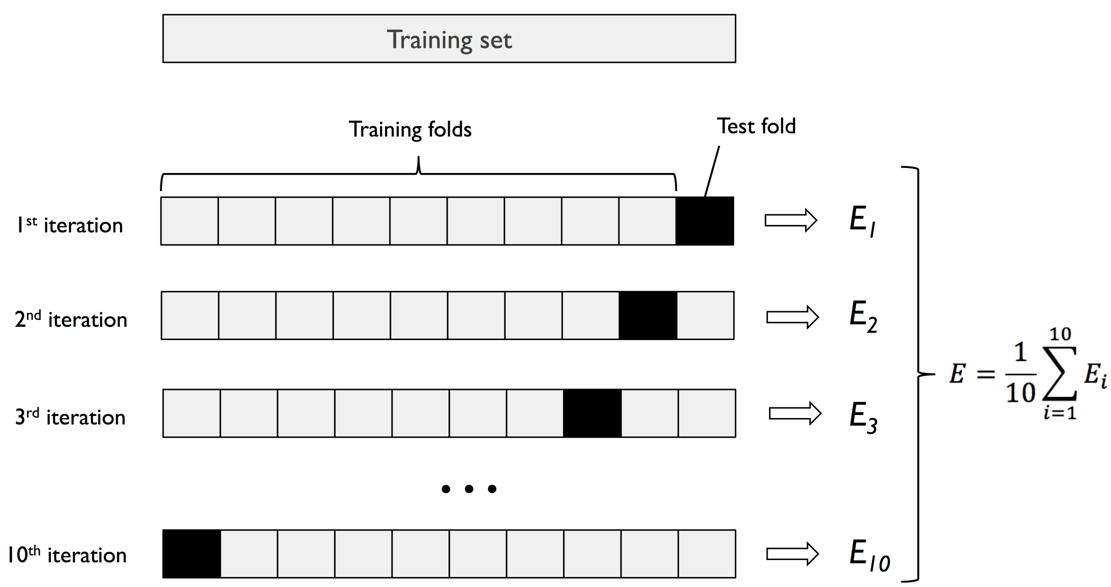
[2]:



3.1 *K*-Fold Cross-Validation

[3]:
```
Image(filename=r'clase9/9_3.png', width=600)
```

[3]:



```
[ ]: import numpy as np
from sklearn.model_selection import StratifiedKFold


kfold = StratifiedKFold(n_splits=10).split(X_train, y_train)

scores = []
for k, (train, test) in enumerate(kfold):
    pipe_lr = make_pipeline(StandardScaler(),PCA(n_components=2),LogisticRegression(random_state=1))
    pipe_lr.fit(X_train[train], y_train[train])
    score = pipe_lr.score(X_train[test], y_train[test])
    scores.append(score)
    print('Fold: %2d, Class dist train.: %s, Class dist test.: %s, Acc: %.3f' % (k+1,np.bincount(y_train[train]), np.bincount(y_train[test]), score))

print('\nCV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

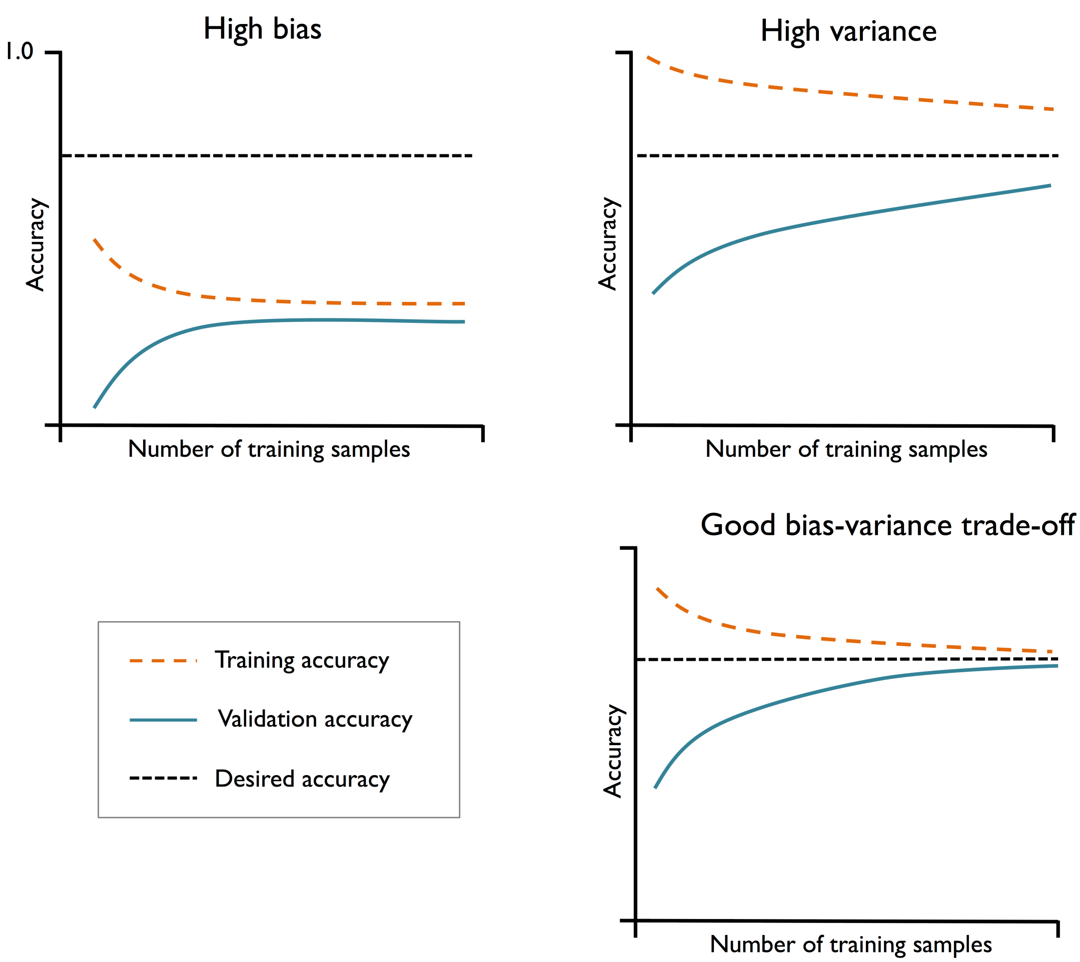
```
[ ]: from sklearn.model_selection import cross_val_score

scores = cross_val_score(estimator=pipe_lr,X=X_train,y=y_train,cv=10,n_jobs=-1)

print('CV accuracy scores: %s' % scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

OjO con el conjunto Test…

4 Overfitting y Underfitting

```
[4]: Image(filename=r'clase9/9_4.png', width=550)
```

[4]:



5 Curva 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, 1.0, 10),cv=10,n_jobs=2)

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='training␣
 ↪accuracy')

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',linestyle='--',
         marker='s', markersize=5,label='validation accuracy')

plt.fill_between(train_sizes,test_mean + test_std,test_mean - test_std,alpha=0.
 ↪15,color='green')

plt.grid()
plt.xlabel('Number of training samples')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.ylim([0.8, 1.03])
plt.tight_layout()
#plt.savefig('images/06_05.png', dpi=300)
plt.show()
```

6 Curva de Validación

[ ]:
```
from sklearn.model_selection import validation_curve

param_range = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]

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)

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='training␣
 ↪accuracy')
```

```
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',linestyle='--',
         marker='s',markersize=5,label='validation accuracy')

plt.fill_between(param_range,test_mean + test_std,test_mean - test_std,alpha=0.
 ↪15,color='green')

plt.grid()
plt.xscale('log')
plt.legend(loc='lower right')
plt.xlabel('Parameter C')
plt.ylabel('Accuracy')
plt.ylim([0.8, 1.0])
plt.tight_layout()
# plt.savefig('images/06_06.png', dpi=300)
plt.show()
```

7 Grid Search

[ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

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

param_range = [0.0001*0.5,0.0001,0.001*0.5,0.001,0.01*0.5,0.01,0.1*0.5,0.1,
               1.*0.5,1.,10.*0.5,10.,100.*0.5,100.,1000.*0.5,1000.]

param_grid = [{'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,scoring='accuracy',cv=10,n_jobs=1)
gs = gs.fit(X_train, y_train)

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

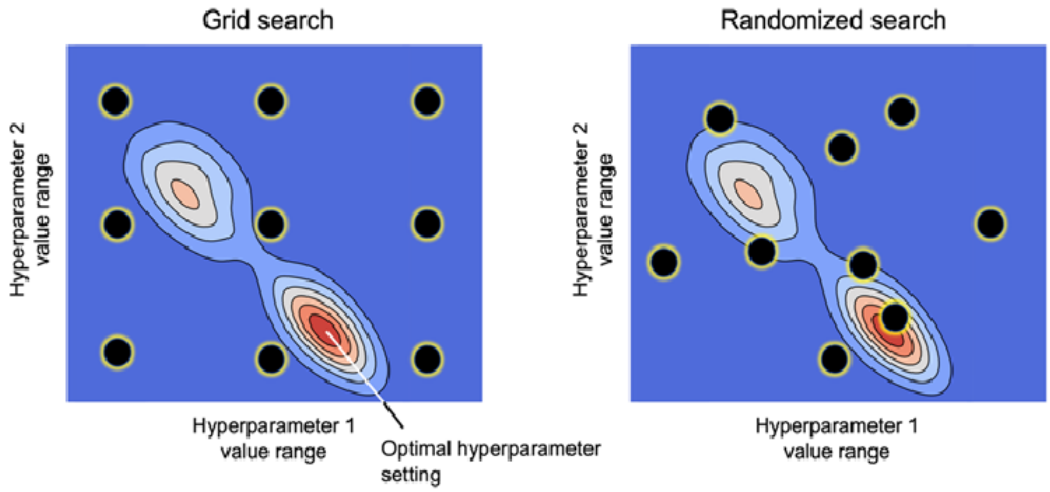
[ ]:
```
clf = gs.best_estimator_

print('Test accuracy: %.3f' % clf.score(X_test, y_test))
```

OjO con (X_test,y_test)

7.1 Búsqueda aleatoria

[5]:
```
Image(filename=r'clase9/9_4_5.png', width=600)
```

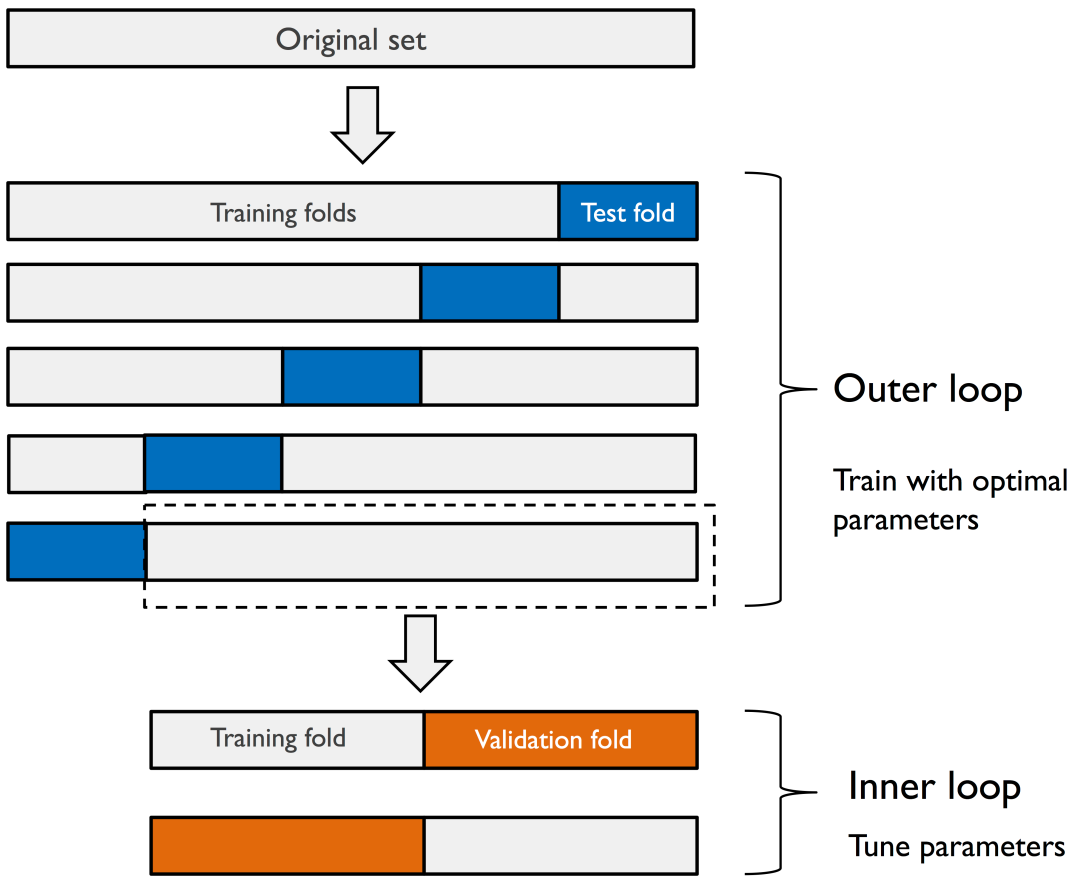
[5]:



7.2 Combinando K-fold CV con Grid Search

[6]:
```
Image(filename=r'clase9/9_5.png', width=550)
```

[6]:




```
[ ]: gs = GridSearchCV(estimator=pipe_svc,param_grid=param_grid,scoring='accuracy',cv=10,n_jobs=-1)

scores = cross_val_score(gs, X_train, y_train, scoring='accuracy',cv=10,n_jobs=-1)

print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

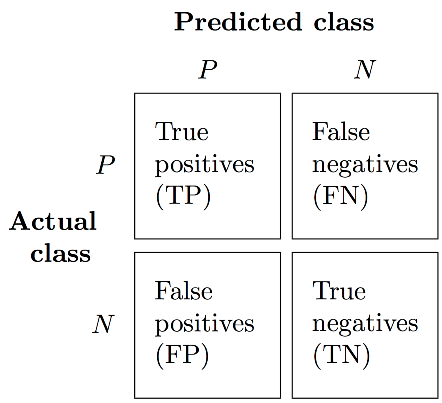
```
[ ]: from sklearn.tree import DecisionTreeClassifier

gs = GridSearchCV(estimator=DecisionTreeClassifier(random_state=0),
                  param_grid=[{'max_depth': [1, 2, 3, 4, 5, 6, 7, None]}],
                  scoring='accuracy',cv=10)

scores = cross_val_score(gs, X_train, y_train,scoring='accuracy', cv=10)

print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

8 Métricas de Rendimiento

8.1 Matriz de Confusión

[7]:
```
Image(filename=r'clase9/9_6.png', width=400)
```

[7]:



[ ]:
```
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)
```

[ ]:
```
fig, ax = plt.subplots(figsize=(2.5, 2.5),dpi=150)
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('Etiqueta Predicha')
plt.ylabel('Etiqueta Real')

plt.tight_layout()
#plt.savefig('images/06_09.png', dpi=300)
plt.show()
```

8.2 Métricas para evaluar Modelos de Clasificación

Error:

$$ERR = \frac{FP + FN}{FP + FN + TP + TN} \quad (1)$$

Exactitud:

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR \quad (2)$$

Tasa de Verdaderos Positivos:

$$TPR = \frac{TP}{P} = \frac{TP}{FN + TP} \quad (3)$$

Tasa de Falsos Positivos:

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} \quad (4)$$

Precisión:

$$PRE = \frac{TP}{TP + FP} \quad (5)$$

Recall:

$$REC = TPR = \frac{TP}{FN + TP} \quad (6)$$

A continuación definiremos:

$$AP = TP + FN, AN = FP + TN \quad (7)$$

$$PP = TP + FP, PN = FN + TN \quad (8)$$

F1-score:

$$F1 = \frac{2 \times TP}{AP + PP} \quad (9)$$

Coeficiente de Correlación de Matthews

$$MCC = \frac{(TP \times TN - FP \times FN)}{\sqrt{PP \times PN \times AP \times AN}} \quad (10)$$

Coeficiente de Kappa de Cohen

$$K = \frac{P_o - P_e}{1 - P_e} \quad (11)$$

donde,

$$P_e = P_{pos} + P_{neg} \quad (12)$$

$$P_{pos} = \frac{AP}{N} \times \frac{PP}{N} \quad (13)$$

$$P_{neg} = \frac{AN}{N} \times \frac{PN}{N} \quad (14)$$

con N el total de datos presentados en la matriz de confusión.

[ ]:
```
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
from sklearn.metrics import matthews_corrcoef,cohen_kappa_score

print('Accuracy (ec. 2): %.3f' % accuracy_score(y_true=y_test, y_pred=y_pred))
print('Precision (ec. 5): %.3f' % precision_score(y_true=y_test, y_pred=y_pred))
print('Recall (ec. 6): %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
print('F1 (ec. 9): %.3f' % f1_score(y_true=y_test, y_pred=y_pred))
print('MCC (ec. 10): %.3f' % matthews_corrcoef(y_true=y_test, y_pred=y_pred))
```

```
print('Kappa (ec. 11): %.3f' % cohen_kappa_score(y1=y_test, y2=y_pred))
```

8.3 Curva ROC

```
[ ]: from sklearn.metrics import roc_curve, auc
from scipy import interp

pipe_lr =␣
 ↪make_pipeline(StandardScaler(),PCA(n_components=2),LogisticRegression(penalty='l2',random_s
 ↪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, thresholds = 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 fold %d (area = %0.2f)'% (i+1, roc_auc))

plt.plot([0, 1],[0, 1],linestyle='--',color=(0.6,0.6,0.6),label='random␣
 ↪guessing')

mean_tpr /= len(cv)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, 'k--',label='mean ROC (area = %0.2f)' %␣
 ↪mean_auc,lw=2)
plt.plot([0, 0, 1],[0, 1, 1],linestyle=':',color='black',label='perfect␣
 ↪performance')

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.legend(loc="lower right")
```

```
plt.tight_layout()
# plt.savefig('images/06_10.png', dpi=300)
plt.show()
```

9 Tratar con Datos Desbalanceados

[ ]:
```
X_imb = np.vstack((X[y == 0], X[y == 1][:40]))
y_imb = np.hstack((y[y == 0], y[y == 1][:40]))
```

[ ]:
```
y_pred = np.zeros(y_imb.shape[0])
np.mean(y_pred == y_imb) * 100
```

[ ]:
```
from sklearn.utils import resample

print('Número de muestras de clase 1 antes:', X_imb[y_imb == 1].shape[0])

X_upsampled, y_upsampled = resample(X_imb[y_imb == 1],y_imb[y_imb ==␣
 ↪1],replace=True,
                                    n_samples=X_imb[y_imb == 0].
 ↪shape[0],random_state=123)

print('Número de muestras de clase 1 después:', X_upsampled.shape[0])
```

[ ]:
```
X_bal = np.vstack((X[y == 0], X_upsampled))
y_bal = np.hstack((y[y == 0], y_upsampled))
```

[ ]:
```
y_pred = np.zeros(y_bal.shape[0])
np.mean(y_pred == y_bal) * 100
```

[ ]:
```

```