Métricas de Rendimiento y Ajuste de Hiperparámetros

¿Qué aprenderemos hoy?

- **Validación Cruzada**
- **Curva de Aprendizaje y Curva de Validación**
- **Búsqueda de Hyperparámetros**

Pipeline de Scikit-Learn

In [ ]:
```
from IPython.display import Image

Image(filename=r'Imagenes_Clase_10/9_1.png', width=500)
```

In [ ]:
```
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()
```

In [ ]:
```
df.shape
```

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

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

In [60]:
```
from sklearn.preprocessing import LabelEncoder

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

In [ ]:
```
y[::10]
```

In [62]:
```
le = LabelEncoder()
y = le.fit_transform(y)
```

In [ ]:
```
y[::10]
```

In [ ]:
```
le.classes_
```

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

In [66]:
```
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)
```

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

In [ ]:
```
from IPython.display import Image

Image(filename=r'Imagenes_Clase_10/9_1.png', width=500)
```

In [ ]:
```
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))
```

In [ ]:
```
pipe_lr2 = make_pipeline(StandardScaler(),PCA(n_components=5),
                         LogisticRegression(random_state=1))

pipe_lr2.fit(X_train, y_train)

y_pred = pipe_lr2.predict(X_test)
print('Test Accuracy pipe_lr2: %.3f' % pipe_lr2.score(X_test, y_test))
```

- Documentación

Cross Validation

In [ ]:
```
Image(filename=r'Imagenes_Clase_10/9_2.png', width=500)
```

K-Fold Cross-Validation

In [ ]:
```
Image(filename=r'Imagenes_Clase_10/9_3.png', width=600)
```

In [ ]:
```
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)))
```

- Documentación

In [ ]:
```
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)))
```

- Documentación

Observación: Siempre dejar de lado completamente hasta el final del proceso al conjunto Test...

Overfitting y Underfitting

In [ ]:
```
Image(filename=r'Imagenes_Clase_10/9_4.png', width=550)
```

Curva de Aprendizaje

In [ ]:
```
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, 20),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()
```

Curva de Validación

In [ ]:
```
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()
```

Grid Search

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

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

param_range = [0.0001,0.01,1.,100.]

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=5,n_jobs=-1,verbose=1)
gs = gs.fit(X_train, y_train)

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

In [ ]:
```
clf = gs.best_estimator_
print('Test accuracy: %.3f' % clf.score(X_test, y_test))
```

Combinando K-fold CV con Grid Search

In [ ]:
```
Image(filename=r'Imagenes_Clase_10/9_5.png', width=550)
```

In [ ]:
```python
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)))
```

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

gs = GridSearchCV(estimator=DecisionTreeClassifier(random_state=0),
                  param_grid=[{'max_depth': [2, 4, 6, 8]}],
                  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)))
```

Búsqueda aleatoria de Hiperparámetros

In [ ]:
```python
Image(filename=r'Imagenes_Clase_10/9_4_5.png', width=600)
```

In [ ]:
```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform

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

# PARAMETROS COMO LISTA

param_range = [0.0001,0.01,1.,100.]
param_grid = [{'svc__C': param_range,'svc__kernel': ['linear']},
              {'svc__C': param_range,'svc__gamma': param_range,'svc__kernel': ['rbf']}]

RSCV = RandomizedSearchCV(pipe_svc, param_grid, n_iter=5,random_state=0,verbose=1)
RSCV = RSCV.fit(X_train, y_train)

print(RSCV.best_score_)
print(RSCV.best_params_)
#search.best_params_
```

In [ ]:
```python
pipe_svc = make_pipeline(StandardScaler(),SVC(random_state=1))

# PARAMETROS COMO DISTRIBUCION

distribuciones = [{'svc__C': uniform(loc=1e-6, scale=100),'svc__kernel': ['linear']},
              {'svc__C': uniform(loc=1e-6, scale=100),'svc__gamma': uniform(loc=0, scale=10000),
               'svc__kernel': ['rbf']}]

RSCV = RandomizedSearchCV(pipe_svc, distribuciones, n_iter=5,random_state=1,verbose=1)
RSCV = RSCV.fit(X_train, y_train)

print(RSCV.best_score_)
print(RSCV.best_params_)
#search.best_params_
```

`conda install scikit-optimize`

In [ ]:
```python
import skopt
print('skopt %s' % skopt.__version__)
```

In [ ]:
```python
from skopt import BayesSearchCV

parametros = dict()
parametros['svc__C'] = (1e-6, 100.0, 'uniform')
parametros['svc__gamma'] = (1e-6, 100.0, 'uniform')
parametros['svc__degree'] = (1,5)
parametros['svc__kernel'] = ['linear', 'poly', 'rbf', 'sigmoid']
#parametros['svc__kernel'] = ['linear', 'rbf']

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

search = BayesSearchCV(estimator=pipe_svc, search_spaces=parametros, n_jobs=-1, cv=5,n_iter=25,verbose=0)
# perform the search
search.fit(X, y)
# report the best result
print(search.best_score_)
print(search.best_params_)
```

`conda install -c conda-forge optuna`

In [43]:
```python
from sklearn.svm import SVC
```

```python
from sklearn.preprocessing import StandardScaler

import optuna
from sklearn.metrics import accuracy_score


scaler=StandardScaler()
scaler=scaler.fit(X_train)
X_train_scaled=scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

In [ ]:
```python
def model_performance(model, X=X_test_scaled, y=y_test):
    y_pred = model.predict(X)
    return round(accuracy_score(y_pred, y),5)

SVC_model=SVC(random_state=1)
SVC_model.fit(X_train_scaled, y_train)

print("Validation accuracy: ", model_performance(SVC_model))
```

In [ ]:
```python
def create_model(trial):
    model_type = trial.suggest_categorical('model_type', ['svc'])
    kernel = trial.suggest_categorical('kernel', ['linear', 'rbf'])
    regularization = trial.suggest_float('svm-regularization', 0.0001, 100)
    gamma = trial.suggest_float('gamma', 0.0001, 100)
    model = SVC(kernel=kernel, C=regularization,gamma=gamma,random_state=0)

    if trial.should_prune():
            raise optuna.TrialPruned()

    return model

def objective(trial):
    model = create_model(trial)
    model.fit(X_train_scaled, y_train)
    return model_performance(model)

study = optuna.create_study(direction='maximize')
study.optimize(objective , n_trials =25)
```