PIPELINES

MACHINE LEARNING

VICTOR CABALLERO JESÚS GONZÁLEZ VIRIDIANA ESPINOSA

SUPERVISADO / NO SUPERVISADO

Supervisado

Data Set: Credit_npo

Librerías

```
1]
```

```
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline
```

Target - Características

Carga el conjunto de datos

```python
# Cargar el conjunto de datos de crédito
file_path_credit_npo = './data/credit_npo.csv'
credit_data = pd.read_csv(file_path_credit_npo)

# Revisar las primeras filas del conjunto de datos
credit_data.head()
```

[72] Python

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	NumberOfTimes90DaysLate	NumberRealEsta
0	0	0.081892	37	0	0.070709	5656.0	12	1	
1	0	0.023413	74	0	0.209197	4870.0	9	0	
2	0	0.000000	43	0	0.080784	5000.0	2	0	
3	0	0.492754	44	0	0.412735	7333.0	4	0	
4	0	1.000000	63	0	0.000000	8333.0	3	0	

Definición de la variable objetivo y las características

```python
# Definir la variable objetivo
y = credit_data['SeriousDlqin2yrs']

# Seleccionar las características (features) para el modelo
X = credit_data.drop(columns=['SeriousDlqin2yrs'])
```

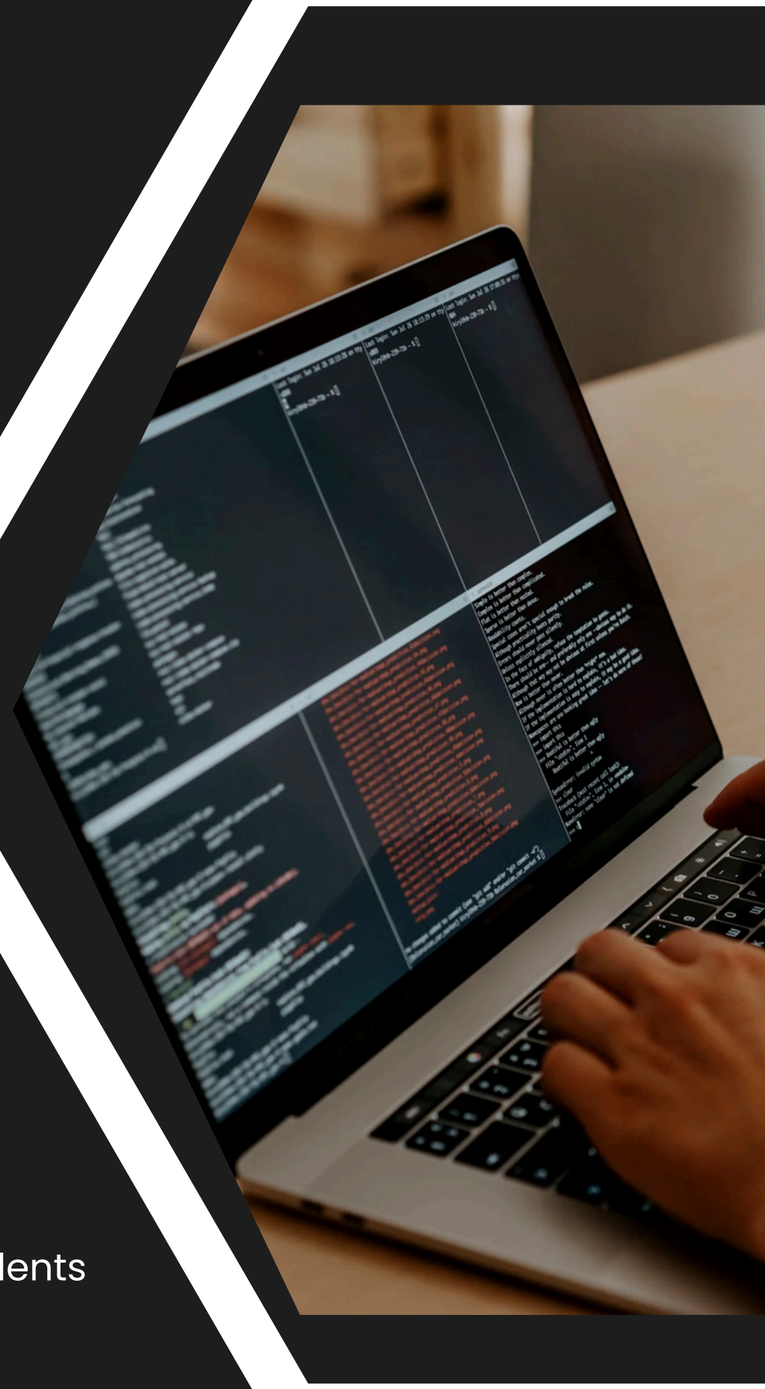
[73] Python

```python
# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[74] Python

Procesamiento de datos

```python
# Redefinir la configuración del preprocesamiento
# Imputar valores faltantes y escalar las características numéricas
num_pipeline = Pipeline(
    steps=[
        ("imputer", SimpleImputer(strategy="mean")),  # Imputar valores faltantes
        ("scaler", StandardScaler())  # Escalar las características
    ]
)

# Definir el transformador de columnas
preprocessing = ColumnTransformer(
    transformers=[
        ("num", num_pipeline, [
            "RevolvingUtilizationOfUnsecuredLines", "age",
            "NumberOfTime30-59DaysPastDueNotWorse",
            "DebtRatio", "MonthlyIncome",
            "NumberOfOpenCreditLinesAndLoans",
            "NumberOfTimes90DaysLate",
            "NumberRealEstateLoansOrLines",
            "NumberOfTime60-89DaysPastDueNotWorse",
            "NumberOfDependents"
        ]),  # Características numéricas
    ],
    remainder="drop"  # Descarta las columnas que no se procesan
)
```



Missings:

- MonthlyIncome
- NumberOfDependents

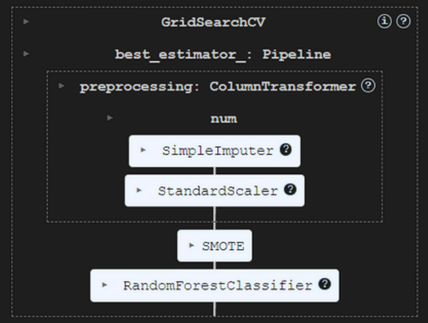
Definir el modelo y los parámetros para GridSearch

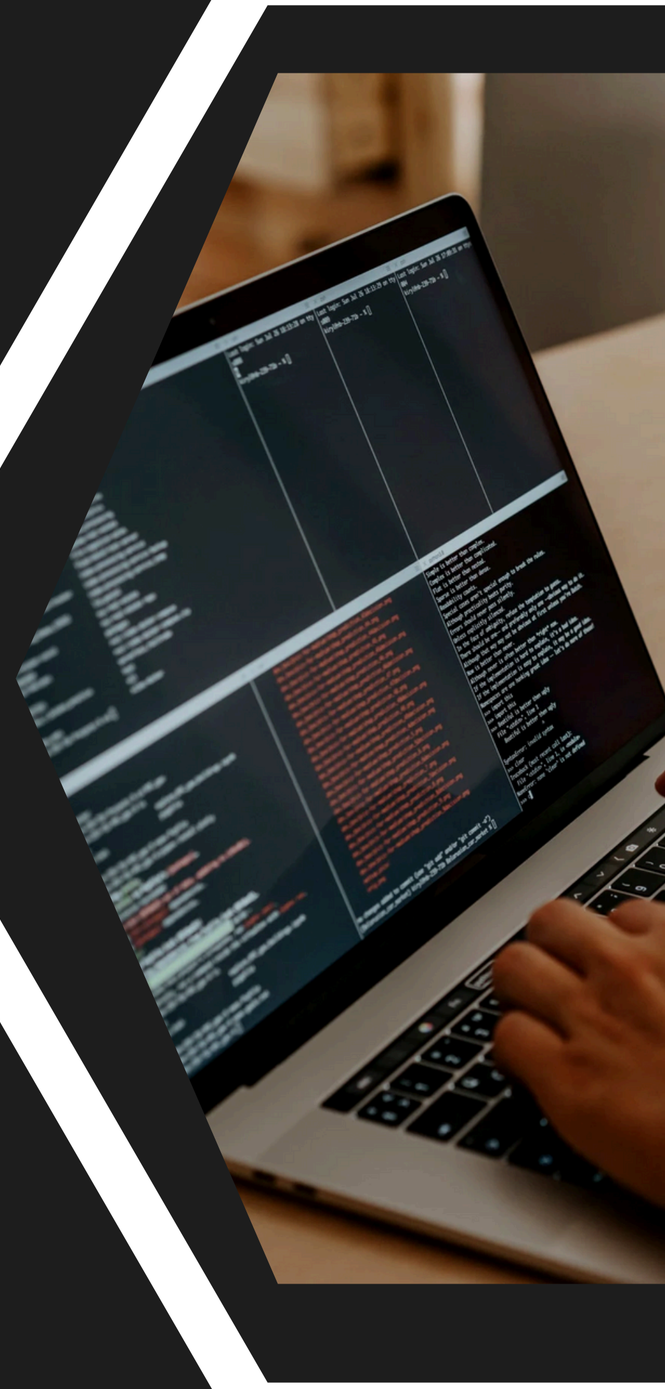
```python
# Definir el modelo
model = RandomForestClassifier(random_state=42)

# Crear la tubería con SMOTE, el preprocesamiento y el modelo
pipeline = ImbPipeline(steps=[
    ('preprocessing', preprocessing),  # Primero, preprocesar los datos
    ('smote', SMOTE(random_state=42)),  # Aplicar SMOTE
    ('model', model)  # Modelo de RandomForest
])

# Definir los parámetros a buscar en GridSearch
param_grid = {
    'model__n_estimators': [10, 50, 100, 200],  # Probar con más o menos árboles
    'model__max_features': ['auto', 'sqrt'],  # Considerar diferentes divisiones
    'model__max_depth': [5, 10, 20, None],  # Probar más niveles
    'model__min_samples_split': [2, 5, 10, 20],  # Probar diferentes cortes
    'model__min_samples_leaf': [1, 2, 4, 6],  # Probar diferentes tamaños de hojas
    'model__class_weight': ['balanced', None]  # Ajustar pesos de clase
}
```

```python
# Ajustar GridSearchCV con validación cruzada
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='f1', verbose=2, n_jobs=-1)
grid_search.fit(X_train, y_train)  # Ajustar el modelo a los datos de entrenamiento
```

Python

```
Fitting 5 folds for each of 1024 candidates, totalling 5120 fits
[CV] END model__class_weight=balanced, model__max_depth=5, model__max_features=auto, model__min_samples_leaf=1, model__min_samples_split=2, model__n_estimators=10; total time=   0.0s
[CV] END model__class_weight=balanced, model__max_depth=5, model__max_features=auto, model__min_samples_leaf=1, model__min_samples_split=2, model__n_estimators=10; total time=   0.0s
[CV] END model__class_weight=balanced, model__max_depth=5, model__max_features=auto, model__min_samples_leaf=1, model__min_samples_split=2, model__n_estimators=10; total time=   0.0s
    warnings.warn(some_fits_failed_message, FitFailedWarning)
  /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/numpy/ma/core.py:2820: RuntimeWarning: invalid value encountered in cast
    _data = np.array(data, dtype=dtype, copy=copy,
  /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/model_selection/_search.py:1102: UserWarning: One or more of the test scores are non-finite: [
  nan        nan        nan ... 0.42615475 0.42749987 0.4217664 ]
    warnings.warn(
```





Mejores parámetros

```python
# Imprimir los mejores parámetros encontrados
print("Mejores parámetros:", grid_search.best_params_)
print("\nMejor puntuación de validación:", grid_search.best_score_)

```

Python

```
Mejores parámetros: {'model__class_weight': 'balanced', 'model__max_depth': 20, 'model__max_features': 'sqrt', 'model__min_samples_leaf': 6, 'model__min_samples_split': 20, 'model__n_estimators': 100}

Mejor puntuación de validación: 0.4313378155096807 6
```

```python
# Hacer predicciones sobre el conjunto de prueba utilizando los mejores parámetros
y_pred = grid_search.predict(X_test)

# Evaluar el rendimiento del modelo
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Imprimir la matriz de confusión y el informe de clasificación
print("Matriz de Confusión:\n", conf_matrix)
print("\nInforme de Clasificación:\n", class_report)
```

[79]

```
Matriz de Confusión:
 [[2208  137]
 [  89   74]]

Informe de Clasificación:
              precision    recall  f1-score   support

           0       0.96      0.94      0.95      2345
           1       0.35      0.45      0.40       163

    accuracy                           0.91      2508
   macro avg       0.66      0.70      0.67      2508
weighted avg       0.92      0.91      0.92      2508
```

No Supervisados

No Supervisado

Data Set: Bebidas energéticas

Librerías

```python
# Importación de bibliotecas
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline, TransformerMixin
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, train_test_split, KFold
from sklearn.base import BaseEstimator, ClusterMixin


# Configuración de estilo para gráficos
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

```

Carga y exploración de los datos

01 Carga del archivo CV

```
# Cargar el archivo CSV y dividir en columnas
data = pd.read_csv('./data/empowering_drinks.csv',sep='|')
data.head(5)
```

	Azúcares	Vitamínas del grupo B	Cafeína	Ácido Cítrico	Taurina
0	1.518613	0.232053	1.034819	1.013009	0.251717
1	0.246290	-0.827996	0.733629	0.965242	-0.293321
2	0.196879	1.109334	1.215533	1.395148	0.269020
3	1.691550	0.487926	1.466525	2.334574	1.186068
4	0.295700	1.840403	0.663351	-0.037874	-0.319276

02 Estimador KMeans con validación cruzada

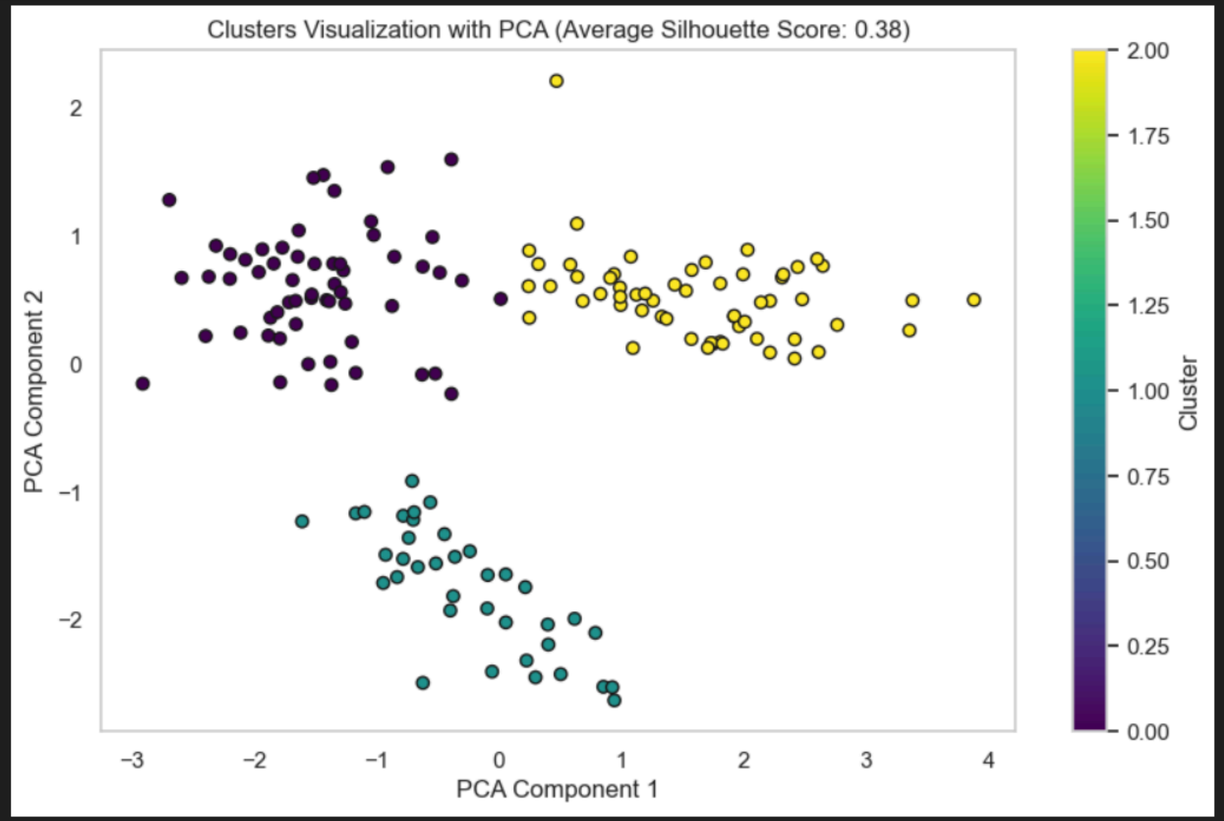
```
# Definición de un estimador para KMeans con validación cruzada
class KMeansWithSilhouette(BaseEstimator, ClusterMixin):
    def __init__(self, n_clusters=3):
        self.n_clusters = n_clusters
        self.kmeans = KMeans(n_clusters=self.n_clusters, random_state=42)

    def fit(self, X, y=None):
        self.kmeans.fit(X)
        return self

    def predict(self, X):
        return self.kmeans.predict(X)

    def score(self, X, y=None):  # Aceptar 'y' como argumento adicional
        labels = self.kmeans.predict(X)
        return silhouette_score(X, labels)

```

03 Preprocesamiento y Clustering

```
# Preprocesamiento y Clustering
X = data.values  # Convertir a numpy array para procesar

# Crear el pipeline para clustering
pipeline_clustering = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('kmeans', KMeansWithSilhouette(n_clusters=3))  # Usar KMeans con validación cruzada
])

# Validación cruzada para el modelo KMeans
kf = KFold(n_splits=5, shuffle=True, random_state=42)
silhouette_scores = []

for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    pipeline_clustering.fit(X_train)  # Ajustar solo en el conjunto de entrenamiento
    score = pipeline_clustering.score(X_test)  # Calcular el silhouette score
    silhouette_scores.append(score)

average_silhouette_score = np.mean(silhouette_scores)

```

Predicción Clusters / Etiquetas / PCA visualización Clusters

```python
# Preprocesamiento y Clustering
X = data.values  # Convertir a numpy array para procesar

# Crear el pipeline para clustering
pipeline_clustering = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('kmeans', KMeansWithSilhouette(n_clusters=3))  # Usar KMeans con validación cruzada
])

# Validación cruzada para el modelo KMeans
kf = KFold(n_splits=5, shuffle=True, random_state=42)
silhouette_scores = []

for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    pipeline_clustering.fit(X_train)  # Ajustar solo en el conjunto de entrenamiento
    score = pipeline_clustering.score(X_test)  # Calcular el silhouette score
    silhouette_scores.append(score)

average_silhouette_score = np.mean(silhouette_scores)

```



Preparación de datos

```python
# Preparar datos para el modelo predictivo
X_features = data.drop(columns=['Cluster'])
y_labels = data['Cluster']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_features, y_labels, test_size=0.2, random_state=42)
```

```python
# Crear el pipeline con transformaciones y modelo
pipeline_model = Pipeline(steps=[
    ('polynomial_features', PolynomialFeatures(degree=2, include_bias=False)),
    ('scaler', StandardScaler()),
    ('logistic_regression', LogisticRegression(max_iter=1000))
])

# Configurar GridSearchCV para ajuste de hiperparámetros
param_grid = {
    'logistic_regression__C': [0.01, 0.1, 1, 10, 100],
    'logistic_regression__solver': ['liblinear', 'saga']
}
grid_search = GridSearchCV(pipeline_model, param_grid, cv=5, scoring='accuracy')

# Ajustar el modelo a los datos de entrenamiento
grid_search.fit(X_train, y_train)

# Mejor parámetros y puntuación
best_params = grid_search.best_params_
best_score = grid_search.best_score_
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/linear_model/_sag.py:349: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/linear_model/_sag.py:349: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/linear_model/_sag.py:349: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/linear_model/_sag.py:349: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/linear_model/_sag.py:349: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
```

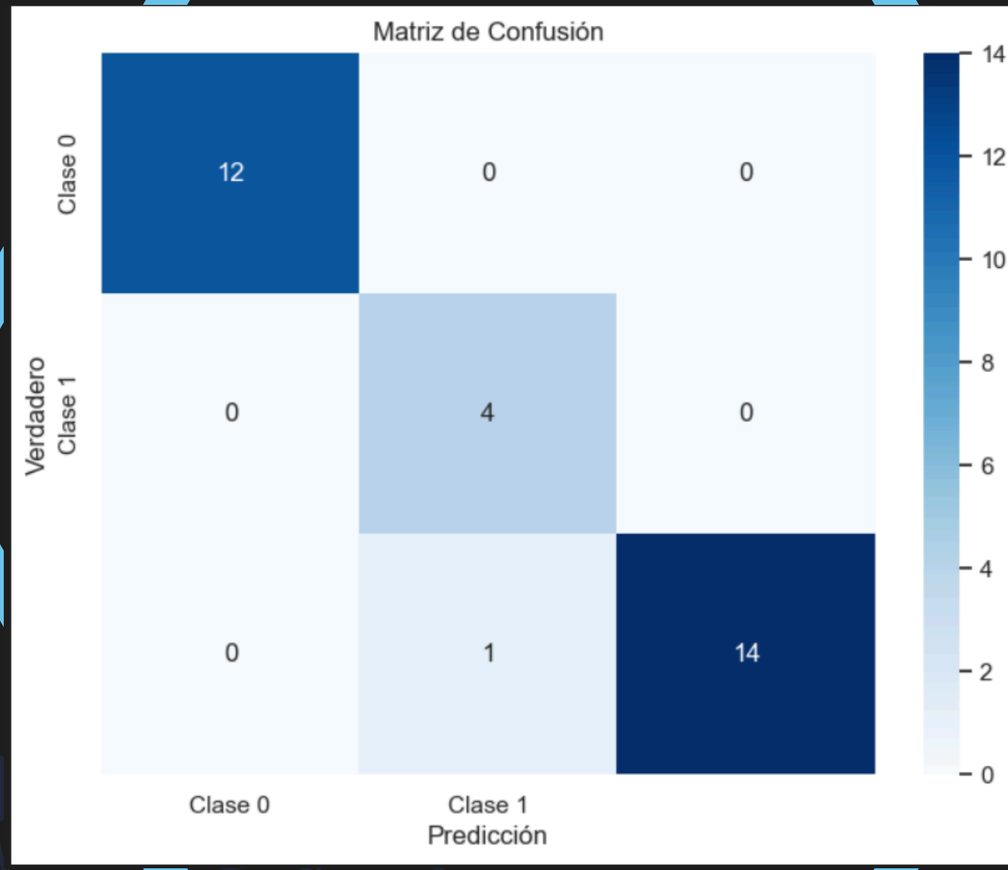
Evaluación Modelo y Matriz de Confusión

```python
# Evaluar el modelo en el conjunto de prueba
y_pred = grid_search.predict(X_test)

# Reporte de clasificación
report = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

best_params, best_score, report, conf_matrix, average_silhouette_score

```

Python

```
({'logistic_regression__C': 10, 'logistic_regression__solver': 'liblinear'},
 1.0,
 '              precision    recall  f1-score   support\n\n           0       1.00      1.00      1.00        12\n           1       0.80      1.00      0.89         4\n           2
 array([[12,  0,  0],
        [ 0,  4,  0],
        [ 0,  1, 14]]),
 0.38315948043168746)
```

```python
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Clase 0', 'Clase 1'], yticklabels=['Clase 0', 'Clase 1'])
plt.ylabel('Verdadero')
plt.xlabel('Predicción')
plt.title('Matriz de Confusión')
plt.show()
```



¡Gracias!