Explicación de Caso Práctico “Módulo IV: Aprendizaje Automático”

Julián Vázquez Sampedro
23/10/2023

Importación de bibliotecas necesarias

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

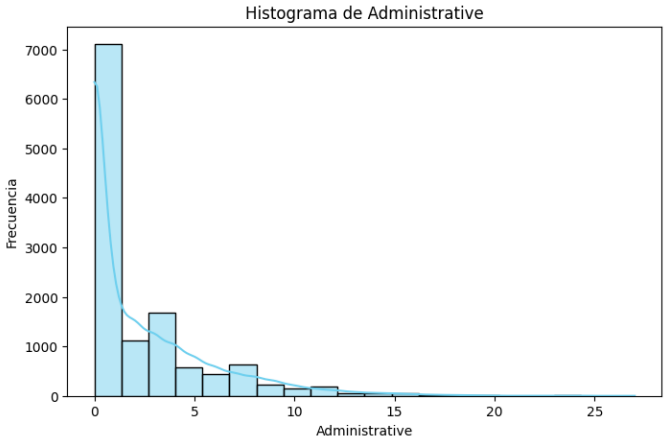
Obtenemos información sobre variables (categóricas y numéricas)

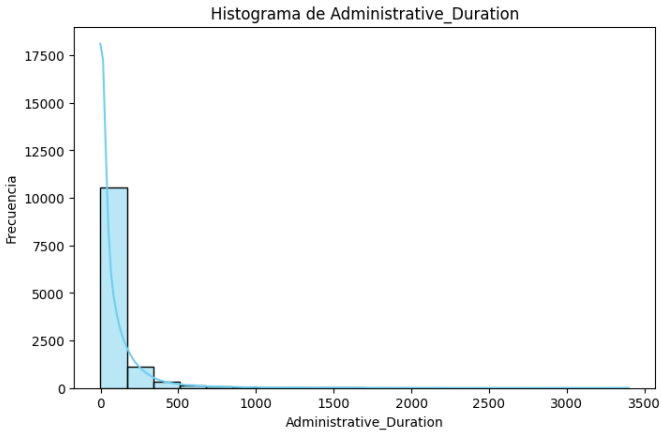
```
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12316 non-null  float64
 1   Administrative_Duration  12316 non-null  float64
 2   Informational            12316 non-null  float64
 3   Informational_Duration   12316 non-null  float64
 4   ProductRelated           12316 non-null  float64
 5   ProductRelated_Duration  12316 non-null  float64
 6   BounceRates              12316 non-null  float64
 7   ExitRates                12316 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(10), int64(4), object(2)
memory usage: 1.5+ MB
None
Variables numéricas:  Index(['Administrative', 'Administrative_Duration', 'Informational',
       'Informational_Duration', 'ProductRelated', 'ProductRelated_Duration',
       'BounceRates', 'ExitRates', 'PageValues', 'SpecialDay',
       'OperatingSystems', 'Browser', 'Region', 'TrafficType'],
      dtype='object')
Variables categóricas:  Index(['Month', 'VisitorType'], dtype='object')
```

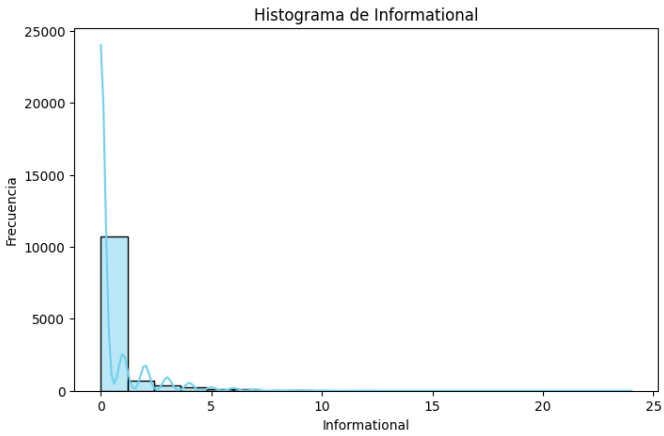
Realización de un análisis de las variables del dataset de Google Analytics como pueden ser histogramas, boxplots, etc.

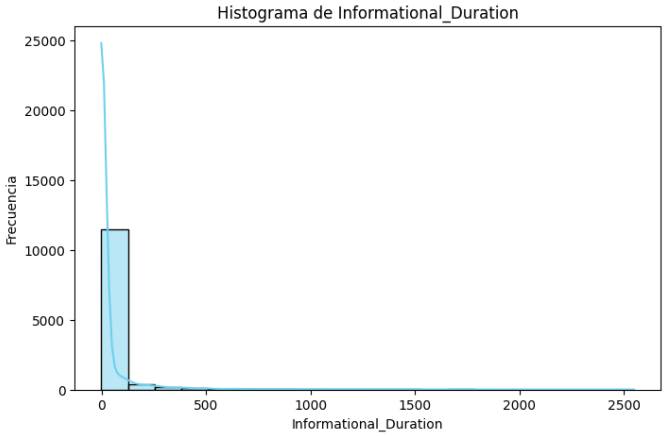
Se puede observar una frecuencia de clientes potenciales elevada en la mayoría de los histogramas donde el grado de conversión de clientes es dependiente del tiempo. Para calcular esta conversión en cada variable habría que realizar el cociente entre clientes convertidos y clientes potenciales.

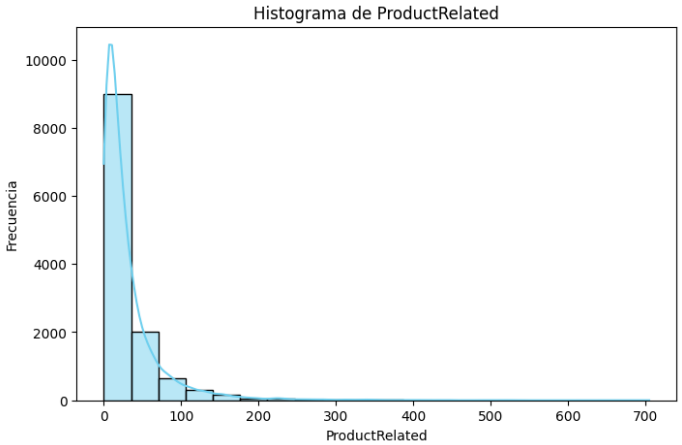
En términos de negocio y visualizando simplemente los valores visuales obtenidos en cada histograma o en cada gráfico boxplot tendría que verse una homogeneidad en las barras en la medida de lo posible, y a lo largo del tiempo. Esto garantizaría que la mayor parte de clientes potenciales acaban convirtiéndose en clientes finales.

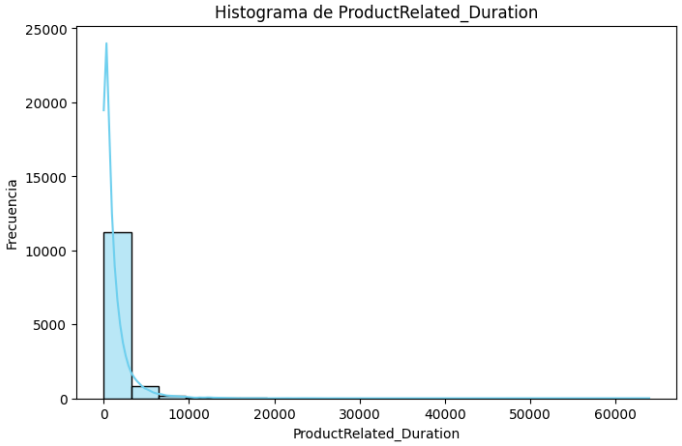


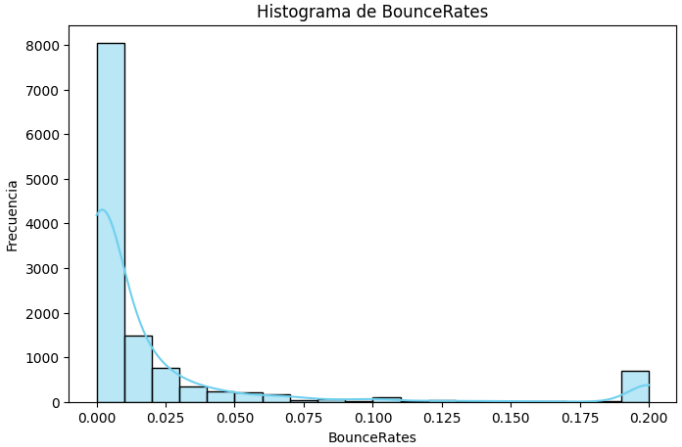


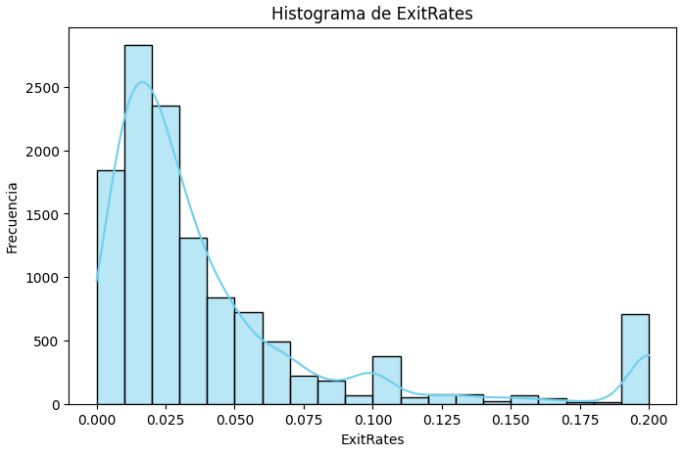


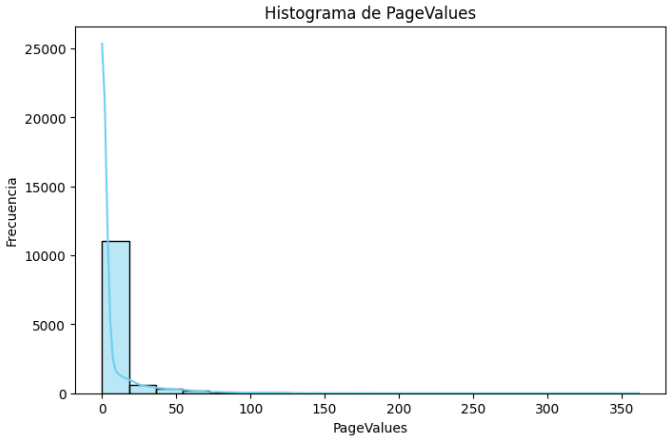

Histograma de Informational_Duration
Frecuencia
Informational_Duration
0
5000
10000
15000
20000
25000
0
500
1000
1500
2000
2500

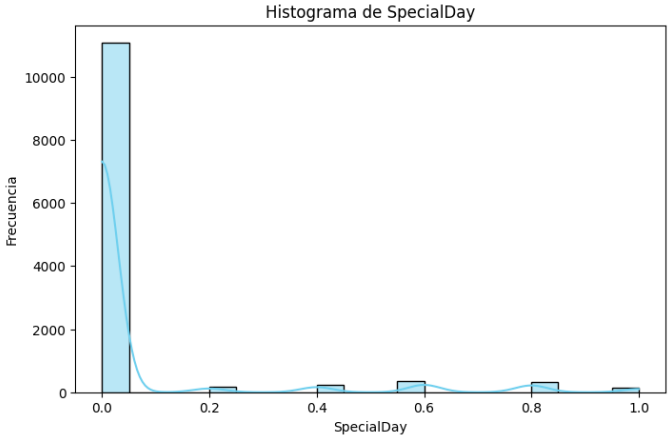

Histograma de ProductRelated
Frecuencia
ProductRelated
0
2000
4000
6000
8000
10000
0
100
200
300
400
500
600
700

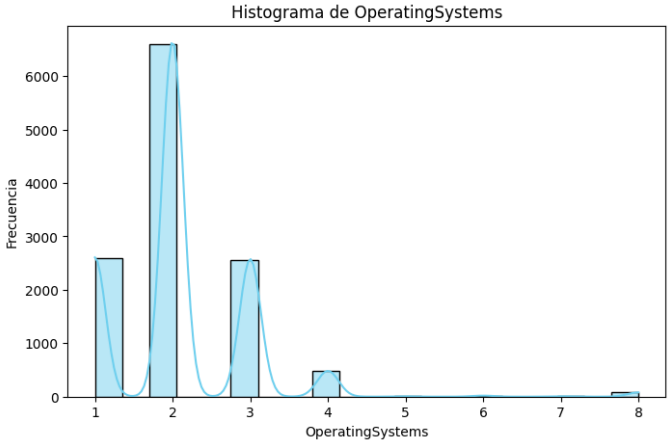

Histograma de ProductRelated_Duration
Frecuencia
ProductRelated_Duration
0
5000
10000
15000
20000
25000
0
10000
20000
30000
40000
50000
60000

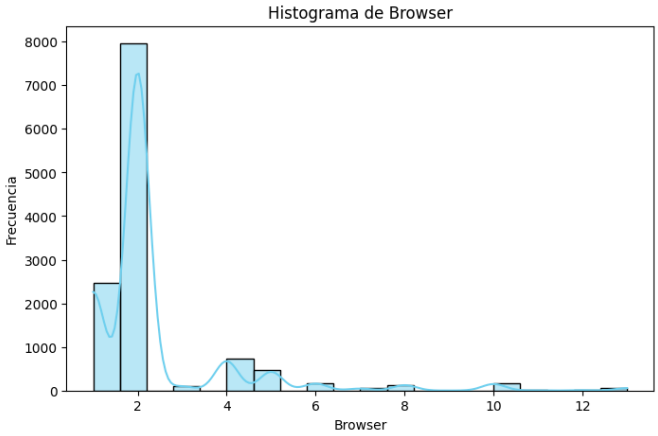

Histograma de BounceRates
Frecuencia
BounceRates
0
1000
2000
3000
4000
5000
6000
7000
8000
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

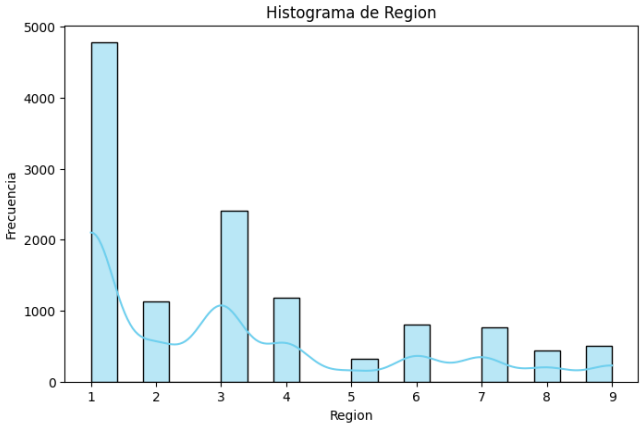

Histograma de ExitRates
Frecuencia
ExitRates
0
500
1000
1500
2000
2500
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

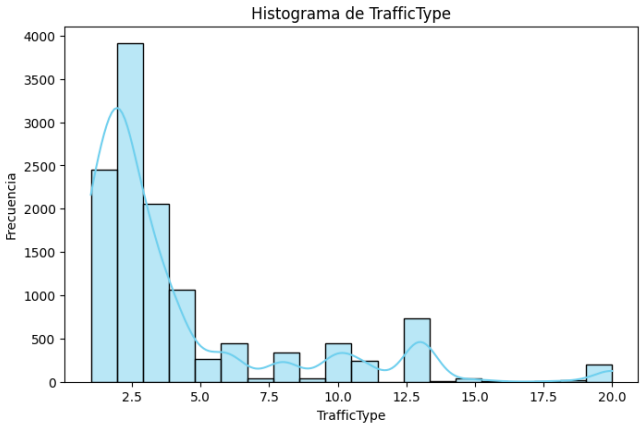

Histograma de PageValues
Frecuencia
PageValues
0
5000
10000
15000
20000
25000
0
50
100
150
200
250
300
350

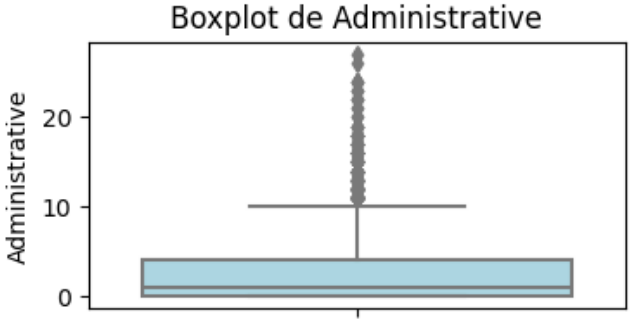

Histograma de SpecialDay
Frecuencia
SpecialDay
0
2000
4000
6000
8000
10000
0.0
0.2
0.4
0.6
0.8
1.0

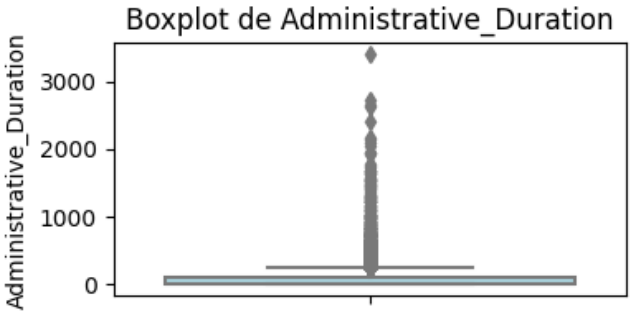

Histograma de OperatingSystems
Frecuencia
OperatingSystems
0
1000
2000
3000
4000
5000
6000
1
2
3
4
5
6
7
8

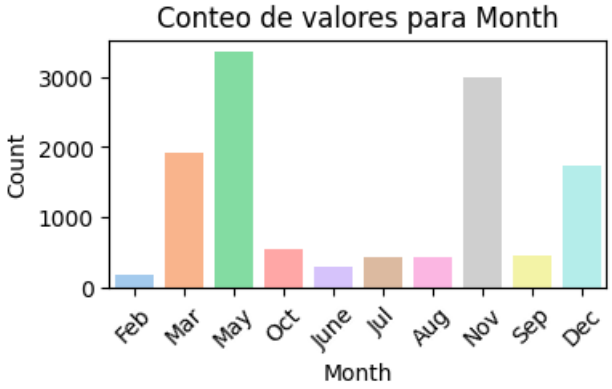

Histograma de Browser
Frecuencia
Browser
0
1000
2000
3000
4000
5000
6000
7000
8000
2
4
6
8
10
12


Histograma de Region
Frecuencia
Region
0
1000
2000
3000
4000
5000
1
2
3
4
5
6
7
8
9


Histograma de TrafficType
Frecuencia
TrafficType
0
500
1000
1500
2000
2500
3000
3500
4000
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Boxplot de Administrative
Administrative
0
10
20


Boxplot de Administrative_Duration
Administrative_Duration
3000
2000
1000
0


Conteo de valores para Month
Count
3000
2000
1000
0
Feb
Mar
May
Oct
June
Jul
Aug
Nov
Sep
Dec
Month


Conteo de valores para VisitorType
Count
10000
7500
5000
2500
0
Returning_Visitor
New_Visitor
Other
VisitorType

Valores faltantes

```
Valores faltantes en el conjunto de datos:  Administrative             14
Administrative_Duration    14
Informational              14
Informational_Duration     14
ProductRelated             14
ProductRelated_Duration    14
BounceRates                14
ExitRates                  14
PageValues                  0
SpecialDay                  0
Month                       0
OperatingSystems            0
Browser                     0
Region                      0
TrafficType                 0
VisitorType                 0
Weekend                     0
Revenue                     0
dtype: int64
```

Valores atípicos

```python
# Identificamos valores atípicos
def detect_outliers(df, features):
    outlier_indices = []
    for col in features:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        outlier_step = 1.5 * IQR
        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 + outlier_step)].index
        outlier_indices.extend(outlier_list_col)
    outlier_indices = list(set(outlier_indices))
    return outlier_indices

outliers = detect_outliers(data_cleaned, numeric_columns)
print("Valores atípicos detectados y eliminados: ", len(outliers))
data_cleaned = data_cleaned.drop(outliers, axis = 0).reset_index(drop = True)
```

✓ 0.0s

```
Valores atípicos detectados y eliminados:  9772
```

Convertir variables categóricas y numéricas

```python
# Codificación one-hot (dummies) para 'Month'
data_encoded = pd.get_dummies(data_cleaned, columns = ['Month'], drop_first = True)
"""El método get_dummies (Pandas) se emplea para convertir variables categóricas en numéricas y crear nuevas columnas binarias .Lo utilizamos para aplicar la codificación one-hot a la variable 'Month'
El argumento drop_first = True se utiliza para evitar la multicolinealidad, eliminando la primera columna generada.
"""
# Utilizamos LabelEncoder para la variable 'VisitorType'
label_encoder = LabelEncoder()
data_encoded['VisitorType'] = label_encoder.fit_transform(data_encoded['VisitorType'])
"""LabelEncoder se emplea para codificar etiquetas categóricas en variables numéricas. Se asigna un número entero único a cada etiqueta única en la serie. Por ej: Rojo, Azul y Verde sería igual a 0, 1 y 2.
```

```
El código proporcionado, se utiliza LabelEncoder para convertir la variable categórica 'VisitorType' en datos numéricos
```

Eliminación de variables no útiles y estandarización

```
# 'Region' se considera menos relevante para el análisis
data_cleaned = data_cleaned.drop(['Region'], axis=1)
```

```
## Estandarizamos los datos

# Inicializamos el objeto StandardScaler()
scaler = StandardScaler() #Permite estandarizacion de media a 0 y desv estandar a 1
# Aplicamos la estandarización a las columnas numéricas
data_cleaned[numeric_columns]                                                    =
scaler.fit_transform(data_cleaned[numeric_columns])
```

Dividir los datos en *train* y en test. Con los datos de *train* se pretende ajustar modelos con CrossValidation y GridSearch.

- Utilizar un modelo lineal. Entre los modelos lineales están las regresiones logísticas, las regresiones lineales, etc.
- Utilizar un modelo de redes neuronales.
- Utilizar cualquier otro modelo de clasificación.

```
LogisticRegression:                precision    recall  f1-score   support

       False       0.97      1.00      0.98       493
        True       0.00      0.00      0.00        16

    accuracy                           0.97       509
   macro avg       0.48      0.50      0.49       509
weighted avg       0.94      0.97      0.95       509

confusion_matrix:  [[493   0]
 [ 16   0]]
Neural Network:                precision    recall  f1-score   support

       False       0.97      1.00      0.99       493
        True       1.00      0.06      0.12        16

    accuracy                           0.97       509
   macro avg       0.99      0.53      0.55       509
weighted avg       0.97      0.97      0.96       509

Confusion Matrix:  [[493   0]
 [ 15   1]]
Random Forest Classifier:                precision    recall  f1-score   support

       False       0.97      1.00      0.99       493
        True       1.00      0.06      0.12        16

    accuracy                           0.97       509
   macro avg       0.99      0.53      0.55       509
weighted avg       0.97      0.97      0.96       509

Confusion Matrix:  [[493   0]
 [ 15   1]]
```

Optimizar algún parámetro de cada modelo utilizando CrossValidation y GridSearch, o de la forma que se estime oportuna, siempre justificándolo.

El empleo de GridSearchCV y la validación cruzada (cross validation) es fundamental en el proceso de entrenamiento y evaluación de modelos de aprendizaje automático por las siguientes razones:

- Optimización de hiperparámetros: Los modelos de aprendizaje automático tienen hiperparámetros que deben ajustarse para lograr un rendimiento óptimo. GridSearchCV permite probar exhaustivamente una serie de combinaciones de hiperparámetros para encontrar la configuración óptima que maximice el rendimiento del modelo.
- Prevención del sobreajuste: La validación cruzada es crucial para evitar el sobreajuste del modelo. Dividir los datos en conjuntos de entrenamiento y prueba múltiples mediante la validación cruzada proporciona una evaluación más robusta del rendimiento del modelo en datos no vistos, lo que ayuda a detectar si el modelo está sobreajustando los datos de entrenamiento.
- Mejor estimación del rendimiento del modelo: La validación cruzada proporciona una mejor estimación del rendimiento del modelo en datos no vistos. Al repetir el proceso de entrenamiento y evaluación en diferentes particiones de los datos, se obtiene una evaluación más estable y fiable del rendimiento del modelo en comparación con una única división de entrenamiento/prueba.
- Generalización del modelo: Al utilizar GridSearchCV en combinación con la validación cruzada, se puede encontrar la configuración de hiperparámetros que generaliza mejor el modelo en una variedad de datos. Esto es esencial para asegurar que el modelo funcione bien en diferentes conjuntos de datos y no esté demasiado ajustado a un conjunto específico.

```
## Empleamos GridSearchCV (sklearn.model_selection) para la optimización de parámetros en cada método
### Regresión Logística
# Definimos la rejilla de parámetros
param_grid_logistic = {'C':[0.1, 1, 10, 100], 'max_iter': [100, 200, 300]}
# Inicializamos GridSearchCV
grid_logistic = GridSearchCV(logistic_model, param_grid_logistic, cv = 5)
# Ajustamos el modelo con los datos de entrenamiento
grid_logistic.fit(x_train, y_train)
# Imprimimos los mejores parámetros y el mejor resultado
print("Mejores parámetros para Regresión Logística: ", grid_logistic.best_params_)
print("Mejor resultado para Regresión Logística: ", grid_logistic.best_score_)

### Redes neuronales
# Realizamos el mismo proceso(rejilla_parametros, Inicializador, Ajuste, Imprimir)
```

```
param_grid_neural   =   {'hidden_layer_sizes':[(50,),   (100,),   (50,   50)],
'activation': ['Logistic', 'relu']}
grid_neural = GridSearchCV(neural_network_model, param_grid_neural, cv = 5)
grid_neural.fit(x_train, y_train)
print("Mejores parámetros para Redes Neuronales:", grid_neural.best_params_)
print("Mejor resultado para Redes Neuronales:", grid_neural.best_score_)

### Random Forest
param_grid_forest = {'n_estimators':[100, 200, 300], 'max_depth': [10, 20, 30]}
grid_forest = GridSearchCV(random_forest_model, param_grid_forest, cv = 5)
grid_forest.fit(x_train, y_train)
print("Mejores parámetros para Random Forest:", grid_forest.best_params_)
print("Mejor resultado para Random Forest:", grid_forest.best_score_)
```

RESULTADOS

Mejores parámetros para Regresión Logística: {'C': 0.1, 'max_iter': 100}
Mejor resultado para Regresión Logística: 0.9656019656019657

Mejores parámetros para Redes Neuronales: {'activation': 'relu', 'hidden_layer_sizes': (50,)}
Mejor resultado para Redes Neuronales: 0.9660933660933662

Mejores parámetros para Random Forest: {'max_depth': 10, 'n_estimators': 200}
Mejor resultado para Random Forest: 0.9660933660933662

Elegir el mejor modelo de los tres según la métrica ROC en CrossValidation. Predecir Test y obtener una métrica estimada.

```
## Para seleccionar el mejor modelo segun metrica ROC en validacion cruzada
empleamos la funcion 'cross_val_score'
"""
La Curva ROC y el area bajo la curva (AUC) se emplean para evaluar la capacidad
de un modelo de clasificacion para distinguir entre clases
- La Curva ROC representa la tasa de verdaderos positivos (Sensibilidad) en
funcion de la tasa de falsos positivos (1 - Especificidad) a medida que se varia
el umbral de clasificacion
- AUC es un resumen numerico de la capacidad de discriminacion del modelo, donde
un valor 1 = rendimiento perfecto y un valor 0.5 = rendimiento aleatorio

La Validacion Cruzada se emplea para evaluar el rendimiento de un modelo de
aprendizaje automatico y para mitigar problemas de sobreajuste y subajuste
Empleamos la validacion cruzada en 5 pliegues (cv = 5)
"""
# Calculamos el área bajo la curva ROC para cada modelo empleando validacion
cruzada
logistic_roc_scores = cross_val_score(grid_logistic.best_estimator_, x, y, cv =
5, scoring='roc_auc')
```

```
neural_roc_scores = cross_val_score(grid_neural.best_estimator_, x, y, cv = 5,
scoring='roc_auc')
forest_roc_scores = cross_val_score(grid_forest.best_estimator_, x, y, cv = 5,
scoring = 'roc_auc')
# Seleccionamos el modelo con el mejor ROC
best_model = max([(logistic_roc_scores.mean(), 'Regresion Logistica'),
                  (neural_roc_scores.mean(), 'Redes neuronales'),
                  (forest_roc_scores.mean(), 'Random Forest')], key = lambda x:
x[0])
# Imprimimos el mejor modelo
print("Mejor modelo: ", best_model[1])
# Predecimos en el cnjunto de prueba y obtenemos métricas estimadas
if best_model[1] == 'Regresion Logistica':
    y_pred = grid_logistic.best_estimator_.predict(x_test)
elif best_model[1] == 'Redes neuronales':
    y_pred = grid_neural.best_estimator_.predict(x_test)
else:
    y_pred = grid_forest.best_estimator_.predict(x_test)

# Calculamos Metrica ROC en el conjunto de prueba
roc_test = roc_auc_score(y_test, y_pred)
print("Métrica ROC estimada en el conjunto de prueba ", roc_test)
```

Mejor modelo: Random Forest

Métrica ROC estimada en el conjunto de prueba 0.5302358012170385

Umbralizar las probabilidades utilizando el umbral que maximice el área bajo la curva ROC.

```
"""
Para la umbralizacion de las probabilidades empleando el umbral que maximie el
area bajo ROC:
- Se calculan las probabilidades de clase positiva en TEST para el mejor modelo
- Se emplea 'roc_curve' para obtener falsos y verdaderos positivos y umbrales
para predicciones
- Encontramos el umbral que maximiza el indice en ROC = punto mas cercano a la
esquina superior izquierda del grafico ROC
- Aplicamos este umbral a las probabilidades originales para clasificar
observaciones positivas o negativas segun umbral optimo
"""

# Obtenemos probabilidades de clase positiva para cada observacion en TEST
probs = grid_logistic.best_estimator_.predict_proba(x_test)[:, 1]
# Obtenemos falsos positivos y verdaderos positivos (tasas) y umbrales
fpr, tpr, thresholds = roc_curve(y_test, probs)
# Encontramos el umbral que maximiza el indice en la curva ROC
optimal_idx = np.argmax(tpr-fpr)
optimal_threshold = thresholds[optimal_idx]
# Aplicamos el umbral óptimo a las probabilidades originales
y_pred_thresholded = (probs >= optimal_threshold.astype(int))

# Imprimimos los resultados de la umbralización
print("Umbral óptimo:", optimal_threshold)
print("Resultados después de la umbralización:")
```

```python
print(classification_report(y_test, y_pred_thresholded))
print("Matriz de confusión después de la umbralización:")
print(confusion_matrix(y_test, y_pred_thresholded))
# Calculamos la métrica ROC después de la umbralización
roc_thresholded = roc_auc_score(y_test, y_pred_thresholded)
print("Métrica ROC después de la umbralización:", roc_thresholded)
```

```
Umbral óptimo: 0.042577926068225094
Resultados después de la umbralización:
              precision    recall  f1-score   support

       False       0.00      0.00      0.00       493
        True       0.03      1.00      0.06        16

    accuracy                           0.03       509
   macro avg       0.02      0.50      0.03       509
weighted avg       0.00      0.03      0.00       509

Matriz de confusión después de la umbralización:
[[  0 493]
 [  0  16]]
Métrica ROC después de la umbralización: 0.5
```