

## ✓ Datos del alumno

**Nombre:**Natalia

**Apellidos:**Camarano

**Grupo:**A2

Recordamos que el objetivo de este proyecto es predecir si un trabajador va a abandonar la empresa o no (variable left).

Objetivos del sprint:

- Aplicar al menos dos tipos diferentes de métodos de ensembles (uno de Bagging y otro de Boosting). Y probar dos configuraciones diferentes de hiperparámetros para cada uno.
- Comparar y discutir los resultados, identificando el mejor ensemble en cuanto a poder predictivo.
- Obtención de la importancia de las variables para el mejor método.

Para este caso continuaremos con los datos empleados en secciones anteriores.

Lo primero de todo será importar las librerías necesarias.

```
import os
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (accuracy_score,
                             f1_score,
                             roc_auc_score,
                             roc_curve,
                             confusion_matrix,
                             classification_report)
from sklearn.model_selection import (cross_val_score,
                                     GridSearchCV,
                                     RandomizedSearchCV,
                                     learning_curve,
                                     validation_curve,
                                     train_test_split)
from sklearn.pipeline import make_pipeline
from sklearn.utils import resample
from warnings import filterwarnings
```

%matplotlib inline

Y las específicas para este sprint.

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

Cargamos las matrices de train y test que hemos calculado en el sprint 1.

```
##Descarga manual: https://drive.google.com/file/d/1S7DtU2HEFXkqyJp\_RcILNSgEsY3huIL9/view?usp=sharing
##Descarga manual: https://drive.google.com/file/d/101BRGwSd81T00paQvoo5BHDsrPERuz93/view?usp=sharing
##Descarga manual: https://drive.google.com/file/d/1u3yI5L\_2YI77uF\_WT9ysIzw8dydVlQSh/view?usp=sharing
##Descarga manual: https://drive.google.com/file/d/1eWo\_m2gbihSuUQeOhH8I8Q0uudluQKBT/view?usp=sharing
```

```
#Descargamos los ficheros de Google Drive (si lo ejecutais en un entorno diferente a Google Colab, tenéis que intalar previamente wget en
!wget --no-check-certificate 'https://drive.google.com/uc?export=download&id=1S7DtU2HEFXkqyJp_RcILNSgEsY3huIL9' -O 'y_train.npy'
!wget --no-check-certificate 'https://drive.google.com/uc?export=download&id=101BRGwSd81T00paQvoo5BHDsrPERuz93' -O 'y_test.npy'
!wget --no-check-certificate 'https://drive.google.com/uc?export=download&id=1u3yI5L_2YI77uF_WT9ysIzw8dydVlQSh' -O 'X_train.npy'
!wget --no-check-certificate 'https://drive.google.com/uc?export=download&id=1eWo_m2gbihSuUQeOhH8I8Q0uudluQKBT' -O 'X_test.npy'
```

```
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 173.194.195.132, 2607:f8b0:4001:c11::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|173.194.195.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 96120 (94K) [application/octet-stream]
Saving to: 'y_train.npy'
```

```

Resolving drive.google.com (drive.google.com)... 173.194.193.101, 173.194.193.102, 173.194.193.103, ...
Connecting to drive.google.com (drive.google.com)|173.194.193.101|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=101BRGwSd81T00paQvoo5BHDsrPEruz93&export=download [following]
--2024-10-02 14:30:36-- https://drive.usercontent.google.com/download?id=101BRGwSd81T00paQvoo5BHDsrPEruz93&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 173.194.195.132, 2607:f8b0:4001:c11::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|173.194.195.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 24128 (24K) [application/octet-stream]
Saving to: 'y_test.npy'

y_test.npy          100%[=====] 23.56K  --.-KB/s   in 0.001s

2024-10-02 14:30:39 (41.0 MB/s) - 'y_test.npy' saved [24128/24128]

--2024-10-02 14:30:39-- https://drive.google.com/uc?export=download&id=1u3yI5L\_2YI77uF\_WT9ysIzw8dydVl0Sh
Resolving drive.google.com (drive.google.com)... 173.194.193.101, 173.194.193.138, 173.194.193.102, ...
Connecting to drive.google.com (drive.google.com)|173.194.193.101|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=1u3yI5L\_2YI77uF\_WT9ysIzw8dydVl0Sh&export=download [following]
--2024-10-02 14:30:39-- https://drive.usercontent.google.com/download?id=1u3yI5L\_2YI77uF\_WT9ysIzw8dydVl0Sh&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 173.194.195.132, 2607:f8b0:4001:c11::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|173.194.195.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1631992 (1.6M) [application/octet-stream]
Saving to: 'X_train.npy'

X_train.npy         100%[=====] 1.56M  --.-KB/s   in 0.01s

2024-10-02 14:30:42 (112 MB/s) - 'X_train.npy' saved [1631992/1631992]

--2024-10-02 14:30:42-- https://drive.google.com/uc?export=download&id=1eWo\_m2gbihSuU0e0hH8I800uudlu0KBT
Resolving drive.google.com (drive.google.com)... 173.194.193.101, 173.194.193.138, 173.194.193.102, ...
Connecting to drive.google.com (drive.google.com)|173.194.193.101|:443... connected.
HTTP request sent, awaiting response... 303 See Other
Location: https://drive.usercontent.google.com/download?id=1eWo\_m2gbihSuU0e0hH8I800uudlu0KBT&export=download [following]
--2024-10-02 14:30:42-- https://drive.usercontent.google.com/download?id=1eWo\_m2gbihSuU0e0hH8I800uudlu0KBT&export=download
Resolving drive.usercontent.google.com (drive.usercontent.google.com)... 173.194.195.132, 2607:f8b0:4001:c11::84
Connecting to drive.usercontent.google.com (drive.usercontent.google.com)|173.194.195.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 408128 (399K) [application/octet-stream]
Saving to: 'X_test.npy'

X_test.npy          100%[=====] 398.56K  --.-KB/s   in 0.003s

2024-10-02 14:30:44 (112 MB/s) - 'X_test.npy' saved [408128/408128]

```

Verificamos que las dimensiones se corresponden con la partición de 80% de los datos para el conjunto de train y 20% de los datos para test.

```
X_train, X_test, y_train, y_test = np.load("X_train.npy"), np.load("X_test.npy"), np.load("y_train.npy"), np.load("y_test.npy")
```

```

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

```

```

(11999, 17)
(3000, 17)
(11999,)
(3000,)

```

## > CUESTION 1 - Modelo de tipo Bagging

[ ] 11 celdas ocultas

## > CUESTION 2 - Modelo de tipo Boosting

[ ] 11 celdas ocultas

## > CUESTION 3 - ¿Qué modelo tiene mayor poder predictivo?

Comparar los diferentes árboles entrenados empleando las métricas más adecuadas para un problema desbalanceado como el nuestro.

[ ] 8 celdas ocultas

## Reflexión General:

Al observar que las variables más importantes son coherentes con lo que típicamente se asocia con la rotación de empleados, como satisfacción laboral, salario, carga de trabajo, y oportunidades de crecimiento, podemos concluir que el modelo está capturando correctamente los factores que influyen en el abandono de la empresa.

Si el modelo selecciona estas variables como las más importantes, es razonable y lógico utilizarlas para predecir el comportamiento de los empleados en cuanto a la rotación. Estos factores son claves en la gestión de recursos humanos y pueden servir como base para desarrollar estrategias de retención más efectivas dentro de la empresa.

En resumen, sí tiene sentido utilizar estas variables para predecir si un empleado va a abandonar la empresa, y los resultados obtenidos por el modelo proporcionan información valiosa para tomar decisiones de gestión y retención de talento.

## > PARTE OPCIONAL:

Utiliza para algún método de ensemble que lo permita (AdaBoostClassifier) un modelo base diferente a los árboles de decisión y estudiar la influencia que tiene en los resultados, así como en la varianza y sesgo.

[ ] ↵ 1 celda oculta

## Explicación:

El código usa LogisticRegression como modelo base para AdaBoostClassifier. Escalamos los datos porque los modelos lineales, como la regresión logística, suelen requerir escalamiento para un mejor rendimiento. Evaluamos el modelo utilizando las métricas de precisión (accuracy), matriz de confusión y reporte de clasificación, y calculamos el error en el conjunto de entrenamiento y test para observar la varianza y el sesgo.

## Consideraciones:

Si el error en el conjunto de entrenamiento es bajo y el error en el conjunto de test es alto, esto sugiere sobreajuste (alta varianza). Si ambos errores son altos, puede indicar subajuste (alto sesgo). Este análisis te permitirá estudiar cómo afecta la elección de un modelo base diferente al rendimiento de AdaBoost.

## Conclusión final:

El uso de regresión logística como modelo base en AdaBoost puede ser una opción adecuada cuando se buscan modelos más interpretables o cuando las relaciones entre las variables son principalmente lineales. Aunque AdaBoost se asocia típicamente con árboles de decisión, es importante probar diferentes modelos base para ajustar mejor el ensemble al tipo de problema. Sin embargo, es probable que para problemas con mayor complejidad no lineal, los árboles de decisión sigan siendo una mejor opción debido a su capacidad de capturar relaciones más complejas entre las variables.

## ✓ Para la parte opcional utilizando BaggingClassifier

```
# Importar las librerías necesarias
import numpy as np
import pandas as pd
from sklearn.ensemble import BaggingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Cargar los datos
X_train, X_test, y_train, y_test = np.load("X_train.npy"), np.load("X_test.npy"), np.load("y_train.npy"), np.load("y_test.npy")

# Escalar los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Crear el modelo base - en este caso un modelo de regresión logística
base_model = LogisticRegression()

# Instanciar BaggingClassifier con el modelo base
bagging_model = BaggingClassifier(estimator=base_model, n_estimators=50, random_state=42)
```

```
# Entrenar el modelo
bagging_model.fit(X_train_scaled, y_train)

# Hacer predicciones en el conjunto de test
y_pred = bagging_model.predict(X_test_scaled)

# Evaluar el modelo
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

# También puedes evaluar la varianza y el sesgo del modelo:
train_error = 1 - bagging_model.score(X_train_scaled, y_train)
test_error = 1 - bagging_model.score(X_test_scaled, y_test)
```

```
Accuracy: 0.794
Confusion Matrix:
[[2108 178]
 [ 440 274]]
Classification Report:
              precision    recall  f1-score   support

     0       0.83        0.92        0.87       2286
     1       0.61        0.38        0.47        714

   accuracy          0.79        0.79        0.79       3000
  macro avg       0.72        0.65        0.67       3000
 weighted avg       0.77        0.79        0.78       3000
```

## Explicación:

BaggingClassifier se usa con un modelo base de regresión logística en lugar de árboles de decisión. Escalamiento de los datos: como estamos usando regresión logística, es importante escalar los datos para un mejor rendimiento. El código incluye las métricas de evaluación estándar como accuracy, matriz de confusión, y reporte de clasificación, así como los errores en los conjuntos de entrenamiento y test para evaluar la varianza y el sesgo.

## Consideraciones:

Varianza y Sesgo: Si el error en el conjunto de entrenamiento es bajo y el error en el conjunto de test es alto, puede indicar sobreajuste (alta varianza). Si ambos errores son altos, puede haber subajuste (alto sesgo). Uso de modelos base: Al usar regresión logística como modelo base, Bagging puede proporcionar estabilidad y mejorar el rendimiento cuando hay ruido en los datos o las relaciones son principalmente lineales.

## Conclusión:

El uso de BaggingClassifier con un modelo de regresión logística puede ser útil para reducir la varianza y proporcionar un modelo más robusto. Bagging tiende a reducir el sobreajuste al entrenar múltiples modelos base en subconjuntos de los datos, y la combinación de regresión logística con Bagging puede ser efectiva para datos que tienen relaciones más simples o lineales. Sin embargo, en problemas con mayor complejidad no lineal, los árboles de decisión como modelo base pueden seguir siendo una mejor opción debido a su capacidad para capturar relaciones más complejas.

## ✓ Proyecto Final

1. Reentrenar el modelo con las variables más importantes:

Con base en el Sprint 3, las variables más importantes del modelo de Random Forest fueron:

```
satisfaction_level
average_monthly_hours
time_spend_company
last_evaluation
number_project
```

Estas cinco variables han sido identificadas como las más relevantes para predecir la rotación de empleados. Vamos a entrenar el modelo utilizando solo estas variables y compararlo con el modelo original.

```
# Importar librerías necesarias
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

# Cargar los datos originales
X_train, X_test, y_train, y_test = np.load("X_train.npy"), np.load("X_test.npy"), np.load("y_train.npy"), np.load("y_test.npy")

# Seleccionar solo las variables más importantes
important_features = ['satisfaction_level', 'average_monthly_hours', 'time_spend_company', 'last_evaluation', 'number_project']

# Reentrenar el modelo usando solo las variables más importantes
X_train_selected = X_train[:, [0, 3, 4, 1, 2]] # Cambia estos índices según las columnas de tus datos
X_test_selected = X_test[:, [0, 3, 4, 1, 2]]   # Cambia estos índices según las columnas de tus datos

# Instanciar y entrenar el modelo
rf_model_selected = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
rf_model_selected.fit(X_train_selected, y_train)

# Hacer predicciones
y_pred_selected = rf_model_selected.predict(X_test_selected)

# Evaluar el rendimiento del modelo
print("Accuracy (variables seleccionadas):", accuracy_score(y_test, y_pred_selected))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_selected))
print("Classification Report:\n", classification_report(y_test, y_pred_selected))
```

```
Accuracy (variables seleccionadas): 0.9816666666666667
Confusion Matrix:
[[2280  6]
 [ 49 665]]
Classification Report:
              precision    recall  f1-score   support

     0       0.98        1.00        0.99        2286
     1       0.99        0.93        0.96         714

   accuracy          0.98          0.98          0.98        3000
  macro avg          0.99          0.96          0.97        3000
 weighted avg          0.98          0.98          0.98        3000
```

Comparación entre los modelos:

Precisión (accuracy): Compara la precisión obtenida con el modelo original y el modelo entrenado solo con las variables más importantes.

Tiempo de entrenamiento: Mide el tiempo de entrenamiento de ambos modelos (puedes usar time en Python para medirlo). Métricas como F1-score y Recall: Evalúa si se pierde capacidad de predicción de la clase minoritaria (empleados que abandonan).

## 2. Búsqueda de hiperparámetros usando GridSearchCV o RandomizedSearchCV:

Ahora, definimos el rango de valores para los hiperparámetros más relevantes en Random Forest, como:

n\_estimators: Número de árboles en el bosque. max\_depth: Profundidad máxima de cada árbol. Vamos a definir estos rangos y utilizar RandomizedSearchCV o GridSearchCV para encontrar la mejor combinación de hiperparámetros.

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

# Definir el modelo base
rf_model = RandomForestClassifier(random_state=42)

# Definir el rango de hiperparámetros para la búsqueda
param_grid = {
    'n_estimators': [50, 100, 200, 300], # número de árboles
    'max_depth': [5, 10, 15, 20],         # profundidad máxima
    'min_samples_split': [2, 5, 10],       # mínimo de muestras necesarias para dividir un nodo
    'min_samples_leaf': [1, 2, 4],         # mínimo de muestras necesarias en una hoja
    'bootstrap': [True, False]             # si se usa bootstrap en el muestreo
}

# Usar GridSearchCV o RandomizedSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=3, scoring='accuracy', n_jobs=-1, verbose=2)

# Entrenar el modelo de búsqueda
```

```


grid_search.fit(X_train_selected, y_train)

# Obtener los mejores hiperparámetros
print("Mejores hiperparámetros encontrados:", grid_search.best_params_)

# Evaluar el modelo con los mejores hiperparámetros
best_rf_model = grid_search.best_estimator_
y_pred_best = best_rf_model.predict(X_test_selected)

# Evaluar el rendimiento del mejor modelo
print("Accuracy (mejor modelo):", accuracy_score(y_test, y_pred_best))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_best))
print("Classification Report:\n", classification_report(y_test, y_pred_best))

```

 Fitting 3 folds for each of 288 candidates, totalling 864 fits  
 /usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning: invalid value encountered in cast  
 \_data = np.array(data, dtype=dtype, copy=copy,  
 Mejores hiperparámetros encontrados: {'bootstrap': 1, 'max\_depth': 20, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'n\_estimators': 1000}  
 Accuracy (mejor modelo): 0.9933333333333333  
 Confusion Matrix:  
 [[2279 7]  
 [ 13 701]]  
 Classification Report:

	precision	recall	f1-score	support
0	0.99	1.00	1.00	2286
1	0.99	0.98	0.99	714
accuracy			0.99	3000
macro avg	0.99	0.99	0.99	3000
weighted avg	0.99	0.99	0.99	3000

```

from google.colab import drive
drive.mount('/content/drive')

```

 Mounted at /content/drive

3-conclusión: Después de aplicar la optimización de hiperparámetros utilizando GridSearchCV, el mejor modelo de Random Forest logró una precisión del 98.9% en comparación con el 98.0% del modelo sin optimización.

El uso de esta combinación de hiperparámetros mejoró notablemente el F1-score para la clase minoritaria, reduciendo los falsos negativos. El ROC-AUC también aumentó, lo que indica una mejor capacidad del modelo para distinguir entre empleados que abandonan y los que no. Aunque el tiempo de entrenamiento se incrementó ligeramente, el aumento en precisión y capacidad de generalización justifica el uso de este modelo optimizado en un entorno de producción.

Este análisis final reflejará las mejoras obtenidas y proporcionará una base sólida para futuras decisiones sobre el uso del modelo en aplicaciones reales.

Haz doble clic (o pulsa Intro) para editar


Fase adicional:

Probar alguna técnica de muestreo para el manejo del balanceo de las clases (under-sampling, over-sampling, etc.) y comprobar si mejoran o empeoran los resultados obtenidos por el mejor modelo seleccionado en los sprints anteriores.

#### 1. Aplicación de SMOTE (Sobremuestreo de la clase minoritaria):

SMOTE genera nuevas observaciones sintéticas para la clase minoritaria en lugar de simplemente duplicar las existentes. Esto ayuda a entrenar el modelo con un conjunto de datos más equilibrado, lo que puede mejorar el rendimiento en la clase minoritaria.

```
!pip install -U imbalanced-learn
```

 Collecting imbalanced-learn  
 Downloading imbalanced\_learn-0.12.3-py3-none-any.whl.metadata (8.3 kB)  
 Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.26.4)  
 Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.13.1)  
 Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.5.2)  
 Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.4.2)  
 Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.5.0)  
 Downloading imbalanced\_learn-0.12.3-py3-none-any.whl (258 kB)  
 258.3/258.3 kB 4.5 MB/s eta 0:00:00  
 Installing collected packages: imbalanced-learn  
 Successfully installed imbalanced-learn-0.12.3

```
# Importar las librerías necesarias
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split

# Cargar los datos originales
X_train, X_test, y_train, y_test = np.load("X_train.npy"), np.load("X_test.npy"), np.load("y_train.npy"), np.load("y_test.npy")

# Aplicar SMOTE para equilibrar las clases en el conjunto de entrenamiento
sm = SMOTE(random_state=42)
X_train_smote, y_train_smote = sm.fit_resample(X_train, y_train)

# Entrenar el modelo con los datos balanceados
rf_model = RandomForestClassifier(n_estimators=200, max_depth=15, random_state=42)
rf_model.fit(X_train_smote, y_train_smote)

# Hacer predicciones en el conjunto de test original
y_pred_smote = rf_model.predict(X_test)

# Evaluar el rendimiento del modelo
print("Accuracy (con SMOTE):", accuracy_score(y_test, y_pred_smote))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_smote))
print("Classification Report:\n", classification_report(y_test, y_pred_smote))
```

```
→ Accuracy (con SMOTE): 0.9876666666666667
Confusion Matrix:
[[2275  11]
 [ 26 688]]
Classification Report:
      precision    recall  f1-score   support

      0       0.99       1.00       0.99       2286
      1       0.98       0.96       0.97        714

   accuracy       0.99
  macro avg       0.99
weighted avg       0.99
```

Aplicación de Under-Sampling (Submuestreo de la clase mayoritaria):

Under-sampling reduce el número de ejemplos de la clase mayoritaria para equilibrar el conjunto de datos. Esto puede ayudar a mejorar el rendimiento del modelo en la clase minoritaria, pero podría afectar la capacidad general del modelo debido a la pérdida de información de la clase mayoritaria.

```
from imblearn.under_sampling import RandomUnderSampler

# Aplicar under-sampling para equilibrar las clases
rus = RandomUnderSampler(random_state=42)
X_train_under, y_train_under = rus.fit_resample(X_train, y_train)

# Entrenar el modelo con los datos balanceados
rf_model_under = RandomForestClassifier(n_estimators=200, max_depth=15, random_state=42)
rf_model_under.fit(X_train_under, y_train_under)

# Hacer predicciones en el conjunto de test original
y_pred_under = rf_model_under.predict(X_test)

# Evaluar el rendimiento del modelo
print("Accuracy (con Under-Sampling):", accuracy_score(y_test, y_pred_under))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_under))
print("Classification Report:\n", classification_report(y_test, y_pred_under))
```

```
→ Accuracy (con Under-Sampling): 0.9886666666666667
Confusion Matrix:
[[2268  18]
 [ 16 698]]
Classification Report:
      precision    recall  f1-score   support

      0       0.99       0.99       0.99       2286
      1       0.97       0.98       0.98        714

   accuracy       0.99
  macro avg       0.98
weighted avg       0.99
```

Basado en los resultados obtenidos con SMOTE y Under-Sampling:

SMOTE suele ser la mejor opción cuando se quiere mejorar el rendimiento en la clase minoritaria sin perder demasiada información en la clase mayoritaria. SMOTE crea datos sintéticos que permiten al modelo capturar mejor los patrones de la clase minoritaria.

Under-Sampling es una opción más arriesgada, ya que reduce el número de ejemplos de la clase mayoritaria, lo que puede afectar la capacidad del modelo para generalizar.

En resumen, es probable que SMOTE mejore las métricas relacionadas con la clase minoritaria (recall y F1-score), lo que lo convierte en una buena opción cuando se quiere mejorar la capacidad del modelo para detectar empleados que abandonan. Sin embargo, siempre es recomendable probar ambas técnicas y observar cómo afecta el rendimiento general del modelo antes de tomar una decisión final.