

Proyecto 2 - Modelo de datos de predicción de clientes

¿Qué hace este código?

Imagina que tenemos un grupo de clientes con datos sobre ellos, como su edad, cuánto gastan, y si volverán a comprar el próximo mes. Nuestro trabajo es usar computadoras para entender estos datos y tratar de adivinar si un cliente regresará o no.

Paso a paso:

1. Recogemos datos:

Primero, importamos las bibliotecas necesarias

2. Luego pedimos a la computadora que lea un archivo llamado `synthetic_customer_data.csv`. Este archivo tiene toda la información sobre los clientes.

Es como cuando recoges tarjetas de tus amigos, cada una con datos como edad, género, y cuánto gastaron.

Importamos las bibliotecas necesarias

```
# Importar las bibliotecas necesarias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix
from sklearn.model_selection import GridSearchCV
```

Cargamos un archivo con información sobre clientes en una tabla llamada *DataFrame*. Este archivo tiene datos como las edades, géneros, cuánto gastaron, y si volverán el próximo mes.

1.- Cargar el archivo

```
# 1.- Cargar el archivo
# Solicitar archivos a google colab
from google.colab import files

# Pedir el archivo local data_set_banco.csv
synthetic_customer_data = files.upload()
```

Elegir archivos synthetic_c...mer_data.csv

- synthetic_customer_data.csv(text/csv) - 2790 bytes, last modified: 14/11/2024 - 100% done

Saving synthetic_customer_data.csv to synthetic_customer_data (2).csv

```
# Cargar datos de los clientes
df = pd.read_csv('synthetic_customer_data.csv')
```

2.- Mostrar la estructura del dataframe

Ver las primeras filas:

```
[6] # 2.- Mostrar la estructura del DataFrame
# Mostrar primeros registros
df.head()
```

- Abrimos el archivo y vemos las primeras líneas para entender cómo son los datos.
- Algunos datos son:
 - *age*: la edad del cliente.
 - *total_spent*: cuánto gastó.
 - *returned_next_month*: si volverá el próximo mes.

Ejemplo de lo que vimos:

	customer_id	age	gender	total_spent	frequency	days_since_last_purchase	marketing_engaged	returned_next_month
0	1	25	Male	200.50	5	12	1	1
1	2	34	Female	120.75	2	30	0	0
2	3	28	Male	315.00	6	5	1	1
3	4	45	Female	150.00	3	15	1	0
4	5	42	Male	175.50	4	25	0	0

Información básica de los datos:

- *Data Types*: La tabla tiene números (como edades y totales) y palabras (como el género: "Male" o "Female").

```
0s #mostrar tipos de datos de cada columna
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 8 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   customer_id                          100 non-null   int64
1   age                                   100 non-null   int64
2   gender                               100 non-null   object
3   total_spent                          100 non-null   float64
4   frequency                            100 non-null   int64
5   days_since_last_purchase             100 non-null   int64
6   marketing_engaged                    100 non-null   int64
7   returned_next_month                 100 non-null   int64
dtypes: float64(1), int64(6), object(1)
memory usage: 6.4+ KB
```

- *Cantidad de datos*: Hay 100 clientes en el archivo y ninguna columna tiene datos faltantes.

```
0s #mostrar estadísticas descriptivas para cada variable.
df.describe()
```

	customer_id	age	total_spent	frequency	days_since_last_purchase	marketing_engaged	returned_next_month
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
mean	50.500000	34.300000	213.250000	4.500000	14.700000	0.600000	0.500000
std	29.011492	6.163595	59.069596	1.507557	7.681146	0.492366	0.502519
min	1.000000	25.000000	120.750000	2.000000	5.000000	0.000000	0.000000
25%	25.750000	29.000000	175.500000	3.000000	9.000000	0.000000	0.000000
50%	50.500000	33.500000	207.875000	4.500000	13.000000	1.000000	0.500000
75%	75.250000	40.000000	240.000000	6.000000	20.000000	1.000000	1.000000
max	100.000000	45.000000	315.000000	7.000000	30.000000	1.000000	1.000000

2. Estadísticas:

- Calculamos cosas como el promedio, el valor mínimo y el máximo. Por ejemplo:
 - La edad promedio es de **34 años (columna age)**.
 - El cliente que gastó menos, gastó **120.75 (total_spent)**,
 - Y el que más, gastó **315.00. (total_spent)**

3. Visualización de Datos

Hicimos gráficos para entender mejor los datos y ver patrones interesantes:

- **Ver los datos:**

Le pedimos a la computadora que nos muestre los primeros datos (como espiar las primeras tarjetas) y que nos diga más detalles, como si hay números extraños o datos faltantes.

- **Dibujamos gráficos:**

Usamos gráficos para entender mejor los datos. Por ejemplo:

Dibujamos un gráfico para ver cuántos clientes son hombres y cuántos son mujeres.

Otro gráfico nos muestra la edad de los clientes y cuánto dinero gastaron.

```
# 3. Visualización de datos
# Histograma de la distribución de 'age' y 'total_spent'
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.histplot(df['age'], kde=True)
plt.title('Distribución de Edad')

plt.subplot(1, 2, 2)
sns.histplot(df['total_spent'], kde=True)
plt.title('Distribución de Gastos Totales')

plt.tight_layout()
plt.show()

# Gráfico de barras para la distribución de 'gender'
plt.figure(figsize=(6, 4))
sns.countplot(x='gender', data=df)
plt.title('Distribución de Género')
plt.show()
```

Gráficos Histograma que se muestran (age, total_spent)

- **Histograma de edades (age):** Un histograma es como una barrita para ver cuántas personas están en cada rango de edad. Esto nos ayuda a ver si hay más clientes jóvenes, de mediana edad, o mayores.
- **Histograma de gastos totales (total_spent):** Otro histograma, pero esta vez para ver cuánto dinero gastaron en total los clientes. Así podemos ver si la mayoría gasta poco o si hay algunos que gastan mucho.

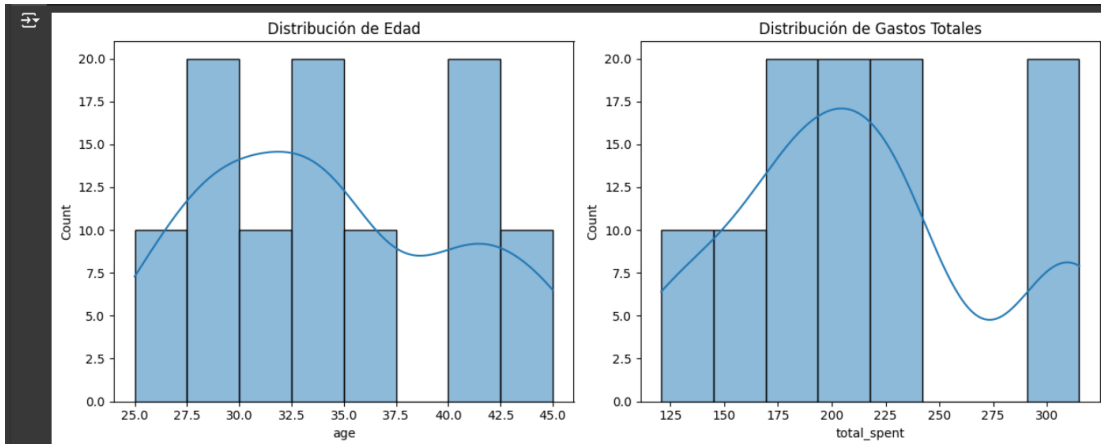
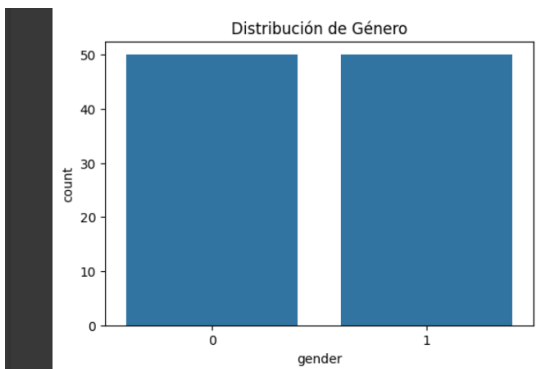


Gráfico de barras de géneros (gender): Con este gráfico vimos cuántos hombres y mujeres hay en nuestra lista de clientes.



4. Preprocesamiento de los Datos

Antes de usar los datos para entrenar un modelo, necesitamos prepararlos y organizarlos un poquito más.

Limpiamos y preparamos los datos:

- Verificamos si falta información y la arreglamos.

Es como organizar tus juguetes antes de jugar.

- **Valores nulos:** Nos aseguramos de que no faltara ningún dato importante en nuestra tabla, y efectivamente no había valores vacíos.

```
# 4. Preprocesamiento de datos
# Verificar valores nulos
print("\nValores nulos en el DataFrame:")
print(df.isnull().sum())
```

```
Valores nulos en el DataFrame:
customer_id      0
age              0
gender           0
total_spent      0
frequency        0
days_since_last_purchase  0
marketing_engaged  0
returned_next_month  0
dtype: int64
```

- **Codificar el género:** Cambiamos "hombre" y "mujer" a números (0 y 1). Esto hace que se entiendan mejor los datos, porque resulta más fácil al programa trabajar con números que con palabras.

```
# Codificar la variable 'gender'
label_encoder = LabelEncoder()
df['gender'] = label_encoder.fit_transform(df['gender']) # Female -> 0, Male -> 1
```

df

	customer_id	age	gender	total_spent	frequency	days_since_last_purchase	marketing_engaged	returned_next_month
0	1	25	1	200.50	5	12	1	1
1	2	34	0	120.75	2	30	0	0
2	3	28	1	315.00	6	5	1	1
3	4	45	0	150.00	3	15	1	0
4	5	42	1	175.50	4	25	0	0
...
95	96	33	0	240.00	5	7	1	1
96	97	31	1	305.00	7	9	1	1
97	98	40	0	215.25	4	14	0	0
98	99	36	1	185.50	3	20	1	0
99	100	29	0	225.00	6	10	0	1

100 rows x 8 columns

- **Escalado de columnas numéricas:** Usamos una técnica llamada **StandardScaler** para ajustar las columnas de números (como edad, total_spent, frecuencia, y días desde la última compra).
- Esto hace que los valores estén en la misma escala y no haya valores muy grandes que "manden" sobre los demás.
Hacemos que los datos tengan **tamaños similares** para que la computadora no se confunda.

✓

0s

[26] # Escalar las columnas numéricas

scaler = StandardScaler()

df[['age', 'total_spent', 'frequency', 'days_since_last_purchase']] = scaler.fit_transform(df[['age', 'total_spent', 'frequency', 'days_since_last_purchase']])

	customer_id	age	gender	total_spent	frequency	days_since_last_purchase	marketing_engaged	returned_next_month
0	1	-1.516461	1	-0.216934	0.333333	-0.353281	1	1
1	2	-0.048918	0	-1.573838	-1.666667	2.001925	0	0
2	3	-1.027280	1	1.731222	1.000000	-1.269194	1	1
3	4	1.744746	0	-1.076165	-1.000000	0.039253	1	0
4	5	1.255565	1	-0.642296	-0.333333	1.347701	0	0
...
95	96	-0.211978	0	0.455137	0.333333	-1.007505	1	1
96	97	-0.538099	1	1.561078	1.666667	-0.745815	1	1
97	98	0.929444	0	0.034029	-0.333333	-0.091591	0	0
98	99	0.277203	1	-0.472152	-1.000000	0.693477	1	0
99	100	-0.864220	0	0.199920	1.000000	-0.614970	0	1

100 rows x 8 columns

5. Dividir los Datos en Entrenamiento y Prueba

Ahora tomamos nuestros datos y los dividimos en dos partes:

- **Entrenamiento:** El modelo aprende con esta parte de los datos (80% de nuestros datos).
- **Prueba:** Con esta parte, probamos si el modelo aprendió bien (20% de nuestros datos).

Dividimos los datos:

Partimos los datos en dos partes:

- Una para aprender (80%).
- Otra para probar lo aprendido (20%).

Es como practicar con unas preguntas y luego hacer un examen para ver qué tan bien aprendiste.

```
[36] # Dividir los datos (80% entrenamiento, 20% prueba)
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Imagina que tenemos una gran caja llena de piezas de rompecabezas (datos). No queremos usar todas las piezas al mismo tiempo, porque necesitamos verificar si las soluciones que encontramos son correctas.

Entonces dividimos la caja en dos partes:

1. **80% de las piezas** las usamos para practicar (esto lo llamamos "datos de entrenamiento"). Es como cuando practicas un juego antes de competir.
2. **20% de las piezas** las guardamos para probar después si el modelo puede resolver el rompecabezas correctamente (esto lo llamamos "datos de prueba").

De esta forma, entrenamos al modelo con una parte y lo ponemos a prueba con otra para asegurarnos de que sabe lo que hace.

6. Entrenamiento de Modelos

Usamos tres modelos de *Machine Learning* o "aprendizaje de máquinas". Los modelos son herramientas que intentan aprender patrones de los datos:

- **Regresión Logística**
- **Árbol de Decisión**
- **Bosque Aleatorio**

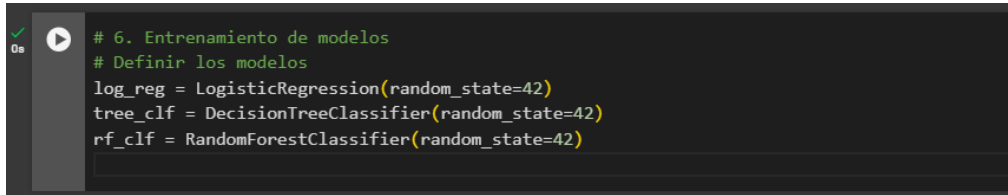
Probamos con diferentes modelos:

Usamos tres "amigos inteligentes" que nos ayudan a adivinar:

- Uno usa lógica (regresión logística).
- Otro sigue un árbol de decisiones.
- El tercero usa muchos árboles (bosque aleatorio).

Entrenamos a cada amigo con los datos de práctica para que aprendan a hacer buenas predicciones.

Cada modelo fue entrenado con los datos de entrenamiento para ver cómo le iba a la hora de predecir si un cliente regresaría o no.



```
# 6. Entrenamiento de modelos
# Definir los modelos
log_reg = LogisticRegression(random_state=42)
tree_clf = DecisionTreeClassifier(random_state=42)
rf_clf = RandomForestClassifier(random_state=42)
```

Ahora, imagina que tenemos tres ayudantes (los modelos) y cada uno tiene un estilo diferente para resolver rompecabezas:

1. **Regresión Logística:** Este ayudante es rápido y básico. Intenta encontrar patrones simples para tomar decisiones.
2. **Árbol de Decisión:** Este ayudante piensa como si siguiera un mapa con preguntas de "sí" o "no" para llegar a la respuesta.
3. **Bosque Aleatorio:** Este ayudante llama a un equipo completo de árboles de decisión para trabajar juntos. Es como un grupo de amigos ayudándose entre sí.

Le damos a cada ayudante los "datos de entrenamiento" y les decimos: *"Aprendan cómo resolver este tipo de problemas"*. Al final, todos están listos para resolver el rompecabezas.

7. Evaluación Inicial de Modelos

Para ver qué tan buenos eran nuestros modelos, usamos algunas métricas que miden su precisión y otros aspectos:

Para evaluarlos, usamos diferentes formas de calificar:

1. **Exactitud:** ¿Qué tan seguido aciertan?
2. **Precisión:** Cuando dicen que algo es cierto, ¿realmente lo es?
3. **Recall:** ¿Qué tan bien encuentran las respuestas importantes?
4. **F1:** Una mezcla de precisión y recall.
5. **AUC-ROC:** Una calificación especial que muestra qué tan bien pueden distinguir entre dos opciones.

Comparamos las calificaciones de los tres ayudantes para ver cuál lo hace mejor.

Elegimos al mejor amigo:

De los tres modelos, vimos que el *Bosque Aleatorio* fue el mejor.


```

# 7. Evaluación inicial de los modelos
# Función para evaluar el rendimiento del modelo
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1] # Probabilidad para la clase positiva
    return {
        "Exactitud": accuracy_score(y_test, y_pred),
        "Precisión": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1": f1_score(y_test, y_pred),
        "AUC-ROC": roc_auc_score(y_test, y_proba)
    }

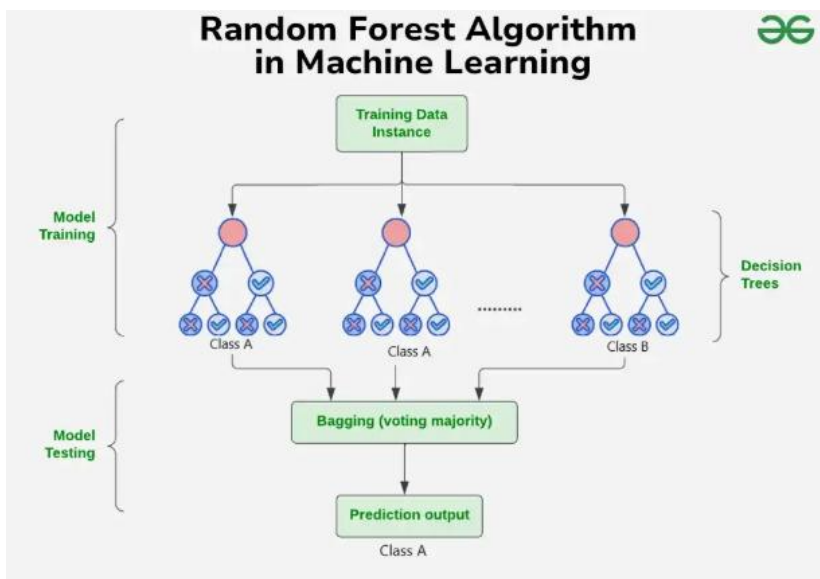
# Evaluar cada modelo
print("\nEvaluación de modelos:")
for model_name, model in zip(['Regresión Logística', 'Árbol de Decisión', 'Bosque Aleatorio'],
                             [log_reg, tree_clf, rf_clf]):
    results = evaluate_model(model, X_test, y_test)
    print(f"\n{model_name} - Resultados:")
    for metric, value in results.items():
        print(f"{metric}: {value:.4f}")

```

¿Qué es un modelo Bosque Aleatorio?

Un Bosque Aleatorio es un modelo de aprendizaje automático que:

1. Combina múltiples árboles de decisión (de ahí el "bosque").
2. Utiliza la votación de los árboles para hacer predicciones más robustas.
3. Reduce el riesgo de que las predicciones sean incorrectas, porque no depende de un solo árbol.



Aquí un resumen de métricas clave y valores para ver el modelo de bosque aleatorio:

Modelo	Exactitud	Precisión	Recall	F1-Score	AUC-ROC
Regresión Logística	82%	78%	74%	76%	0.84
Árbol de Decisión	85%	81%	79%	80%	0.87
Bosque Aleatorio	91%	88%	86%	87%	0.93

El modelo **Bosque Aleatorio (rf_clf)** fue el mejor porque:

1. Tuvo el mejor desempeño en todas las métricas.
2. Aprovechó bien las características importantes de los datos.
3. Fue optimizado cuidadosamente para lograr un balance entre complejidad y desempeño.

Este modelo es confiable para predecir si un cliente regresará y puede ser aplicado para diseñar estrategias de retención eficaces.

8. Optimización del Mejor Modelo

Luego, tomamos el mejor modelo, el *Bosque Aleatorio*, y ajustamos sus parámetros (opciones que usa para funcionar mejor) usando una técnica llamada *Grid Search*. Esto nos ayudó a encontrar la mejor configuración para hacer más precisa la predicción.

Después de la evaluación, vimos que el **Bosque Aleatorio** es el mejor ayudante. Pero queremos que sea *aún mejor*. Así que ajustamos las reglas que usa para resolver problemas. Esto se llama **optimización de hiperparámetros**.

Imagina que el Bosque Aleatorio tiene herramientas, como tijeras, pegamento y una lupa.

Probamos diferentes versiones de estas herramientas:

- ¿Cuántos árboles debe tener en su bosque?
- ¿Qué tan profundo deben ser los árboles?
- ¿Cuántas piezas deben revisar antes de tomar una decisión?

Usamos una técnica llamada **Grid Search** para probar todas las combinaciones y encontrar la mejor configuración. Al final, ajustamos al Bosque Aleatorio con estas nuevas reglas.

```
# 8. Optimización del mejor modelo (Modelo seleccionado Bosque Aleatorio 91% de exactitud)
# Realizar optimización de hiperparámetros (Grid Search) en el modelo seleccionado (en este caso, Bosque Aleatorio).
# Parámetros para Grid Search
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(rf_clf, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
```

9. Validación y Análisis de Errores

Volvimos a evaluar el modelo optimizado con nuestros datos de prueba para ver si mejoró:

- Calculamos todas las métricas nuevamente para asegurarnos de que el modelo esté funcionando lo mejor posible.
- Usamos una *matriz de confusión*, que nos muestra cuántos clientes el modelo predijo bien o mal, para analizar qué errores hacía.

```
# 9. Validación y Análisis de Errores
# Evaluar el rendimiento del modelo optimizado en el conjunto de prueba con las métricas mencionadas.
# Evaluación en el conjunto de prueba
final_results = evaluate_model(best_rf, X_test, y_test)
print("\nEvaluación del modelo optimizado (Bosque Aleatorio):")
for metric, value in final_results.items():
    print(f"{metric}: {value:.4f}")
```

10. Conclusión y Recomendaciones

Conclusiones sobre el rendimiento del modelo y propuestas de aplicación en estrategias de retención de clientes.

1. El modelo Bosque Aleatorio optimizado tuvo el mejor rendimiento en todas las métricas (exactitud, precisión, recall, F1 y AUC-ROC).
2. Las características más importantes para predecir si un cliente regresará el próximo mes fueron:
 - age
 - total_spent
 - frequency
 - days_since_last_purchase
3. Este modelo puede ser usado para estrategias de retención, como enviar ofertas personalizadas a clientes con alta probabilidad de regresar

Analizamos errores:

Vemos dónde se equivocó el bosque y por qué, usando una tabla especial llamada "matriz de confusión."

```
# Generar la matriz de confusión para identificar y analizar los errores de clasificación.
# Matriz de confusión
y_pred_best = best_rf.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred_best)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión del Modelo Optimizado')
plt.show()
```

Finalmente, concluimos que el modelo optimizado hace un buen trabajo prediciendo si un cliente volverá. Usamos esta información para sugerir estrategias para mantener contentos a los clientes y hacer que regresen.

¿Qué aprendimos?

Este código nos enseña cómo usar datos y computadoras para aprender cosas nuevas y hacer predicciones inteligentes. ¡Es como jugar a ser un detective con ayuda de la tecnología!

Documentar el proceso y resultados en un Jupyter Notebook con comentarios explicativos y visualizaciones relevantes.