UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

MAESTRÍA EN CIENCAS DE DATOS

APRENDIZAJE AUTOMÁTICO

MAESTRO: JOSÉ ALBERTO BENAVIDES VÁZQUEZ

TAREA #6 APRENDIZAJE SUPERVIZADO

ALUMNO: EDWIN MARTÍN ROMERO SILVA

MATRÍCULA: 1731276

Aprendizaje Supervisado

El aprendizaje supervisado se utiliza para entrenar modelos predictivos basados en datos etiquetados.

Estos algoritmos se entrenan **utilizando un conjunto de datos etiquetado** (A mí me gusta nombrar esta etiqueta variable objetivo o variable Target), lo que significa que se proporciona al modelo un conjunto de ejemplos de entrada junto con las respuestas deseadas o etiquetas correspondientes.

El objetivo del aprendizaje supervisado es que el modelo con base en datos etiquetados aprenda a realizar predicciones precisas en nuevos datos no etiquetados.

Mi base de datos

Mi base de datos tiene 20,508 registros y está enfocada en Crédito, contiene información del cliente al momento de la solicitud de crédito y mi problema está enfocado en predecir si el cliente será un buen o un mal cliente, por lo que utilizaré la variable 'MALO' como variable objetivo o Target.

Mi BD tiene variables numéricas como el ingreso y la edad, pero también contiene variables categóricas como el estatus de la vivienda o la ocupación del cliente.

Estas variables categóricas fueron reducidas a el menor número de categorías posibles y posteriormente las reemplacé por variables Dummy (0/1) para que pudieran formar parte del entrenamiento del modelo. Utilicé la función get dummies de Pyhton.

	Float e Int
0	CNT_CHILDREN
1	AMT_INCOME_TOTAL
2	DAYS_BIRTH
3	DAYS_EMPLOYED
4	FLAG_MOBIL
5	FLAG_WORK_PHONE
6	FLAG_PHONE
7	FLAG_EMAIL
8	CNT_FAM_MEMBERS
10	FLG_Genero_Masculino
11	FLG_auto
12	FLG_casa

	Categóricas
0	NAME_EDUCATION_TYPE
1	NAME_FAMILY_STATUS
2	NAME_HOUSING_TYPE
3	Cat_Tipo_Ingresos
4	Cat_Ocupacion

	Target
9	MALO

Target

La variable MALO es una bandera que marca con un ‘1’ cuando el cliente se atrasó mas de 60 días en los primeros 12 meses desde que se origina el crédito, y marca con un ‘0’ en cualquier otro caso. De los 20,508 registros, 2017 están marcados co mo clientes malos. Es decir 9.84% de la población.

Generalmente se escogen 60 días de atraso y no una cantidad menor (por ejemplo 30 días) ya que, en cantidades menores, es mas probable que el cliente se ponga al corriente con su crédito (Llegar a 0 días de atraso de nuevo).

Como mi variable target es una variable que solo contiene 0 y 1, necesito un algoritmo supervisado que permita resolver problemas de clasificación Binaria.

Segmentación Train/Test

Para entrenar el modelo segmenté mis datos en 2 partes, Train y Test, con el objetivo de comprobar que el modelo funciona para predecir incluso en datos distintos a los de entrenamiento. Es decir, para reducir los problemas de sobreajuste.

70% para la muestra de Train y 30% para la muestra de Test.

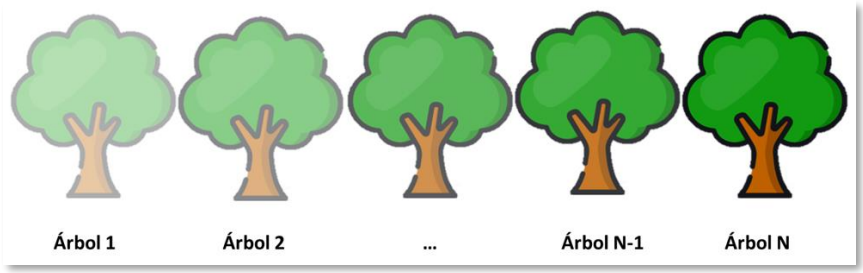
```
df = pd.get_dummies(df, drop_first = True)
```

```
df_train = df.sample(frac = 0.7 ,random_state = 42)
df_test = df.drop(df_train.index)

y_train = df_train['MALO']
x_train = df_train.drop(columns = ['MALO'])

y_test = df_test['MALO']
x_test = df_test.drop(columns = ['MALO'])
```

Utilicé 2 algoritmos de Aprendizaje Supervisado en mi base de datos: Xtreme Gradient Boosting y Light Gradient Boosting.

Xtreme Gradient Boosting

- Es un algoritmo de aprendizaje automático supervisado que se basa en la técnica de boosting.
- El boosting es una técnica de ensamble en la que se combinan N modelos simples (generalmente árboles de decisión) para crear un modelo más fuerte. El objetivo principal del boosting es mejorar el rendimiento del modelo, disminuyendo los errores del modelo anterior en iteraciones sucesivas.
- Es decir, el XGB funciona entrenando N modelos simples enfocándose en corregir los errores del modelo anterior y enfocandose en las instancias que el modelo previo clasificó incorrectamente.
- Este algoritmo es ampliamente utilizado y es conocido por su eficiencia y capacidad para manejar una variedad de problemas de regresión y clasificación.



Investigué un poco sobre los hiper parámetros más importantes de este algoritmo, muchos de ellos están enfocados en reducir el sobreajuste. Sin embargo, utilicé prácticamente todos los valores por defecto. Únicamente especifiqué que mi problema es de clasificación binaria con objetive = 'binary logistic'.

```
from xgboost import XGBClassifier

# Crear el modelo
model = XGBClassifier( objective='binary:logistic',
                       n_estimators=1000)
```

n_estimators: Especifica el número de árboles (estimadores) que se deben entrenar en el modelo. Un valor más alto generalmente conlleva a un mejor rendimiento, pero también puede aumentar el riesgo de sobreajuste.

learning_rate: Controla la tasa de aprendizaje utilizada para actualizar los pesos de los árboles en cada iteración.

max_depth: Define la profundidad máxima de cada árbol en el conjunto.

min_child_weight: Controla la cantidad mínima de instancias necesarias en cada nodo hoja.

subsample: Este hiperparámetro controla la fracción de datos de entrenamiento utilizados para ajustar cada árbol.

colsample_by: Fracción de características (columnas) utilizadas para ajustar cada árbol y en cada nivel de cada árbol, respectivamente.

Resultados:

Los resultados con este algoritmo y los hiperparametros por defecto dejan mucho que desear ya que el AUC (Area Under Curve) es apenas aceptable en la muestra de Train y muy malo en la muestra de Test.

La diferencia tan grande entre el AUC de Train y test podría indicar problemas de sobreajuste.

```
from sklearn.metrics import roc_auc_score

y_pred_train = model.predict(x_train)
auc_train    = roc_auc_score(y_train, y_pred_train)

y_pred_test = model.predict(x_test)
auc_test    = roc_auc_score(y_test, y_pred_test)

print(f'AUC TRAIN: {auc_train}')
print(f'AUC TEST: {auc_test}')
```

```
AUC TRAIN: 0.7543024338429639
AUC TEST: 0.5764789054005622
```

Light Gradient Boosting

```
import lightgbm as lgb

params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
}

# Crear un conjunto de datos LightGBM a partir de los datos de entrenamiento
train_data = lgb.Dataset(x_train, label=y_train)

# Entrenar el modelo
num_round = 100  # Número de iteraciones (puedes ajustarlo)
bst = lgb.train(params, train_data, num_round)
```

Este modelo comparte muchas características de Xtreme Gradient Boosting Model, sin embargo, se caracteriza por ser más veloz. Debido a su eficiente manejo de memoria y algoritmo de búsqueda de hojas a nivel de hoja, LightGBM suele ser más rápido en conjuntos de datos grandes y puede ser más adecuado para problemas de "big data".

Los resultados en este modelo son ligeramente superiores en ambas muestras Train y Test, respecto al modelo xgboost.

Parece que utilizar este modelo es ligeramente superior que una decisión aleatoria.

```
y_pred = bst.predict(x_train)
auc_train = roc_auc_score(y_train, y_pred)

y_pred = bst.predict(x_test)
auc_test = roc_auc_score(y_test, y_pred)

print(f'AUC TRAIN: {auc_train}')
print(f'AUC TEST: {auc_test}')
```

```
AUC TRAIN: 0.859469541023702
AUC TEST: 0.6665972057349123
```

Concluyo que, utilizando los parámetros por default, el Light Gradient Boosting es superior.

AUC (Area Under Curve)

Elegí este valor para medir el desempeño ya que me parece adecuado para mi problema de clasificación binaria. Explicaré en que consiste.

Primero, el ROC es una representación gráfica que se utiliza comúnmente en problemas de clasificación para evaluar el rendimiento de un modelo, particularmente en modelos binarios. La curva ROC muestra la relación entre dos tasas:

- **Tasa de Verdaderos Positivos (TPR):** Esta es la proporción de positivos reales que el modelo clasifica correctamente como positivos. Es decir, mide la capacidad del modelo para identificar instancias positivas.
- **Tasa de Falsos Positivos (FPR):** Esta es la proporción de negativos reales que el modelo clasifica incorrectamente como positivos. Mide la tasa de errores en la que el modelo clasifica incorrectamente instancias negativas como positivas.

La curva ROC traza la TPR en el eje vertical y la FPR en el eje horizontal a medida que se ajusta el umbral de decisión del modelo.

El AUC, es el área debajo de la curva ROC. Cuanto mayor sea el AUC, mejor será el rendimiento del modelo. El AUC varía de 0 a 1, donde:

- AUC < 0.5 indica que el modelo es peor que una decisión aleatoria.
- AUC = 0.5 indica que el modelo tiene un rendimiento similar al aleatorio.
- AUC > 0.5 indica que el modelo tiene cierta capacidad de discriminación y es mejor que una decisión aleatoria.
- AUC = 1 indica un clasificador perfecto que clasifica todas las instancias positivas antes que las negativas.