Selección de Modelo

A continuación, se mostrará el proceso de selección del modelo, las posibles mejoras y conclusiones de la elección

In [1]:
```
import pandas as pd
import numpy as np
import time

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report

import xgboost as xgb
from xgboost import plot_importance

import warnings
warnings.filterwarnings('ignore')
```

0.- Carga de Conjuntos

Se cargan los .csv correspondientes al conjunto de entrenamiento y prueba. Ambos .csv contienen las carácteristicas obtenidas del notebook original. En los .csv, la primera columna corresponde a la clase de la fila.

In [2]:
```
x_train = pd.read_csv('datasets/x_train.csv', header = None).to_numpy()
y_train = np.ravel(pd.read_csv('datasets/y_train.csv', header = None).to_numpy())
x_test = pd.read_csv('datasets/x_test.csv', header = None).to_numpy()
y_test = np.ravel(pd.read_csv('datasets/y_test.csv', header = None).to_numpy())
```

In [3]:
```
print(x_train.shape, x_test.shape)
print(y_train.shape, y_test.shape)
```

```
(45698, 37) (22508, 37)
(45698,) (22508,)
```

Tambien se cargan los .csv correspondientes a los conjuntos seleccionando las características más importantes.

1.- Primera Evaluación

1.1.- Logistic Regression Evaluation

In [4]:
```
logReg = LogisticRegression()
model = logReg.fit(x_train, y_train)
```

In [5]:
```
start = time.time()
y_pred = model.predict(x_test)
end = time.time()
print("Tiempo en predicción:", end - start, "[s]")
```

```
Tiempo en predicción: 0.009990453720092773 [s]
```

In [6]:
```
confusion_matrix(y_test, y_pred)
```

Out[6]:
```
array([[18311,    92],
       [ 3985,   120]], dtype=int64)
```

In [7]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      1.00      0.90     18403
           1       0.57      0.03      0.06      4105

    accuracy                           0.82     22508
   macro avg       0.69      0.51      0.48     22508
weighted avg       0.77      0.82      0.75     22508
```

1.2.- XGBClassifier Evaluation

In [8]:
```
modelxgb = xgb.XGBClassifier(random_state=1, learning_rate=0.01, objective='binary:logistic')
modelxgb = modelxgb.fit(x_train, y_train)
```

```
[16:27:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

In [9]:
```
start = time.time()
y_predxgb = modelxgb.predict(x_test)
```

```
end = time.time()
print("Tiempo en predicción:", end - start, "[s]")
```

```
Tiempo en predicción: 0.04216647148132324 [s]
```

In [10]:
```
confusion_matrix(y_test, y_predxgb)
```

Out[10]:
```
array([[18359,    44],
       [ 4017,    88]], dtype=int64)
```

In [11]:
```
print(classification_report(y_test, y_predxgb))
```

```
              precision    recall  f1-score   support

           0       0.82      1.00      0.90     18403
           1       0.67      0.02      0.04      4105

    accuracy                           0.82     22508
   macro avg       0.74      0.51      0.47     22508
weighted avg       0.79      0.82      0.74     22508
```

1.3.- Conclusión

Es posible apreciar que el modelo de Regresión Logística tiene mejor rendimiento en la clasificación que el modelo de XGBClassifier, ya que, obtiene una mejor clasificación en la clase '1' o 'Atraso' y casi el mismo rendimiento en la clase '0' o 'No Atraso'.

Además, se puede apreciar que en tiempos de procesamiento, el modelo de Regresión Logística tarda menos tiempo (0.004 [s] vs 0.03 [s]) en predecir la clase de un vuelo. Lo cual tiene relevancia a la hora de exponer el modelo y realizar multiples solicitudes en poco tiempo.

Tal como se mencionó en el notebook original, ambos rendimientos no son suficientes para resolver el problema deseado, ya que, la cantidad de falsos negativos es demasiada alta, debido al imbalance de las clases. Es por esto, que se decide evaluar el desempeño de ambos modelos aplicando un **peso** a las clases al momento de entrenar.

El modelo de Regresión Logística posee un parámetro llamado **class_weight**, que permite equiparar el peso entre las clases para mejorar el imbalance que hay en los conjuntos en el entrenamiento. Así también, existe el parámetro **scale_pos_weight** en el modelo de XGBClassifier.

2.- Mejoras y Segunda Evaluación

2.1.- Logistic Regression Improvement

In [12]:
```
logReg = LogisticRegression(class_weight = 'balanced')
model = logReg.fit(x_train, y_train)
```

In [13]:
```
start = time.time()
y_pred = model.predict(x_test)
end = time.time()
print("Tiempo en predicción:", end - start, "[s]")
```

```
Tiempo en predicción: 0.00750279426574707 [s]
```

In [14]:
```
confusion_matrix(y_test, y_pred)
```

Out[14]:
```
array([[10958,  7445],
       [ 1477,  2628]], dtype=int64)
```

In [15]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.60      0.71     18403
           1       0.26      0.64      0.37      4105

    accuracy                           0.60     22508
   macro avg       0.57      0.62      0.54     22508
weighted avg       0.77      0.60      0.65     22508
```

2.2.- XGBClassifier Improvement

Se calcula la proporción de imbalance entre las clases.

In [16]:
```
n_y0 = len(y_train[y_train == 0])
n_y1 = len(y_train[y_train == 1])
scale = n_y0/n_y1
print(scale)
```

```
4.3705488306499
```

In [17]:
```
modelxgb = xgb.XGBClassifier(random_state=1, learning_rate=0.01, objective='binary:logistic', scale_pos_weight = scale)
modelxgb = modelxgb.fit(x_train, y_train)
```

```
[16:27:46] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost
1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Exp
licitly set eval_metric if you'd like to restore the old behavior.
```

In [18]:
```
start = time.time()
y_predxgb = modelxgb.predict(x_test)
end = time.time()
print("Tiempo en predicción:", end - start, "[s]")
```

```
Tiempo en predicción: 0.06588602066040039 [s]
```

In [19]:
```
confusion_matrix(y_test, y_predxgb)
```

Out[19]:
```
array([[9591, 8812],
       [1238, 2867]], dtype=int64)
```

In [20]:
```
print(classification_report(y_test, y_predxgb))
```

```
              precision    recall  f1-score   support

           0       0.89      0.52      0.66     18403
           1       0.25      0.70      0.36      4105

    accuracy                           0.55     22508
   macro avg       0.57      0.61      0.51     22508
weighted avg       0.77      0.55      0.60     22508
```

2.3.- Conclusión

Con los cambios realizados en ambos modelos, es posible apreciar que el rendimiento ha cambiado considerablemente. Teniendo ambos modelos un comportamiento similar, si bien la clasificación de la clase '0' o 'No Atraso' disminuyó su rendimiento (esperable por el balanceo de clases), la clasificación de la clase '1' o 'Atraso' aumento considerablemente.

En consecuencia, la cantidad de falsos negativos disminuyeron considerablemente, y por el contrario, los falsos positivos aumentaron. Pero, **¿Se puede determinar esta situación como mejor?**. Para la resolución del problema de negocio, se entenderá como positivo el aumento de falsos positivos (y por consecuencia la disminución de falsos negativos), ya que, es preferible determinar que un avión llegará atrasado y que este **no** llegue atrasado, que, determinar que un avión no llegará atrasado y si lo haga. De esta manera, se pueden tener consideraciones a la hora de que un vuelo llegue o salga del aeropuerto con un mejor resultado.

Finalmente, se considera que el mejor modelo (y el seleccionado) es el de **Regresión Logística**, el cual tiene un mejor rendimiento que el de XGBoost, y, con un tiempo de procesamiento mucho menor.