Taller semana 6: XGBoost

Semana 6 - Taller sumativo - Estimación de incumplimiento de clientes bancarios

Profesor: *Fernando Lozano* - **Autor Notebook:** *Daniel Felipe López*

Introducción

Descripción

Este *jupyter notebook* contiene el material necesario para el desarrollo del Taller de la Semana 6 del curso *MLS: Machine learning supervisado*. En esta tarea usted verá la solución a un problema de clasificación con la técnica de métodos de combinacion de modelos Xgboost.

Objetivos de Aprendizaje

Aplicar técnicas de métodos de combinacion de modelos para resolver problemas de clasificación.

Teoría

XGBoost

XGBoost es un algoritmo de aprendizaje automático, que combina varios modelos más débiles para formar un modelo más fuerte y preciso. Para esto, XGBoost utiliza árboles de decisión como modelos base, usa regularización y optimización de la construcción de arboles. Un par de parámetros importantes son:

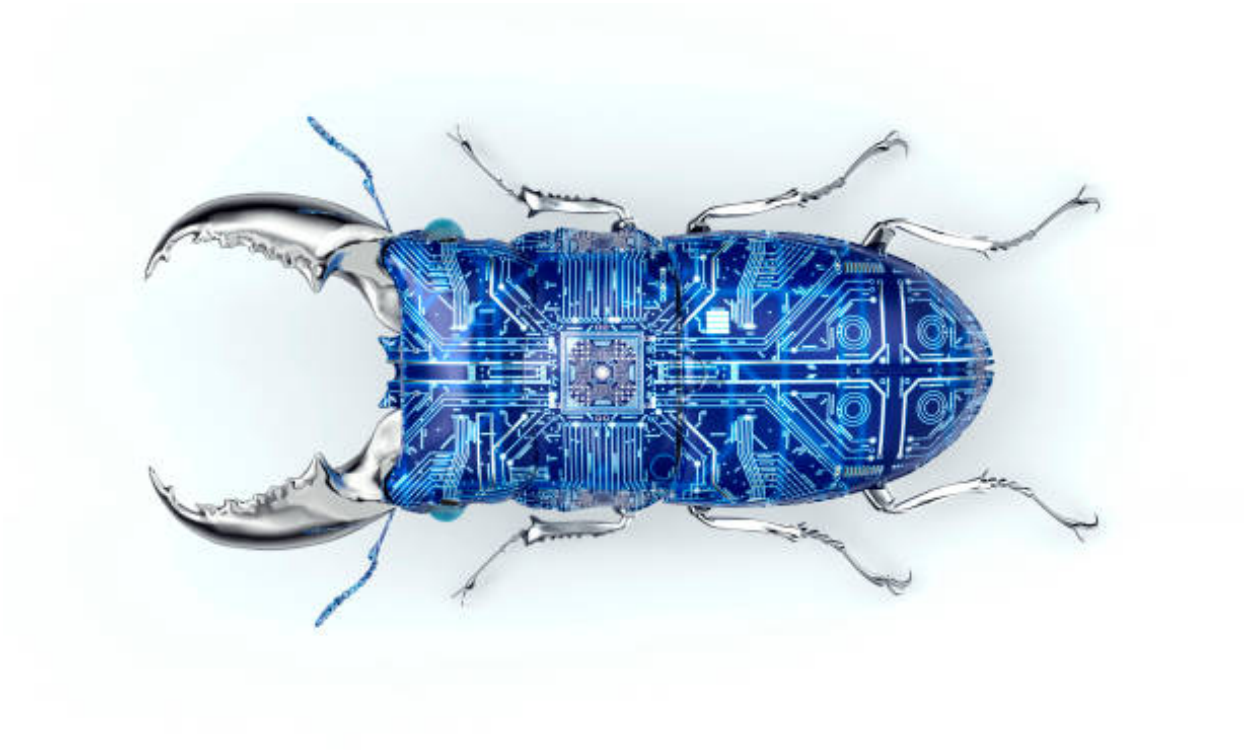
- El parámetro max_depth en XGBoost controla la profundidad máxima de cada árbol de decisión. Limitar la profundidad ayuda a prevenir el sobreajuste al restringir la complejidad del árbol. El parámetro es un entero con rango entre [1, ∞] y el valor por

defecto es 6.

- El parámetro n_estimators especifica el número de árbole1s que se construirán en el ensamblaje. Cada árbol se construye secuencialmente, y un mayor número de estimadores permite un modelo más preciso, ya que se agregan más estimaciones al ensamblaje, aunque también aumenta la carga computacional. El rango posible es entero entre [0, ∞] y el valor por defecto es 100.
- El parámetro learning_rate controla el encogimiento de los pesos de los datos de los nuevos modelos para hacer el modelo más conservativo. El rango es de [0,1] y el valor por defecto en XGBoost es 0.3.
- El parámetro reg_lambda controla la regularizacion L2 que usa el modelo. Aumentar el valor hace el modelo más conservativo. El rango posible es de [0, ∞] y el valor por defecto es 1.

Problema y conjunto de datos

El problema consiste en predecir si un software va a tener fallas, basado en las características de Maccabe y Halstead del código. El conjunto de datos fue extraido de https://kaggle.com/competitions/playground-series-s3e23 (https://kaggle.com/competitions/playground-series-s3e23). Para ver una descripcción detallada e información sobre las variables, ver aquí (https://storage.googleapis.com/kagglesdsdata/datasets/80237/186575/about%20JM1%20D
X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F20231101%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20231101T033659Z&X-Goog-Expires=259200&X-Goog-SignedHeaders=host&X-Goog-Signature=30a251e7e6299919c196b0547ecefc93e8e4d382abee6da098d63ef319812189bc4a
en especial, la sección sobre notas sobre Mccabe/Halstead.



Citación: Walter Reade, Ashley Chow. (2023). Binary Classification with a Software Defects Dataset. Kaggle. https://kaggle.com/competitions/playground-series-s3e23 (https://kaggle.com/competitions/playground-series-s3e23)

Metodología

En éste cuaderno encontrará un ejercicio práctico paso a paso. Realice la solución a cada uno de los puntos propuestos y explique claramente su procedimiento con comentarios en el código y celdas tipo Markdown. Para esto, complete las céldas de código marcadas con el siguiente comentario:

```
# ================================================
# COMPLETAR ======================================
#

# ================================================
```

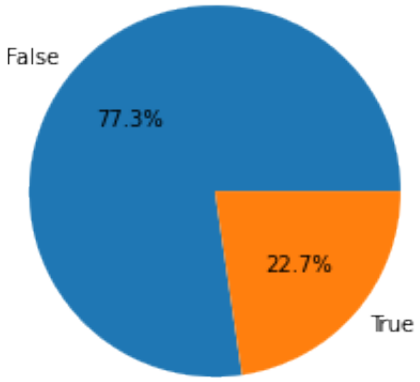
In [1]:
```
#Importar librerias necesarias
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 100);
import seaborn as sns
import time
from random import random, seed
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from scipy.stats import uniform, randint
from sklearn.preprocessing import MinMaxScaler, RobustScaler, LabelEnc
from random import random, seed
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV,
from sklearn.metrics import confusion_matrix, accuracy_score , average
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils.class_weight import compute_class_weight
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
from xgboost import plot_importance

import warnings # Ignorar las warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
#Leer la base de datos
df = pd.read_csv("train.csv").reset_index(drop=True)
```

In [3]:
```
#Mapa para transformar la variable objetivo
target_map={
    False:0,
    True: 1
}

# Separación de variable objetivo y variables explicativas
X = df.loc[:, df.columns != 'defects']
y = df['defects']

# Aplicar el reemplazo
y_numeric = np.array(y.replace(target_map))
```

In [4]:
```
#Observar balance de clases

#Contar y normalizar
dfsizes = y.value_counts()/y.shape[0]
#Crear figura
fig, ax = plt.subplots()
#Pintar
ax.pie(list(dfsizes),labels=list(dfsizes.index) ,autopct='%1.1f%%')
```

Out[4]:
```
([<matplotlib.patches.Wedge at 0x7fab2558af50>,
  <matplotlib.patches.Wedge at 0x7fab2559c690>],
 [Text(-0.8327451150636926, 0.7187040930296401, 'False'),
  Text(0.8327450814187249, -0.7187041320132517, 'True')],
 [Text(-0.4542246082165596, 0.39202041437980367, '77.3%'),
  Text(0.454224589864759, -0.3920204356435918, '22.7%')])
```



Modelo XGBoost básico

In [5]:
```
# Inicializar el modelo xgboost
model = xgb.XGBClassifier()
# Inicializar el instrumento de cross validation
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=52)
# Establecer métricas de evaluación
scoring = ['accuracy','precision', 'recall', 'f1']
# Realizar scores
scores = cross_validate(model, X, y, cv=kfold, scoring=scoring, verbos
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent wo
rkers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   13.3s
[Parallel(n_jobs=-1)]: Done   3 out of   5 | elapsed:   25.4s remaini
ng:   16.9s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:   37.3s finishe
d
```

In [6]:
```
display(pd.DataFrame(scores))
print("Accuracy: %.2f%% (%.2f%%)" % (scores['test_accuracy'].mean()*10
```

	fit_time	score_time	test_accuracy	test_precision	test_recall	test_f1
0	12.267476	0.074336	0.812264	0.638947	0.394754	0.488008
1	12.189105	0.070132	0.811133	0.646476	0.367873	0.468914
2	11.940355	0.071295	0.808775	0.630758	0.376978	0.471913
3	11.905601	0.074036	0.811026	0.637276	0.385516	0.480411
4	11.830557	0.071709	0.810879	0.638106	0.382614	0.478385

```
Accuracy: 81.08% (0.11%)
```

Ejercicio: Mejora del modelo básico

- El objetivo de este ejercicio es mejorar el modelo de XGBoost propuesto anteriormente. Para esto, implemente una búsqueda aleatoria o una búsqueda por red para encontrar mejores parámetros. Utilice el objeto kfold anterior para utilizar los mismos datos en el entrenamiento de los modelos (Máximo 100 fits, usando n_jobs = -1 para paralelización).
- Compare el mejor modelo con el modelo inicial en cuanto a estadísticas de clasificación y tiempo de ejecución. De sus conclusiones al respecto, relacionado con el papel de cada parámetro en su búsqueda.
- Implemente costos respecto a la presencia de las clases en el conjunto de datos para el mejor modelo sintonizado y compare con el mismo modelo sin costos. De sus conclusiones.

Búsqueda

A continuación, encontrará una función que le ayudará a visualizar los resultados de cada modelo incluyendo los parámetros usados.

In [7]:
```
def report_best_scores(results, n_top=3):
    # Esta función espera una instancia de resultados de búsqueda de c
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_accuracy'] == i
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".form
                  results['mean_test_accuracy'][candidate],
                  results['std_test_accuracy'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate
            print("")
    # Retorna los parámetros del mejor modelo basado en accuracy.
    return list(results.sort_values("rank_test_accuracy")['params'])[0

def see_results(results):
    # Esta función espera una instancia de pandas dataframe de los res
    display(results[results.columns.drop(list(results.filter(regex='sp
```

In [13]:
```
# ================================================
# COMPLETAR ======================================
#

# Definir parámetros (una forma de hacer esto es con las funciones uni
```

```
# Inicializar la búsqueda aleatoria con el kfold definido anteriorment

# Realizar el fit
# your code here

model = xgb.XGBClassifier()

#Búsqueda del mejor modelo en función del número de estimadores y maxi
#Generar parámetros para entrenar varios modelos.
param_grid = dict(max_depth=range(1, 20, 4),
                  n_estimators = range(5, 100, 10),
                  )

#Separación de datos en k-cross validation
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=52)

#Definición de los puntajes
scoring = ['accuracy','precision', 'recall', 'f1']

# La función gridsearch se usará para entrenar según los parámetros. E
search = GridSearchCV(model, param_grid, scoring=scoring, refit='f1',
search.fit(X, y)

# Resumir resultados de cada modelo y extraer el mejor
print("Mejor modelo para función de error: %f using %s" % (search.best
# ========================================================
resultados = pd.DataFrame(search.cv_results_)
see_results(resultados)
best_model_params = report_best_scores(resultados)
best_model_params
```

```
Mejor modelo para función de error: 0.505915 using {'max_depth': 1, '
n_estimators': 5}
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_
11	1.776125	0.017340	0.035819	0.002295	5	
12	2.743715	0.026122	0.038749	0.000600	5	
10	0.709873	0.005533	0.031452	0.000325	5	
13	3.648374	0.044632	0.041849	0.001979	5	

Comparación con el modelo original

In [14]:
```
# ====================================================
# COMPLETAR ==========================================
#

# Para usar un diccionario como parámetro de una función, es posible u
# your code here

model = xgb.XGBClassifier(**best_model_params)

# Inicializar el instrumento de cross validation
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=52)
# Establecer métricas de evaluación
scoring = ['accuracy','precision', 'recall', 'f1']
# Realizar scores
scores = cross_validate(model, X, y, cv=kfold, scoring=scoring, verbos


display(pd.DataFrame(scores))
print("Accuracy: %.2f%% (%.2f%%)" % (scores['test_accuracy'].mean()*10
# ====================================================
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent wo
rkers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    2.6s
[Parallel(n_jobs=-1)]: Done   3 out of   5 | elapsed:    4.3s remaini
ng:    2.9s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    6.1s finishe
d
```

	fit_time	score_time	test_accuracy	test_precision	test_recall	test_f1
0	1.762436	0.034616	0.814376	0.647579	0.397139	0.492341
1	1.760521	0.034521	0.816735	0.664677	0.386300	0.488621
2	1.709777	0.033544	0.813099	0.649207	0.381530	0.480612
3	1.687943	0.036831	0.813728	0.643281	0.399610	0.492978
4	1.728581	0.034198	0.816136	0.658076	0.393020	0.492128

```
Accuracy: 81.48% (0.14%)
```

YOUR ANSWER HERE