UNIVERSIDAD DE GRANADA

PRÁCTICA 1

Sistemas Inteligentes para la Gestión en la Empresa

Máster Universitario en Ingeniería Informática

Autor

Pablo Valenzuela Álvarez (pvalenzuela@correo.ugr.es)

ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

ÍNDICE

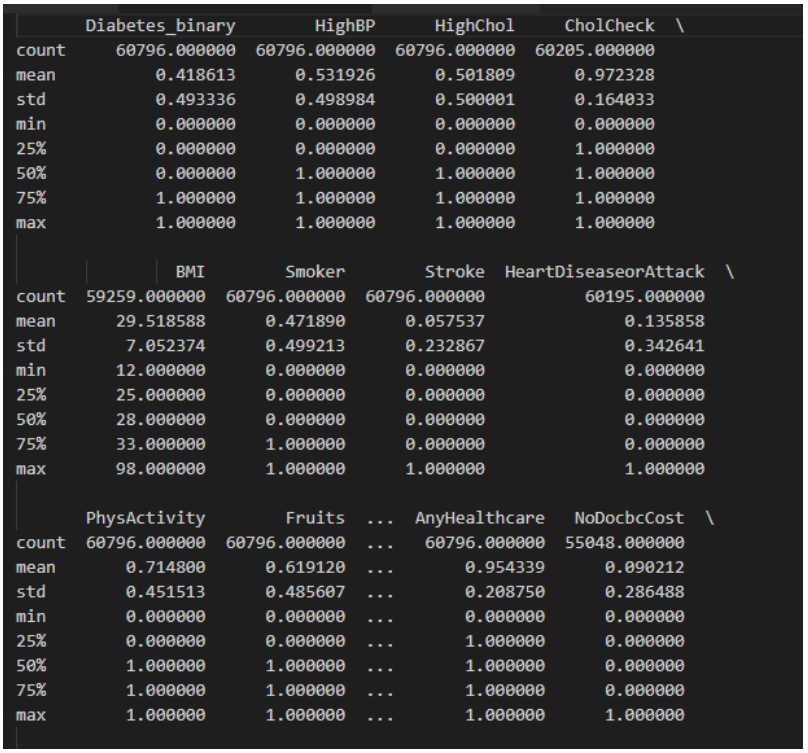
1. Introducción

El objetivo de esta práctica es realizar un preprocesamiento de datos y crear modelos predictivos usando nuestros datos preprocesados. Comenzaremos analizando un conjunto de datos, donde tendremos que identificar posibles irregularidades y problemas y resolverlas. Para ello deberemos aplicar distintas técnicas para “limpiar” los datos y dejarlos preparados para su posterior análisis predictivo. El conjunto de datos que vamos a usar es un conjunto modificado del problema *Diabetes Healht Indicator Dataset* [1] disponible en Kaggle.

2. Preprocesado de datos

2.1. Exploración y análisis de datos

Una descripción de los datos (ver figura 1) nos indica que bastantes de las características contienen solo los valores “0/1” que pueden corresponderse a respuestas “Si/No”. Si no dirigimos al enlace [1], podemos confirmar esta afirmación.



	Diabetes_binary	HighBP	HighChol	CholCheck	\
count	60796.000000	60796.000000	60796.000000	60205.000000	
mean	0.418613	0.531926	0.501809	0.972328	
std	0.493336	0.498984	0.500001	0.164033	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	1.000000	
50%	0.000000	1.000000	1.000000	1.000000	
75%	1.000000	1.000000	1.000000	1.000000	
max	1.000000	1.000000	1.000000	1.000000	

	BMI	Smoker	Stroke	HeartDiseaseorAttack	\
count	59259.000000	60796.000000	60796.000000	60195.000000	
mean	29.518588	0.471890	0.057537	0.135858	
std	7.052374	0.499213	0.232867	0.342641	
min	12.000000	0.000000	0.000000	0.000000	
25%	25.000000	0.000000	0.000000	0.000000	
50%	28.000000	0.000000	0.000000	0.000000	
75%	33.000000	1.000000	0.000000	0.000000	
max	98.000000	1.000000	1.000000	1.000000	

	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	\
count	60796.000000	60796.000000	...	60796.000000	55048.000000	
mean	0.714800	0.619120	...	0.954339	0.090212	
std	0.451513	0.485607	...	0.208750	0.286488	
min	0.000000	0.000000	...	0.000000	0.000000	
25%	0.000000	0.000000	...	1.000000	0.000000	
50%	1.000000	1.000000	...	1.000000	0.000000	
75%	1.000000	1.000000	...	1.000000	0.000000	
max	1.000000	1.000000	...	1.000000	1.000000	

Figura 1. Descripción de los datos.

El dataset guarda estos valores como enteros (ver figura 2) , así que debemos guardarlos en una variable para prevenir posibles errores a la hora de imputar valores perdidos.

```
[8 rows x 22 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60796 entries, 0 to 60795
Data columns (total 22 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Diabetes_binary       60796 non-null  int64
 1   HighBP                60796 non-null  int64
 2   HighChol              60796 non-null  int64
 3   CholCheck             60205 non-null  float64
 4   BMI                   59259 non-null  float64
 5   Smoker                60796 non-null  int64
 6   Stroke                60796 non-null  int64
 7   HeartDiseaseorAttack  60195 non-null  float64
 8   PhysActivity          60796 non-null  int64
 9   Fruits                60796 non-null  int64
 10  Veggies               60796 non-null  int64
 11  HvyAlcoholConsump     60796 non-null  int64
 12  AnyHealthcare         60796 non-null  int64
 13  NoDocbcCost           55048 non-null  float64
 14  GenHlth               60796 non-null  int64
 15  MentHlth              60796 non-null  int64
 16  PhysHlth              60796 non-null  int64
 17  DiffWalk              60796 non-null  int64
 18  Sex                   60796 non-null  int64
 19  Age                   60796 non-null  int64
 20  Education             60796 non-null  int64
 21  Income                60128 non-null  float64
dtypes: float64(5), int64(17)
memory usage: 10.2 MB
None
```

Figura 2. Tipo de datos de las características.

2.2. Imputación de valores perdidos

Nuestros datos contienen valores perdidos para ciertas variables (ver figura 3). Para solucionar este problema vamos a usar dos técnicas de imputación de valores perdidos. Es importante rellenar los valores que faltan para preservar los datos y generar un mejor modelo para el clasificador que lo vaya a usar posteriormente.

```
Diabetes_binary            0
HighBP                     0
HighChol                   0
CholCheck                591
BMI                     1537
Smoker                     0
Stroke                     0
HeartDiseaseorAttack     601
PhysActivity               0
Fruits                     0
Veggies                    0
HvyAlcoholConsump          0
AnyHealthcare              0
NoDocbcCost             5748
GenHlth                    0
MentHlth                   0
PhysHlth                   0
DiffWalk                   0
Sex                        0
Age                        0
Education                  0
Income                   668
dtype: int64
```

Figura 3. Valores perdidos en los datos.

Los valores *BMI* e *Income* contienen valores numéricos, mientras que las demás son categóricos. Esto es importante considerarlo a la hora de imputar valores perdidos, sobre todo cuando tenemos variables categóricas, y tienen que ser tratadas distinto que las numéricas.

En este caso, se ha optado por tratar las variables numéricas con *IterativeImputer* [2] (que usa un modelo de regresión). Mientras que las variables categóricas se han tratado con *SimpleImputer* [3] con las estrategia del más frecuente o moda (ver figura 4).

```python
imputer = IterativeImputer(max_iter=10, random_state=12345)
imputer2 = SimpleImputer(strategy='most_frequent')

datos_imp = datos.copy()

cols_numericas = ['BMI','Income']
datos_imp[cols_numericas] = imputer.fit_transform(datos_imp[cols_numericas])

cols_categoricas = ['CholCheck', 'HeartDiseaseorAttack', 'NoDocbcCost']
imputer2.fit(datos_imp[cols_categoricas])

datos_imp[cols_categoricas] = imputer2.fit_transform(datos_imp[cols_categoricas])
```

Figura 4. Tratamiento de valores perdidos.

2.3. Valores anómalos

El diagrama de cajas siguiente (ver figura 5) muestra que nuestro conjunto contiene valores anómalos para las variables numéricas. Normalmente, estos valores corresponden a casos únicos o errores en la toma de datos, por lo que pueden ser ignorados en la mayoría de los casos.

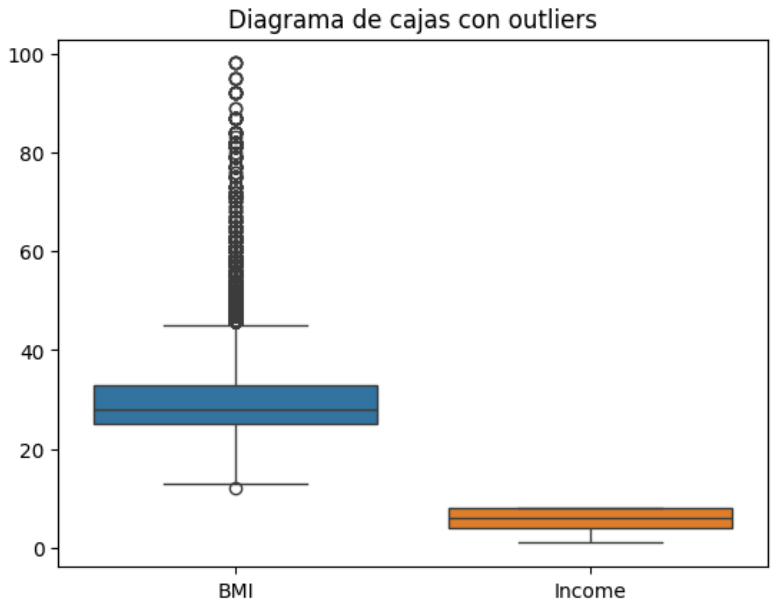


Figura 5. Diagrama de cajas de los valores anómalos.

Si contamos los valores anómalos, nos da un resultado de 2867 (ver figura 6). No es mucha cantidad sabiendo que nuestro conjunto contiene cerca de 60000 datos, por lo que la estrategia a seguir será la de su eliminación (ver figura 7).

```
Q1 = datos_out['BMI'].quantile(0.25)
Q3 = datos_out['BMI'].quantile(0.75)
IQR = Q3 - Q1

lower = Q1 - (1.2 * IQR)
upper = Q3 + (1.2 * IQR)

outliers = datos_out[(datos_out['BMI'] < lower) | (datos_out['BMI'] > upper)]

print(len(outliers))
```

✓ 0.0s

```
2867
```

Figura 6. Código para identificar los valores anómalos.

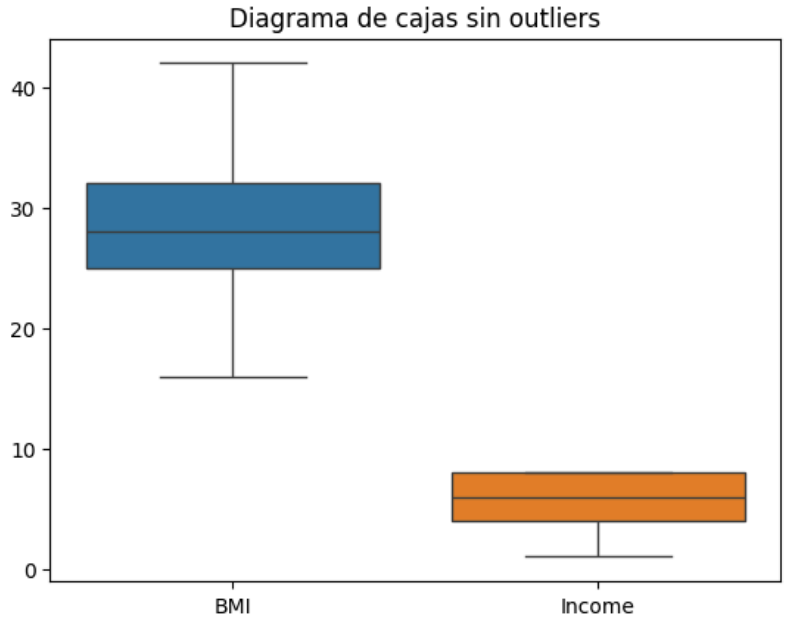


Figura 7. Diagrama de cajas tras la eliminación de los valores anómalos.

2.4. Normalización de los datos numéricos

Es necesario normalizar los datos numéricos ya que facilita el entrenamiento de los modelos de aprendizaje, permite una interpretación más clara de los coeficientes y mitiga el impacto de valores atípicos. En el código se ha usado la librería *MinMaxScaler* [4] para normalizar los valores de las variables numéricas entre 0 y 1.

```
datos_norm = datos.copy()
cols_numericas = ['BMI', 'Age', 'Income']
normalizar = datos_norm[cols_numericas]

scaler = MinMaxScaler()
datos_norm[cols_numericas] = np.round(scaler.fit_transform(normalizar), decimals=4)
```

Figura 8. Código para normalizar los datos numéricos.

2.5. Selección de características

Se puede hacer una reducción de características investigando la correlación entre las características del conjunto. Los valores pueden oscilar entre 1 (correlación positiva fuerte) y -1 (correlación negativa fuerte), esto significa que las características cercanas a 1 y -1 pueden ser eliminadas ya que representan comportamientos similares.

Si observamos la siguiente figura (ver figura 9), podemos observar que no hay correlaciones lo suficientemente fuertes como para considerar la eliminación de alguna de ellas, por lo que no vamos a hacer reducción de características.

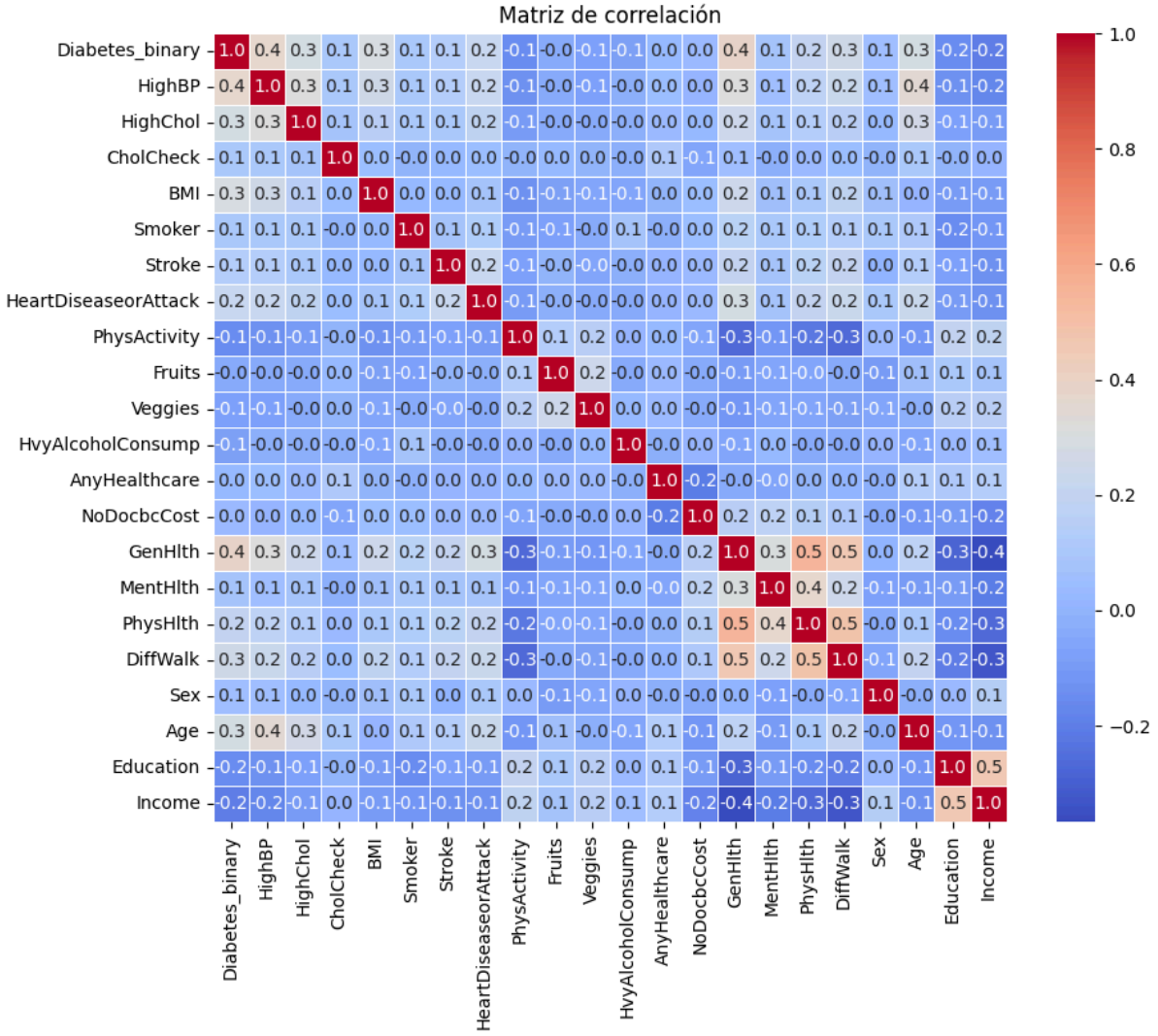


Figura 9. Matriz de correlación lineal de los datos del conjunto.

2.6 Balanceo de datos

El balanceo de datos es un proceso en el que se ajusta la distribución de clases para que estén representadas de forma equitativa. Con esta técnica ayudamos a mejorar el rendimiento del modelo, evitar el sesgo en la predicción que se aplicaba en los conjuntos no balanceados, y por consiguiente reducir el sobreajuste.

Nuestro conjunto de datos tiene una distribución de clases no del todo balanceada (ver figura 10), por lo que tenemos que aplicar esta técnica.

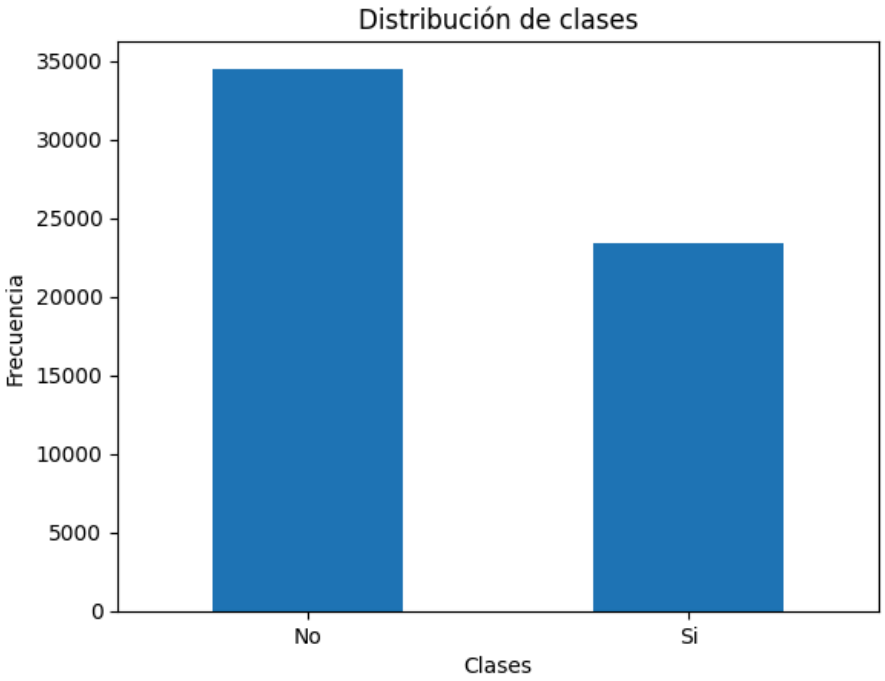


Figura 10. Datos no balanceados.

Nuestro código utiliza la librería *SMOTENC* [5], que es una variante de la técnica de oversampling SMOTE que considera las variables categóricas. En la siguiente figura (ver figura 11), la variable *indices* contiene los índices de las columnas con datos categóricos, que tiene que ser pasada como argumento en el constructor de *SMOTENC*.

```
from imblearn.over_sampling import SMOTENC

X = datos_bal.drop(columns=['Diabetes_binary'], axis=1)
y = datos_bal['Diabetes_binary']

indices = [datos.columns.get_loc(col) for col in cols]

smote = SMOTENC(categorical_features=indices, random_state=12345)
X_bal, y_bal = smote.fit_resample(X, y)
```

Figura 11. Código para balancear los datos.

En la siguiente figura (ver figura 12) se puede ver la distribución de clases una vez ya hecho el balanceo de datos.

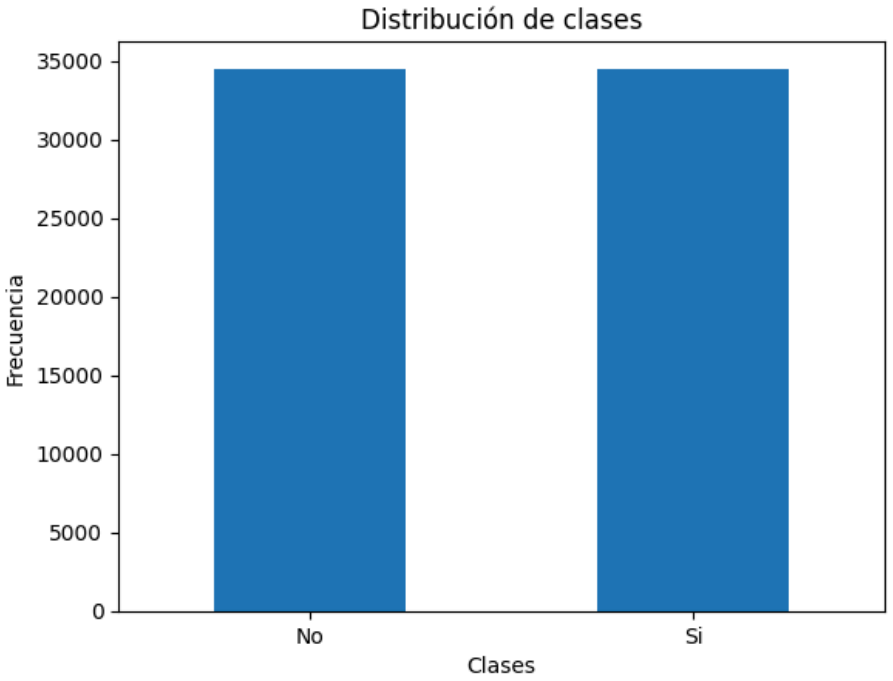


Figura 12. Datos balanceados.

3. Clasificadores

Se han usado los clasificadores *RandomForestClassifier* [6] y *GradientBoostingClassifier* [7] en nuestro problema.

RandomForestClassifier es un algoritmo de aprendizaje automático que usa un conjunto de árboles de decisión entrenados con muestras aleatorias del conjunto de datos para predecir un valor. Y *GradientBoostingClassifier* construye un modelo predictivo en forma de un conjunto de datos de decisión, donde cada árbol se ajusta para corregir los errores del anterior.

El código se ha ajustado (ver figura 13) para realizar una búsqueda randomizada sobre un conjunto de hiperparámetros, y así obtener el mejor modelo para cada clasificador.

```
hyper_rfc = {
    'n_estimators': [50,100,200],
    'criterion':['gini','entropy'],
    'max_depth':[3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False],
}
hyper_gbc = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

from sklearn.model_selection import RandomizedSearchCV
rs_rfc = RandomizedSearchCV(
    rfc,
    param_distributions=hyper_rfc,
    random_state=0,
    n_jobs=-1,
    cv=5,
    scoring='accuracy',
    return_train_score=True
)
rs_gbc = RandomizedSearchCV(
    gbc,
    param_distributions=hyper_gbc,
    random_state=0,
    n_jobs=-1,
    cv=5,
    scoring='accuracy',
    return_train_score=True
)
```

Figura 13. Ajuste de hiperparametros.

El resultado de la búsqueda del mejor modelo para ambos clasificadores es el siguiente:

- RFC best parametros: {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_depth': 10, 'criterion': 'gini', 'bootstrap': True}
  RFC best score: 0.7771097428467947

- GBC best parametros: {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': 5, 'learning_rate': 0.1}
  GBC best score: 0.7837739949293734

Por lo tanto, el mejor modelo creado pertenece al algoritmo de *GradientBoosting*. En la siguiente sección probaremos ambos modelos con los datos de test.

3.1. Discusión de resultados

Como se ve en la siguiente figura (ver figura 14), el resultado de ejecutar el código da como mejor modelo el obtenido por el algoritmo de *GradientBoosting*.

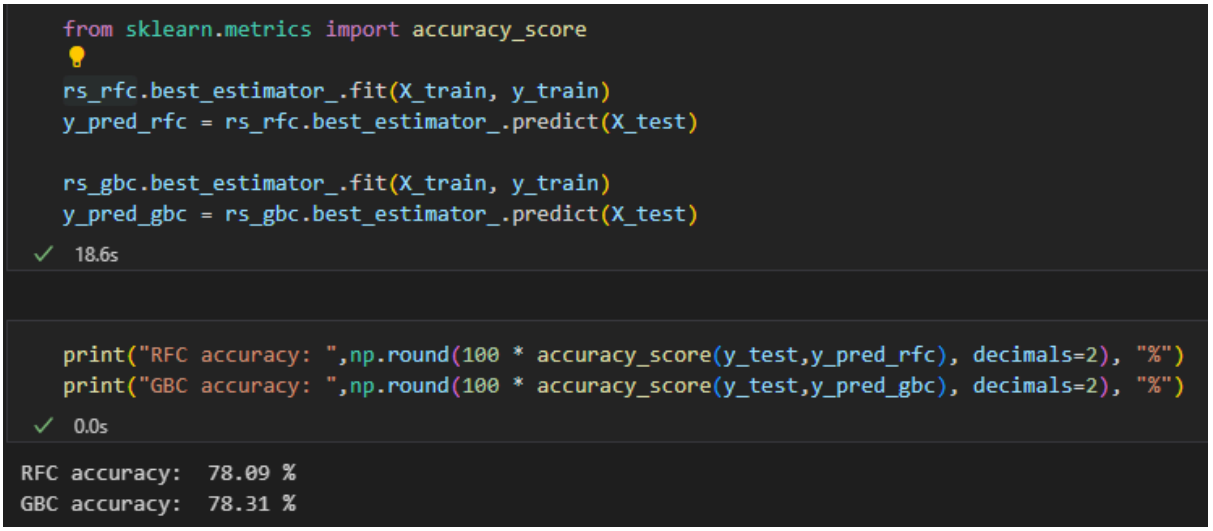


Figura 14. Resultado final de ambos clasificadores.

Esto puede ser debido al enfoque secuencial que tiene este modelo, la ponderación de errores de los ejemplos no clasificados correctamente hace que este menos propenso al sobreajuste.

Al ser solo una diferencia de tres décimas podemos concluir que ambos modelos pueden ser válidos, pero para aclararlo mejor vamos a realizar el análisis de la curva ROC (ver figura 15).

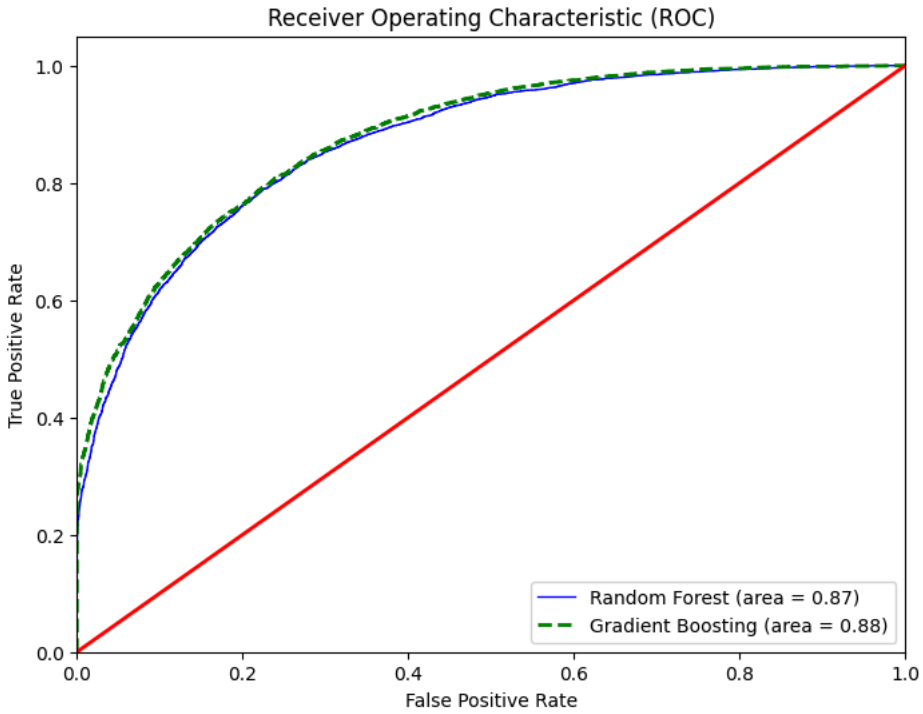


Figura 15. Curvas ROC de los clasificadores.

Ambos modelos presentan valores bastante buenos y, como se puede ver en la gráfica, son casi idénticos.

En resumen, podemos considerar ambos modelos válidos y, vistos los resultados podemos decir que tienen una capacidad predictiva bastante sólida..

4. Conclusión

En esta práctica, hemos realizado diversas técnicas de preprocesamiento de datos tales como la limpieza de datos, la imputación de valores perdidos, etc. También hemos demostrado la importancia de preparar correctamente los datos antes de construir modelos predictivos.

Además, al comparar los algoritmos de aprendizaje automático, hemos comprendido la importancia que tiene la elección del algoritmo según el problema dado.

Por último, este estudio demuestra la importancia del procesamiento de datos para obtener resultados confiables y útiles en la aplicación de técnicas de aprendizaje automático en problemas del mundo real.

5. Enlace a Colab

El enlace para acceder al cuaderno en Google Colab es el siguiente: practica1.ipynb.

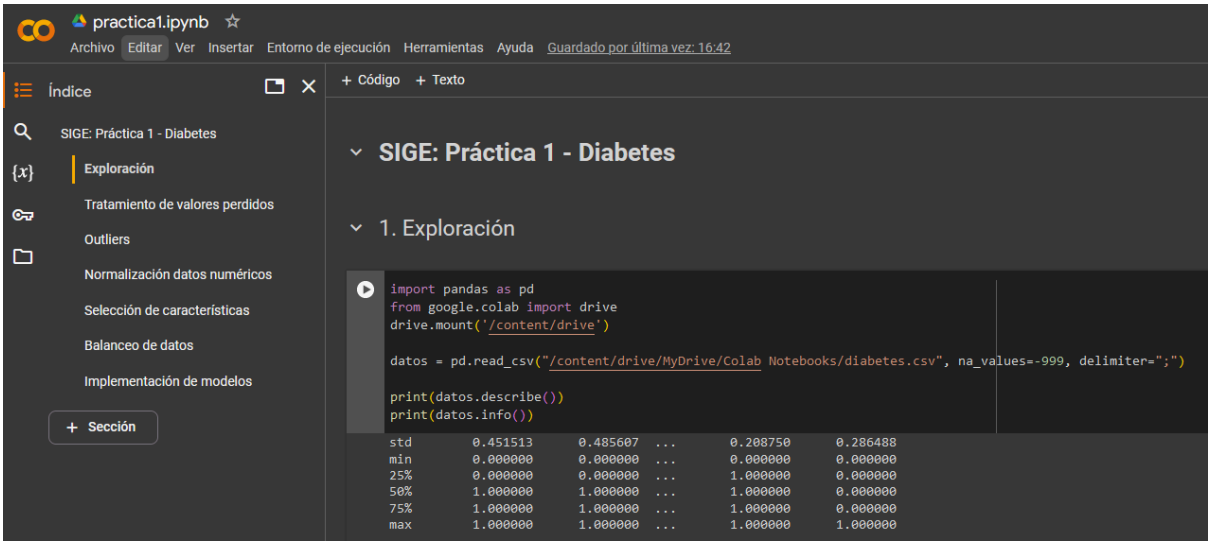


Figura 16. Imagen del cuaderno en Google Colab.

Nota: Parece que hay un problema en colab cuando intentamos ejecutar *SMOTENC* y *RandomSearch*, lo que apenas tarda 30 segundos y 2 minutos en mi ordenador personal, en colab puede demorar 4 minutos y más de 15 minutos.

Nota2: Los resultados que aparecen en esta memoria son los obtenidos en mi ordenador personal, en colab son un poco peores.

6. Bibliografía

[1]. “Diabetes Health Indicators Dataset.” *Kaggle*, https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset.

[2]. “sklearn.impute.IterativeImputer — scikit-learn 1.4.2 documentation.” *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html.

[3]. “sklearn.impute.SimpleImputer — scikit-learn 1.4.2 documentation.” *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html.

[4]. “sklearn.preprocessing.MinMaxScaler — scikit-learn 1.4.2 documentation.” *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[5]. “SMOTENC — Version 0.12.2.” *Imbalanced-learn*, https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTENC.html.

[6]. “sklearn.ensemble.RandomForestClassifier — scikit-learn 1.4.2 documentation.” *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[7]. “sklearn.ensemble.GradientBoostingClassifier — scikit-learn 1.4.2 documentation.” *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.