Estudio sobre readmisión de pacientes con diabetes

Práctica final – Análisis de Datos Clínicos

Fernando Augusto Marina Urriola
Leticia Sáez Viadero

Tabla de contenido

Leticia Sáez Viadero
Fernando A Marina Urriola

Objetivos

- Usar la herramienta WEKA o el lenguaje Python (NumPy, Pandas, scikit-learn) para construir modelos predictivos;
- Buscar y seleccionar los modelos con la mayor precisión, en nuestro caso, la medida será el 'ROC Area' (área bajo la curva ROC) usando la estrategia de validación cruzada de 10 particiones (10-CV, Cross Validation).
- La variable a predecir es: readmitted (situada en último lugar)

WEKA

Weka (Waikato Environment for Knowledge Analysis) es una plataforma de software destinada al aprendizaje automático y la minería de datos (Data Mining) escrita en lenguaje Java y desarrollada en la Universidad de Waikato (Nueva Zelanda). La plataforma Weka se caracteriza por los siguientes parámetros:

- **Disponible**: esta plataforma de software es libre gracias a la licencia pública general de GNU.
- **Adaptable**: al estar implementada en lenguaje Java, es compatible casi con cualquier plataforma.
- **Funcional**: está formada por un amplio repositorio de técnicas para preprocesamiento de datos y modelado.
- **Sencilla**: su uso es muy fácil gracias a su interfaz gráfica de usuario.

La minería de datos o Data Mining utiliza cuatro clases de tareas:

1. Clasificación.
2. Agrupamiento (clustering).
3. Regresión.
4. Reglas de asociación.

Para llevarlas a cabo, se sirve de técnicas estadísticas, algoritmos matemáticos y algoritmos de aprendizaje automático que ayudan a mejorar el rendimiento en base a la experiencia.

El aprendizaje automático o machine learning, como hemos visto en otras entradas es un subcampo del Data Science, el cual abarca el proceso de obtención del conocimiento a través de la búsqueda, análisis, visualización y despliegue de datos.

Python

Python es un lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de todo tipo. A diferencia de otros lenguajes como Java o .NET, se trata de un lenguaje interpretado, es decir, que no es necesario compilarlo para ejecutar las aplicaciones escritas en Python, sino que se ejecutan directamente por el ordenador utilizando un programa denominado interpretador, por lo que no es necesario "traducirlo" a lenguaje máquina.

Python es un lenguaje sencillo de leer y escribir debido a su alta similitud con el lenguaje humano. Además, se trata de un lenguaje multiplataforma de código abierto y, por lo tanto, gratuito, lo que permite desarrollar software sin límites. Con el paso del tiempo, Python ha ido ganando adeptos gracias a su sencillez y a sus amplias posibilidades, sobre todo en los últimos años, ya que facilita trabajar con inteligencia artificial, big data, machine learning y data science, entre muchos otros campos en auge.

Anotación

Se hará uso del programa 'WEKA' y asimismo el programa 'Python'.

Para la primera ronda de eliminación se hace uso del programa 'WEKA' mientras que para lo demás se utiliza 'Python'. Para esta primera ronda se utiliza 'WEKA' ya que es mucho más visual y permite ver el número de variables con valores missing y su porcentaje, los distintos valores que hay en cada atributo con el parámetro 'distinct' y ver cuántos hay de cada uno… Para la segunda ronda, y el resto de la práctica, se ha utilizado el programa 'Python'.

Leticia Sáez Viadero
Fernando A Marina Urriola

Nivel básico

1.Conversión de fichero

Pasos para convertir de '.csv' a '.arff'.

1. Abrir el programa WEKA.

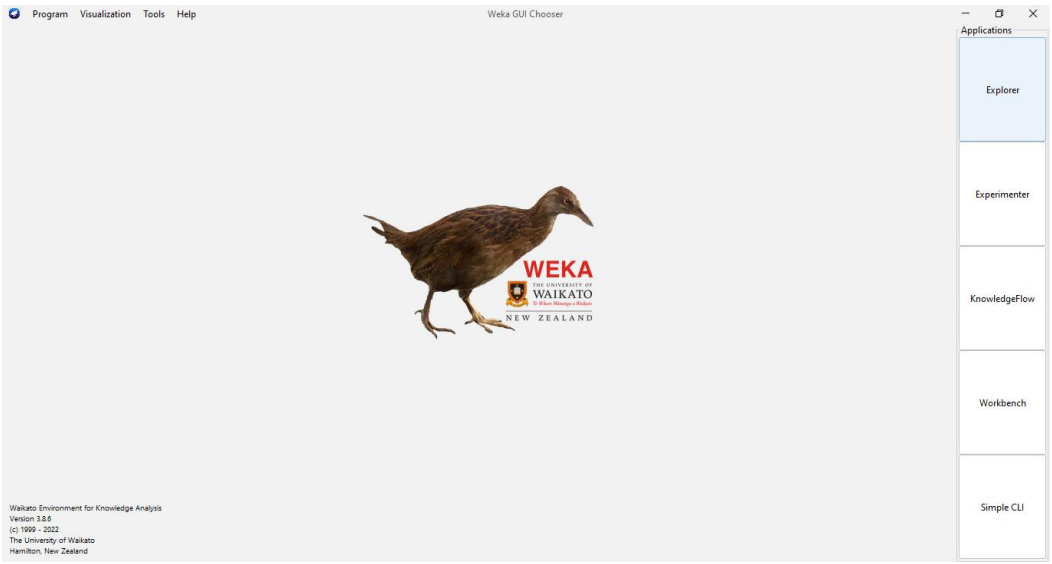


Ilustración 1. Aplicación WEKA

2. Abrir la ARF-Viewer en la ventana "Tools" en el menú y selecciona "ArffViewer".

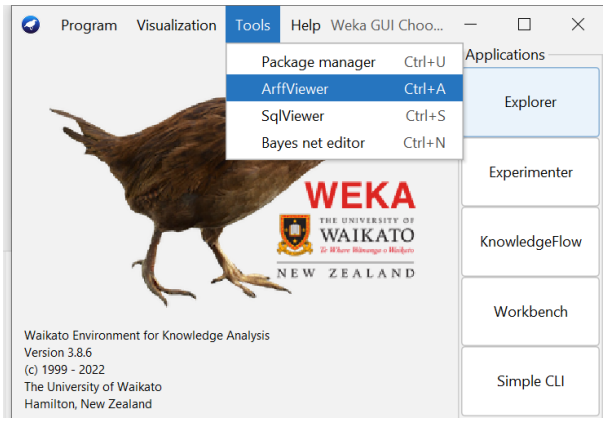


Ilustración 2. Ventana "Tools".

3. Se presentará una Ventana ARFF-Viewer vacía.

Ilustración 3. Ventana vacia "ARFF-Viewer".

4. Abre el archivo CSV en ARFF-Viewer pulsando en el menú “File” y selecciona “Open”. Navega en tu directorio de trabajo. Cambia el filtro “Files of Type:” a “CSV data files(*.csv)”. Selecciona el archivo y pulsa “Open”.

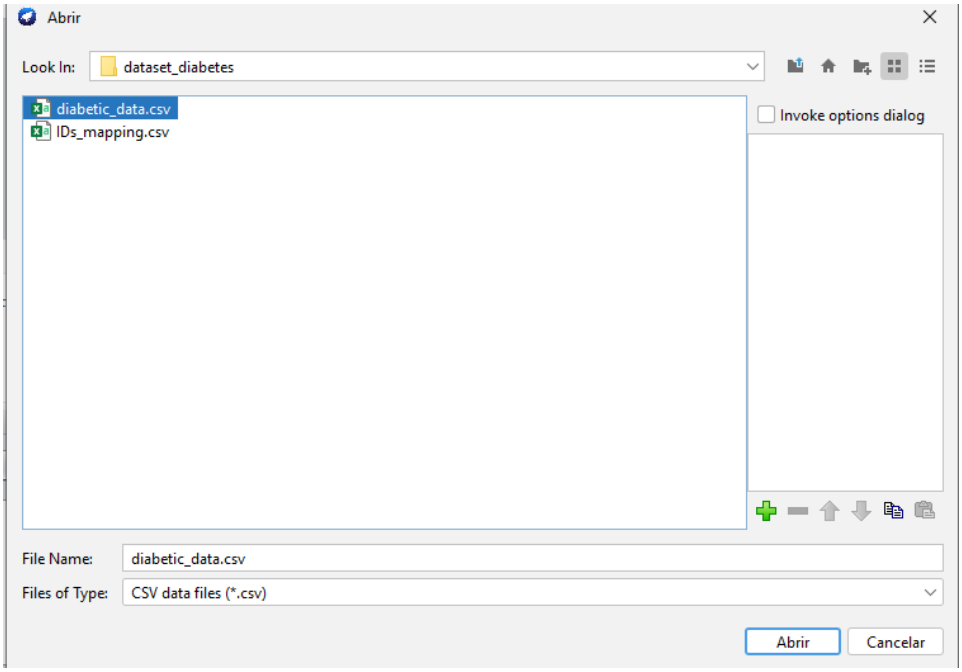
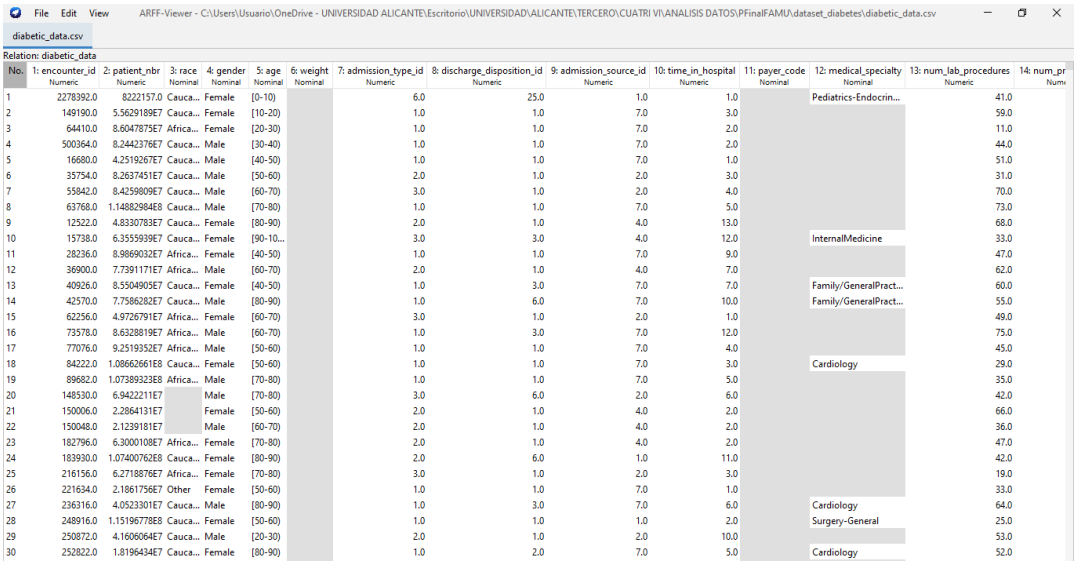


Ilustración 4. Cambio de archivo.

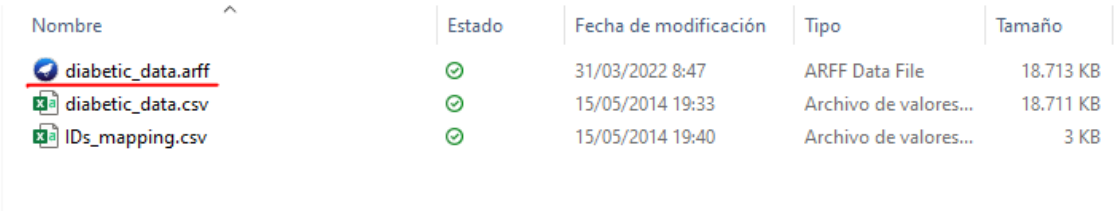
5. Se debe ver un ejemplo de tu archivo CSV cargado en un ARFF-Viewer.



No.	1: encounter_id Numeric	2: patient_nbr Numeric	3: race Nominal	4: gender Nominal	5: age Nominal	6: weight Nominal	7: admission_type_id Numeric	8: discharge_disposition_id Numeric	9: admission_source_id Numeric	10: time_in_hospital Numeric	11: payer_code Nominal	12: medical_specialty Nominal	13: num_lab_procedures Numeric	14: num_pr Num
1	2278392.0	8222157.0	Cauca...	Female	[0-10)		6.0	25.0	1.0	1.0		Pediatrics-Endocrin...	41.0	
2	149190.0	5.5629189E7	Cauca...	Female	[10-20)		1.0	1.0	7.0	3.0			59.0	
3	64410.0	8.6047875E7	Africa...	Female	[20-30)		1.0	1.0	7.0	2.0			11.0	
4	500364.0	8.2442376E7	Cauca...	Male	[30-40)		1.0	1.0	7.0	2.0			44.0	
5	16680.0	4.2519267E7	Cauca...	Male	[40-50)		1.0	1.0	7.0	1.0			51.0	
6	35754.0	8.2637451E7	Cauca...	Male	[50-60)		2.0	1.0	2.0	3.0			31.0	
7	55842.0	8.4259809E7	Cauca...	Male	[60-70)		3.0	1.0	2.0	4.0			70.0	
8	63768.0	1.14882984E8	Cauca...	Male	[70-80)		1.0	1.0	7.0	5.0			73.0	
9	12522.0	4.8330783E7	Cauca...	Female	[80-90)		2.0	1.0	4.0	13.0			68.0	
10	15738.0	6.3555939E7	Cauca...	Female	[90-10...		3.0	3.0	4.0	12.0		InternalMedicine	33.0	
11	28236.0	8.9869032E7	Africa...	Female	[40-50)		1.0	1.0	7.0	9.0			47.0	
12	36900.0	7.7391171E7	Africa...	Male	[60-70)		2.0	1.0	4.0	7.0			62.0	
13	40926.0	8.5504905E7	Cauca...	Female	[40-50)		1.0	3.0	7.0	7.0		Family/GeneralPract...	60.0	
14	42570.0	7.7586282E7	Cauca...	Male	[80-90)		1.0	6.0	7.0	10.0		Family/GeneralPract...	55.0	
15	62256.0	4.9726791E7	Africa...	Female	[60-70)		3.0	1.0	2.0	1.0			49.0	
16	73578.0	8.6328819E7	Africa...	Male	[60-70)		1.0	3.0	7.0	12.0			75.0	
17	77076.0	9.2519352E7	Africa...	Male	[50-60)		1.0	1.0	7.0	4.0			45.0	
18	84222.0	1.08662661E8	Cauca...	Female	[50-60)		1.0	1.0	7.0	3.0		Cardiology	29.0	
19	89682.0	1.07389323E8	Africa...	Male	[70-80)		1.0	1.0	7.0	5.0			35.0	
20	148530.0	6.9422211E7		Male	[70-80)		3.0	6.0	2.0	6.0			42.0	
21	150006.0	2.2864131E7		Female	[50-60)		2.0	1.0	4.0	2.0			66.0	
22	150048.0	2.1239181E7		Male	[60-70)		2.0	1.0	4.0	2.0			36.0	
23	182796.0	6.3000108E7	Africa...	Female	[70-80)		2.0	1.0	4.0	2.0			47.0	
24	183930.0	1.07400762E8	Cauca...	Female	[80-90)		2.0	6.0	1.0	11.0			42.0	
25	216156.0	6.2718876E7	Africa...	Female	[70-80)		3.0	1.0	2.0	3.0			19.0	
26	221634.0	2.1861756E7	Other	Female	[50-60)		1.0	1.0	7.0	1.0			33.0	
27	236316.0	4.0523301E7	Cauca...	Male	[80-90)		1.0	3.0	7.0	6.0		Cardiology	64.0	
28	248916.0	1.15196778E8	Cauca...	Female	[50-60)		1.0	1.0	1.0	2.0		Surgery-General	25.0	
29	250872.0	4.1606064E7	Cauca...	Male	[20-30)		2.0	1.0	2.0	10.0			53.0	
30	252822.0	1.8196434E7	Cauca...	Female	[80-90)		1.0	2.0	7.0	5.0		Cardiology	52.0	

Ilustración 5. CSV cargado.

6. Guarda el conjunto de datos en un formato ARFF pulsando en “File” y seleccionando “Save as…”. Pon un nombre al archivo con la extensión “.arff” y pulsa en el botón de guardar.



Nombre	Estado	Fecha de modificación	Tipo	Tamaño
diabetic_data.arff	✓	31/03/2022 8:47	ARFF Data File	18.713 KB
diabetic_data.csv	✓	15/05/2014 19:33	Archivo de valores...	18.711 KB
IDs_mapping.csv	✓	15/05/2014 19:40	Archivo de valores...	3 KB

Ilustración 6. Guardar el archivo ".arff".

Se ha generado el fichero “.arff” en la ruta donde hemos elegido. Ya se puede comenzar a tratar los datos en WEKA.

2. Adaptación de los datos

Se abren los datos en WEKA.

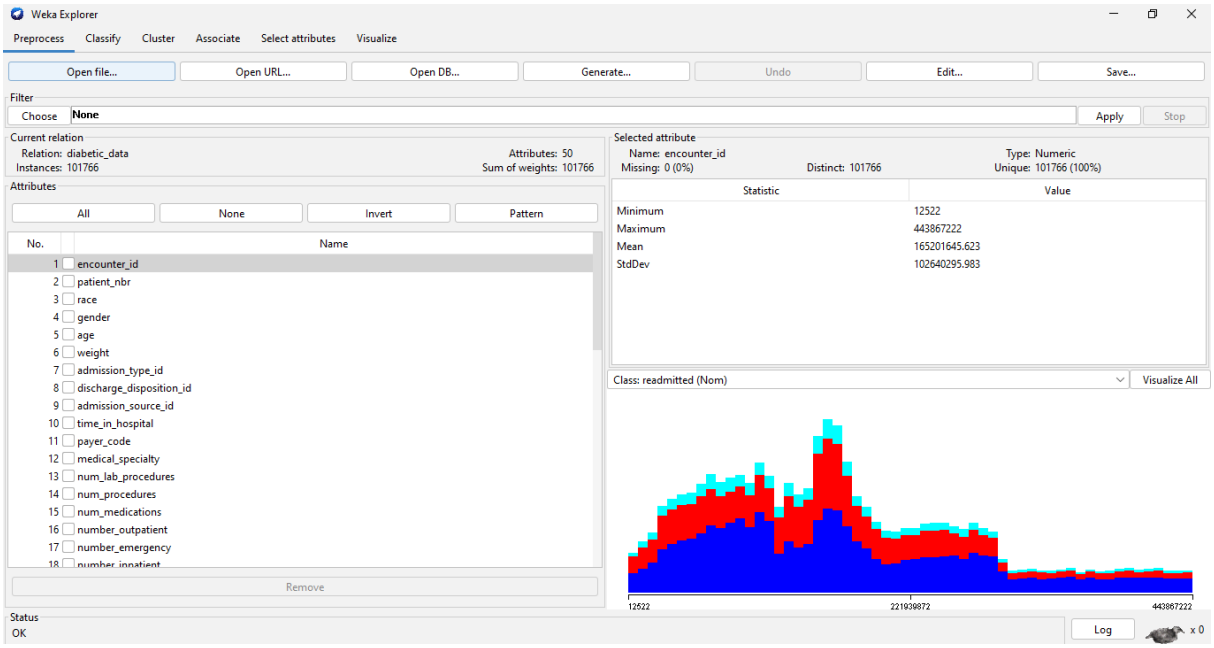


Ilustración 7. Abrir datos en WEKA.

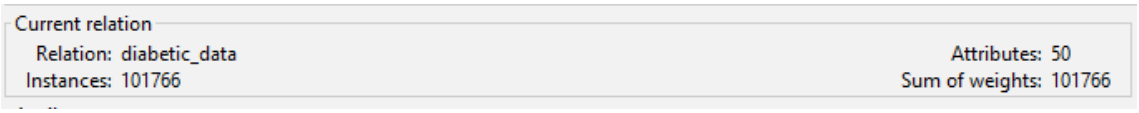


Ilustración 8. Correlación actual.

Como se puede comprobar, el archivo “diabetic_data” consta de 50 atributos y 101766 instancias.

2.1 Eliminando atributos

Se eliminarán atributos que principalmente se encuentran desequilibrados en el número de muestras por caso o aquellos que se han considerado irrelevantes para el estudio presente.

Para eliminar atributos, primero se seleccionan y se pulsa el botón de eliminar que se encuentra al final. Los atributos seleccionados se eliminarán del conjunto de datos. Después de haber preprocesado el conjunto, se guardan para construir el modelo.

Primera ronda de eliminación

En esta primera ronda se eliminarán las siguientes variables que, por ejemplo, están desequilibradas en muestras, muchas de ellas contienen más del 90% de muestras asociadas a una clase, algo que no nos sirve para clasificar, véanse los siguientes ejemplos:

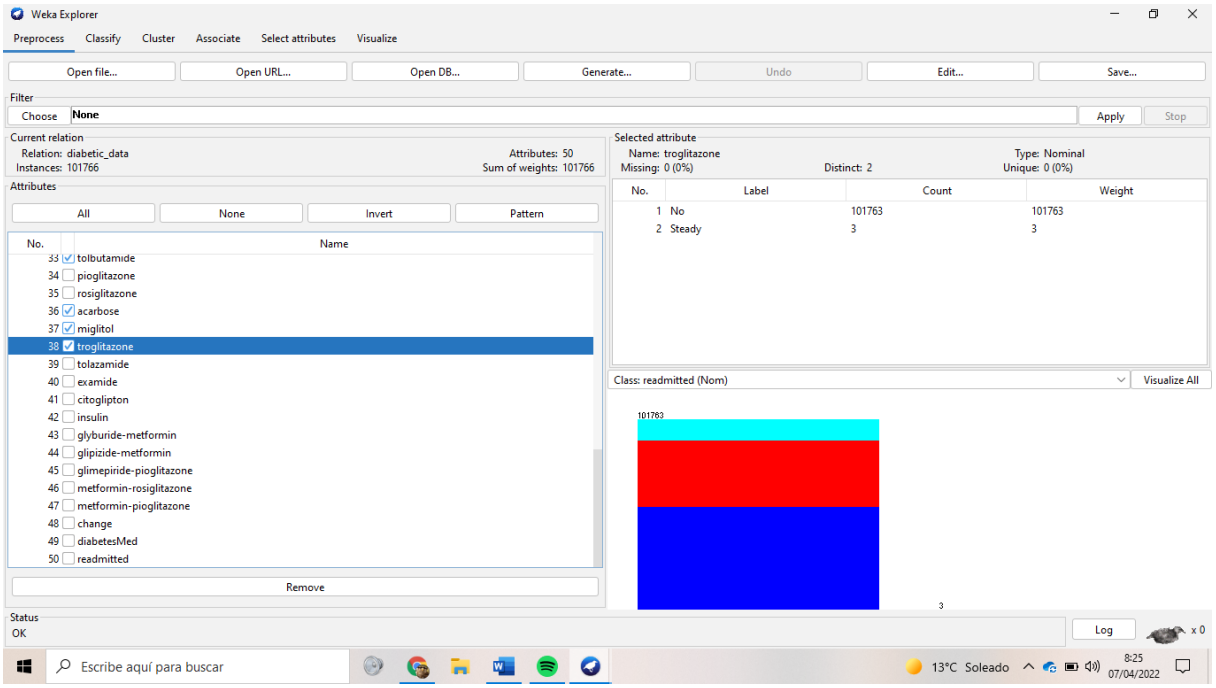


Ilustración 9. Ejemplo de atributo eliminado.

En este caso, al ser simplemente dos variables y llevar el atributo “No” el casi total de los “count”, no es relevante.

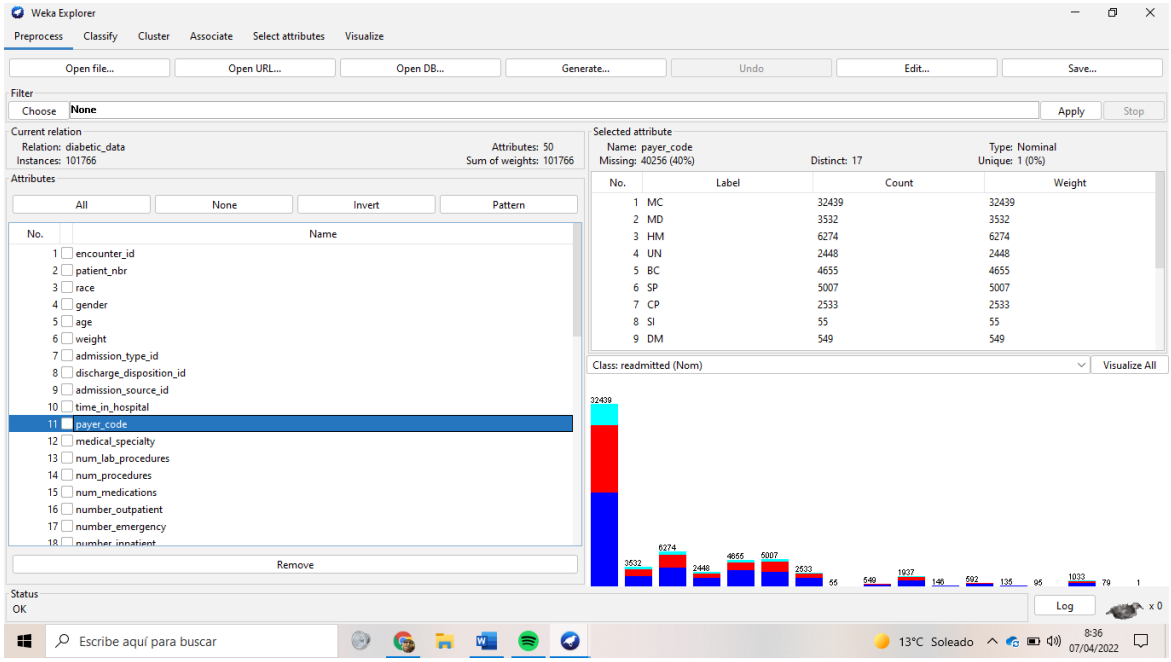


Ilustración 10. Ejemplo de atributo eliminado.

En la ilustración 10, se puede ver que, aparentemente, los datos están correctamente repartidos, pero si nos fijamos en la cabecera siguiente:

Selected attribute

Name: payer_code
Missing: 40256 (40%)
Distinct: 17
Type: Nominal
Unique: 1 (0%)

No.	Label	Count	Weight
1	MC	32439	32439
2	MD	3532	3532
3	HM	6274	6274
4	UN	2448	2448
5	BC	4655	4655
6	SP	5007	5007
7	CP	2533	2533
8	SI	55	55
9	DM	549	549

Ilustración 11. Eliminación atributo.

Se observa que el 40% de los datos está missing, lo que hace irrelevante esta variable. Por lo tanto, se eliminará esta variable y todas las que estén en una situación similar.

Se eliminarán las siguientes variables:

- *Weight*
- *encounter_id*
- *payer_code*
- *medical_speciality*
- *metformin*
- *repaglinide*
- *nateglinide*
- *chlorpropamide*
- *glimepiride*
- *acetohexamide*
- *tolbutamide*
- *acarbose*
- *miglitol*
- *troglitazone*
- *tolazamide*
- *examide*
- *citoglipton*
- *glyburide-metformin*
- *glypizide-metformin*
- *glimepiride-pioglitazone*
- *metformin-rosiglitazone*
- *metformin-pioglitazone*

Segunda ronda de eliminación

En esta segunda ronda de eliminación se hará uso de Python, ya que, esta herramienta nos permite hacer un estudio de mayor profundidad en los datos que queremos obtener.

Primero, para obtener las variables se deben realizar estas instrucciones en una libreta de Google colab o jupyter notebook:

```
[58] import pandas as pd ##para manejar los datos
     import os
     import numpy as np
     import matplotlib.pyplot as plt ##para hcaer graficop
     import seaborn as sns ##para hcaer graficos
     from sklearn.preprocessing import LabelEncoder
     from sklearn.preprocessing import StandardScaler
```

Ilustración 12. Librerías importadas.

```
[59] #conectar drive a esto
     from google.colab import drive
     #me conecto a mi drive
     drive.mount('/content/mydrive/')
```

Ilustración 13. Conexión drive y Google Colab.

```
[60] #cd
     os.chdir('/content/mydrive/MyDrive/dataset_diabetes')
```

```
#control f9 cargar todo
df = pd.read_csv('diabetic_data.csv')
```

Ilustración 14. Importación dataset.

En la ilustración 12 están importadas todas las librerías necesarias para el desarrollo de la práctica. Entre ellas se encuentran:

- Pandas (para el manejo de datos).
- Os (para interactuar con el sistema operativo).
- Numpy (para el cálculo numérico y análisis de datos).
- Matplotlib.pyplot (para hacer gráficos).
- Seaborn (para hacer gráficos).
- LabelEncoder de Sklearn.preprocessing (para codificar etiquetas).

En la ilustración 13, se ha importado la librería drive para tener conectado el drive al Google Colab; esta conexión se realiza con el comando 'drive.mount' y la ruta donde lo queremos situar ya que la posteriormente se necesitará esa ruta.

En la ilustración 14, contamos con dos celdas. La primera celda, se ha conectado con el dataset de diabetes siguiente la ruta anterior; esto se realiza con la librería 'os', con el comando 'os.chdir'.

En la segunda celda, se hace uso de la librería 'pandas' para cargar el fichero con los datos y, a este, se le asigna una variable: 'df'.

Una vez realizados los pasos previamente explicados, se pasa a visualizar el contenido y a eliminarlo o, en algunos casos, editarlo.

```
#creamos una lista con las variables que se han eliminado en weka
variablesEliminadasWeka=['weight','metformin','repaglinide','nateglinide','chlorpropamide','glimepiride','acetohexamide',
                         'tolbutamide','acarbose','miglitol','troglitazone','examide',
                          'citoglipton','glyburide-metformin','glipizide-metformin',
                         'metformin-rosiglitazone','metformin-pioglitazone','tolazamide','payer_code','medical_specialty']
df = df.drop(df[variablesEliminadasWeka], axis=1)
df.info()
```

Ilustración 15. Eliminación de variables Python.

En este fragmento del código se han eliminado las variables que se han sido previamente eliminadas en WEKA. Para ello se hace un vector con todas las variables que se quieren eliminar y haciendo uso de la librería 'Pandas', se eliminan de la lista de variables. Finalmente, se muestra el resultado en la ilustración 16.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 30 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   encounter_id              101766 non-null  int64
 1   patient_nbr               101766 non-null  int64
 2   race                      101766 non-null  object
 3   gender                    101766 non-null  object
 4   age                       101766 non-null  object
 5   admission_type_id         101766 non-null  int64
 6   discharge_disposition_id  101766 non-null  int64
 7   admission_source_id       101766 non-null  int64
 8   time_in_hospital          101766 non-null  int64
 9   num_lab_procedures        101766 non-null  int64
 10  num_procedures            101766 non-null  int64
 11  num_medications           101766 non-null  int64
 12  number_outpatient         101766 non-null  int64
 13  number_emergency          101766 non-null  int64
 14  number_inpatient          101766 non-null  int64
 15  diag_1                    101766 non-null  object
 16  diag_2                    101766 non-null  object
 17  diag_3                    101766 non-null  object
 18  number_diagnoses          101766 non-null  int64
 19  max_glu_serum             101766 non-null  object
 20  A1Cresult                 101766 non-null  object
 21  glipizide                 101766 non-null  object
 22  glyburide                 101766 non-null  object
 23  pioglitazone              101766 non-null  object
 24  rosiglitazone             101766 non-null  object
 25  insulin                   101766 non-null  object
 26  glimepiride-pioglitazone  101766 non-null  object
 27  change                    101766 non-null  object
 28  diabetesMed               101766 non-null  object
 29  readmitted                101766 non-null  int64
```

Ilustración 16. Variables finales Python.

A continuación, se van a eliminar o editar con Python las variables que no han sido eliminadas anteriormente ya que, ahora, se mira su significado y si este tiene sentido o no.

```
[49] df = df.drop(['diag_1'], axis=1)
     df = df.drop(['diag_2'], axis=1)
     df = df.drop(['diag_3'], axis=1)
```

Ilustración 17. Variables a eliminar por su significado.

Se eliminan estas variables ya que, como podemos observar en la siguiente ilustración, los valores de las instancias no son del mismo tipo, es decir, encontramos instancias con valores alfanuméricos, algo que no nos interesa ya que necesitamos solo datos numéricos.

```
df['diag_2'].value_counts()

276     6752
428     6662
250     6071
427     5036
401     3736
        ...
E918       1
46         1
V13        1
E850       1
927        1
Name: diag_2, Length: 749, dtype: int64
```

Ilustración 18. Lectura variables atributo 'diag_2'.

En la ilustración 18 encontramos 2273 variables cuyo valor es '?'. Ninguna variable puede tener un valor desconocido por lo que se añade ese valor a la variable 'Caucasian' que es la variable con una muestra más grande.

```
df['race'].value_counts()

Caucasian          76099
AfricanAmerican    19210
?                   2273
Hispanic            2037
Other               1506
Asian                641
Name: race, dtype: int64
```

Ilustración 19. Muestras variable 'race'.

```
df.loc[df['race']=='?','race']='Caucasian'
df['race'].value_counts()
```

Ilustración 20. Edición variable 'race'.

En esta celda se han cambiado los valores ‘?’ por ‘caucasian’ ya que esta es la variable con más muestras tiene, en el atributo ‘race’.

```
df.loc[df['race']=='?','race']='Caucasian'
df['race'].value_counts()
```

```
Caucasian          78372
AfricanAmerican    19210
Hispanic            2037
Other               1506
Asian                641
Name: race, dtype: int64
```

Ilustración 21. Muestras variable 'race'.

Para la variable género se procede igual que en el anterior.

```
df['gender'].value_counts()
```

```
Female             54708
Male               47055
Unknown/Invalid        3
Name: gender, dtype: int64
```

Ilustración 22. Variables género.

Y una persona no puede tener género invalido, por lo que se añade al grupo de los hombres. Se obtiene lo siguiente:

```
df.loc[df['gender']=='Unknown/Invalid','gender']='Male'
df['gender'].value_counts()
```

Ilustración 23. Modificando variable género.

Se añaden los tres valores a los hombres.

```
df.loc[df['gender']=='Unknown/Invalid','gender']='Male'
df['gender'].value_counts()
```

```
Female    54708
Male      47058
Name: gender, dtype: int64
```

Ilustración 24. Modificando variable género.

A continuación, se va a utilizar la herramienta 'labelEnconder'.

El preprocesamiento se utiliza para codificar las etiquetas de destino con un valor entre 0 y n_classes-1. Como se ha dicho, este transformador debe usarse para codificar valores destino, es decir, los valores 'y', y no la entrada 'x'.

La etiqueta 'labelEncoder' utilizar para normalizar etiquetas o para transformar etiquetas no numéricas en etiquetas numéricas. Entre sus métodos se encuentran 'fit()' que ajusta el codificador de etiquetas, 'get_params([])' que obtiene parámetros para este estimador, 'inverse_transform()' que transforma las etiquetas de nuevo en la codificación original, 'set_params(**params)' que establece los parámetros de este estimador, 'transform()' que transforma etiquetas a codificación normalizada y, por último, 'fit_transform()' que ajusta el codificador de etiquetas y devuelve las etiquetas codificadas.

Con esta herramienta convertimos a numérico y dando uso a este último método descrito. A continuación, un ejemplo:

```
print(df['A1Cresult'].value_counts())
df['A1Cresult']=l.fit_transform(df['A1Cresult'])
df['A1Cresult'].value_counts()
```

```
None    84748
>8       8216
Norm     4990
>7       3812
Name: A1Cresult, dtype: int64
2    84748
1     8216
3     4990
0     3812
Name: A1Cresult, dtype: int64
```

Ilustración 25. Cambio a variable numérica.

```
df['max_glu_serum']=l.fit_transform(df['max_glu_serum'])
df['max_glu_serum'].value_counts()
```

```
2    96420
3     2597
0     1485
1     1264
Name: max_glu_serum, dtype: int64
```

Ilustración 26. Cambio a variable numérica.

```
print(df['age'].value_counts())
df['age']=l.fit_transform(df['age'])
df['age'].value_counts()
```

```
[70-80)     26068
[60-70)     22483
[50-60)     17256
[80-90)     17197
[40-50)      9685
[30-40)      3775
[90-100)     2793
[20-30)      1657
[10-20)       691
[0-10)        161
Name: age, dtype: int64
7    26068
6    22483
5    17256
8    17197
4     9685
3     3775
9     2793
2     1657
1      691
0      161
Name: age, dtype: int64
```

Ilustración 27. Cambio a variable numérica.

```
print(df['glipizide'].value_counts())
df['glipizide']=l.fit_transform(df['glipizide'])
df['glipizide'].value_counts()
```

```
No        89080
Steady    11356
Up          770
Down        560
Name: glipizide, dtype: int64
1    89080
2    11356
3      770
0      560
Name: glipizide, dtype: int64
```

Ilustración 28. Cambio a variable numérica.

```
print(df['glyburide'].value_counts())
df['glyburide']=l.fit_transform(df['glyburide'])
df['glyburide'].value_counts()
```

```
No        91116
Steady     9274
Up          812
Down        564
Name: glyburide, dtype: int64
1    91116
2     9274
3      812
0      564
Name: glyburide, dtype: int64
```

Ilustración 29. Cambio a variable numérica.

```
print(df['pioglitazone'].value_counts())
df['pioglitazone']=l.fit_transform(df['pioglitazone'])
df['pioglitazone'].value_counts()
```

```
No        94438
Steady     6976
Up          234
Down        118
Name: pioglitazone, dtype: int64
1    94438
2     6976
3      234
0      118
Name: pioglitazone, dtype: int64
```

Ilustración 30. Cambio a variable numérica.

```
print(df['rosiglitazone'].value_counts())
df['rosiglitazone']=l.fit_transform(df['rosiglitazone'])
df['rosiglitazone'].value_counts()
```

```
No        95401
Steady     6100
Up          178
Down         87
Name: rosiglitazone, dtype: int64
1    95401
2     6100
3      178
0       87
Name: rosiglitazone, dtype: int64
```

Ilustración 31. Cambio a variable numérica.

```
print(df['insulin'].value_counts())
df['insulin']=l.fit_transform(df['insulin'])
df['insulin'].value_counts()
```

```
No        47383
Steady    30849
Down      12218
Up        11316
Name: insulin, dtype: int64
1    47383
2    30849
0    12218
3    11316
Name: insulin, dtype: int64
```

Ilustración 32. Cambio a variable numérica.

```
df['readmitted']=l.fit_transform(df['readmitted'])
df['readmitted'].value_counts()
```

```
2    54864
1    35545
0    11357
Name: readmitted, dtype: int64
```

Ilustración 33. Cambio a variable numérica.

A continuación, se va a convertir aquellos valores que son de tipo 'object' a tipo 'int64', haciendo uso del 'labelEncoder' anteriormente usado. Esto se realiza para que los clasificadores puedan trabajar con las variables que eran de tipo 'object'.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 29 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   encounter_id              101766 non-null  int64
 1   patient_nbr               101766 non-null  int64
 2   race                      101766 non-null  object
 3   gender                    101766 non-null  object
 4   age                       101766 non-null  int64
 5   admission_type_id         101766 non-null  int64
 6   discharge_disposition_id  101766 non-null  int64
 7   admission_source_id       101766 non-null  int64
 8   time_in_hospital          101766 non-null  int64
 9   num_lab_procedures        101766 non-null  int64
 10  num_procedures            101766 non-null  int64
 11  num_medications           101766 non-null  int64
 12  number_outpatient         101766 non-null  int64
 13  number_emergency          101766 non-null  int64
 14  number_inpatient          101766 non-null  int64
 15  diag_1                    101766 non-null  object
 16  diag_2                    101766 non-null  object
 17  diag_3                    101766 non-null  object
 18  number_diagnoses          101766 non-null  int64
 19  max_glu_serum             101766 non-null  int64
 20  A1Cresult                 101766 non-null  int64
 21  glipizide                 101766 non-null  int64
 22  glyburide                 101766 non-null  int64
 23  pioglitazone              101766 non-null  int64
 24  rosiglitazone             101766 non-null  int64
 25  insulin                   101766 non-null  int64
 26  change                    101766 non-null  object
 27  diabetesMed               101766 non-null  object
 28  readmitted                101766 non-null  int64
dtypes: int64(22), object(7)
memory usage: 22.5+ MB
```

Ilustración 34. Variables para cambiar.

Se realiza con las siguientes instrucciones para las categorías resaltadas:

```
print(df['race'].value_counts())
df['race']=l.fit_transform(df['race'])
df['race'].value_counts()
```

```
Caucasian          78372
AfricanAmerican    19210
Hispanic            2037
Other               1506
Asian                641
Name: race, dtype: int64
2    78372
0    19210
3     2037
4     1506
1      641
Name: race, dtype: int64
```

Ilustración 35. Cambio tipo de variable.

```
print(df['gender'].value_counts())
df['gender']=l.fit_transform(df['gender'])
df['gender'].value_counts()
```

```
Female    54708
Male      47058
Name: gender, dtype: int64
0    54708
1    47058
Name: gender, dtype: int64
```

Ilustración 36. Cambio tipo de variable.

```
print(df['diabetesMed'].value_counts())
df['diabetesMed']=l.fit_transform(df['diabetesMed'])
df['diabetesMed'].value_counts()
```

```
Yes    78363
No     23403
Name: diabetesMed, dtype: int64
1    78363
0    23403
Name: diabetesMed, dtype: int64
```

Ilustración 37. Cambio tipo de variable.

```
print(df['change'].value_counts())
df['change']=l.fit_transform(df['change'])
df['change'].value_counts()
```

```
No    54755
Ch    47011
Name: change, dtype: int64
1    54755
0    47011
Name: change, dtype: int64
```

Ilustración 38. Cambio tipo de variable.

A continuación, hay que fijarse en el archivo 'IDs_mapping.csv' que proporcionaba la descarga del dataset.

En este archivo se encuentran las siguientes variables: admission_type_id, discharge_disposition_id, admission_source_id.

Pero se encuentra el siguiente problema: hay un valor dentro de esta variable que es NULL. Esto ocurre para las otras dos variables (ver siguientes ilustraciones).

admission_type_id,description
1,Emergency
2,Urgent
3,Elective
4,Newborn
5,Not Available
6,NULL
7,Trauma Center
8,Not Mapped

Ilustración 39. Valores no válidos.

11	discharge_disposition_id,description
12	1,Discharged to home
13	2,Discharged/transferred to another short term hospital
14	3,Discharged/transferred to SNF
15	4,Discharged/transferred to ICF
16	5,Discharged/transferred to another type of inpatient care institution
17	6,Discharged/transferred to home with home health service
18	7,Left AMA
19	8,Discharged/transferred to home under care of Home IV provider
20	9,Admitted as an inpatient to this hospital
21	10,Neonate discharged to another hospital for neonatal aftercare
22	11,Expired
23	12,Still patient or expected to return for outpatient services
24	13,Hospice / home
25	14,Hospice / medical facility
26	15,Discharged/transferred within this institution to Medicare approved swing bed
27	16,Discharged/transferred/referred another institution for outpatient services
28	17,Discharged/transferred/referred to this institution for outpatient services
29	18,NULL
30	19,"Expired at home. Medicaid only, hospice."
31	20,"Expired in a medical facility. Medicaid only, hospice."
32	21,"Expired, place unknown. Medicaid only, hospice."
33	22,Discharged/transferred to another rehab fac including rehab units of a hospital .
34	23,Discharged/transferred to a long term care hospital.
35	24,Discharged/transferred to a nursing facility certified under Medicaid but not certified under Medicare.
36	25,Not Mapped
37	26,Unknown/Invalid
38	30,Discharged/transferred to another Type of Health Care Institution not Defined Elsewhere
39	27,Discharged/transferred to a federal health care facility.
40	28,Discharged/transferred/referred to a psychiatric hospital of psychiatric distinct part unit of a hospital

Ilustración 40. Valores no válidos.

admission_source_id,description
1, Physician Referral
2,Clinic Referral
3,HMO Referral
4,Transfer from a hospital
5, Transfer from a Skilled Nursing Facility (SNF)
6, Transfer from another health care facility
7, Emergency Room
8, Court/Law Enforcement
9, Not Available
10, Transfer from critial access hospital
11,Normal Delivery
12, Premature Delivery
13, Sick Baby
14, Extramural Birth
15,Not Available
17,NULL
18, Transfer From Another Home Health Agency
19,Readmission to Same Home Health Agency
20, Not Mapped
21,Unknown/Invalid
22, Transfer from hospital inpt/same fac reslt in a sep claim
23, Born inside this hospital
24, Born outside this hospital
25, Transfer from Ambulatory Surgery Center
26,Transfer from Hospice

Ilustración 41. Valores no válidos.

Para evitar el posible ruido y problemas que nos puedan proporcionar estas variables, se van a juntar todos los valores de este estilo en una sola variable, por ejemplo, en NULL. Es decir, para el primer caso, introduciremos 'Not Mapped' en 'NULL'.

Esto debe realizarse del siguiente modo:

```
df.loc[(df['admission_type_id']==5) | (df['admission_type_id']==6) | (df['admission_type_id']==8) ,'admission_type_id']=5
df['admission_type_id'].value_counts()
```

```
1    53990
3    18869
2    18480
5    10396
7       21
4       10
Name: admission_type_id, dtype: int64
```

Ilustración 42. Unión valores no válidos.

Se añaden las instancias con valor 5,6 y 8 a la variable 5, es decir, a 'Not Available', para así eliminar ruido y evitar problemas posteriores.

A continuación, se va a utilizar el 'labelEncoder' que se ha usado antes para que asigne correctamente los valores de la variable. Y se obtendrá el siguiente resultado:

```
df['admission_type_id']=l.fit_transform(df['admission_type_id'])
df['admission_type_id'].value_counts()
```

```
0    53990
2    18869
1    18480
4    10396
5       21
3       10
Name: admission_type_id, dtype: int64
```

Ilustración 43. Asignando los valores correctamente.

Se realizará el mismo procedimiento para las otras dos variables: discharge_disposition_id, admission_source_id.

Para discharge_disposition_id:

```
[198] #añadimos los valores NUll, Not Available etc... a 18 (NULL)
      df.loc[(df['discharge_disposition_id']==18) |
             (df['discharge_disposition_id']==25) |
             (df['discharge_disposition_id']==26) ,'discharge_disposition_id']=18
      df['discharge_disposition_id']=l.fit_transform(df['discharge_disposition_id'])
      df['discharge_disposition_id'].value_counts()
```

```
0     60234
2     13954
5     12902
17     4680
1      2128
20     1993
10     1642
4      1184
3       815
6       623
21      412
12      399
13      372
24      139
7       108
14       63
22       48
8        21
16       14
15       11
18        8
9         6
23        5
11        3
19        2
Name: discharge_disposition_id, dtype: int64
```

Ilustración 44. Asignando los valores correctamente.

Para admission_source_id:

```
#añadimos los valores NUll, Not Available etc... a 9 (NotAvailable)
df.loc[(df['admission_source_id']==9) |
       (df['admission_source_id']==15) |
       (df['admission_source_id']==17) |
       (df['admission_source_id']==20) |
       (df['admission_source_id']==21)
        ,'admission_source_id']=9
df['admission_source_id']=l.fit_transform(df['admission_source_id'])
df['admission_source_id'].value_counts()
```

```
6     57494
0     29565
8      7067
3      3187
5      2264
1      1104
4       855
2       187
7        16
13       12
9         8
12        2
10        2
14        2
11        1
Name: admission_source_id, dtype: int64
```

Ilustración 45. Asignando los valores correctamente.

Hasta aquí se han eliminado o editado las variables que son necesarias eliminar/editar.

Visualización de los datos resultantes

A continuación, se va a realizar un estudio de la correlación de las variables que se han obtenido después de hacer la limpieza de los datos, esto servirá para ver si un par de variables se encuentran altamente correlacionadas o no.

Se crea una matriz de correlación con la siguiente instrucción:

```
sns.set(rc={"figure.figsize":(20, 20)})
upp_mat = np.triu(df.corr())
sns.heatmap(df.corr(), vmin = -1, vmax = +1, annot = True, cmap = 'coolwarm', mask = upp_mat)
```

Ilustración 46. Matriz de correlación.

Y se obtiene el siguiente mapa de calor:

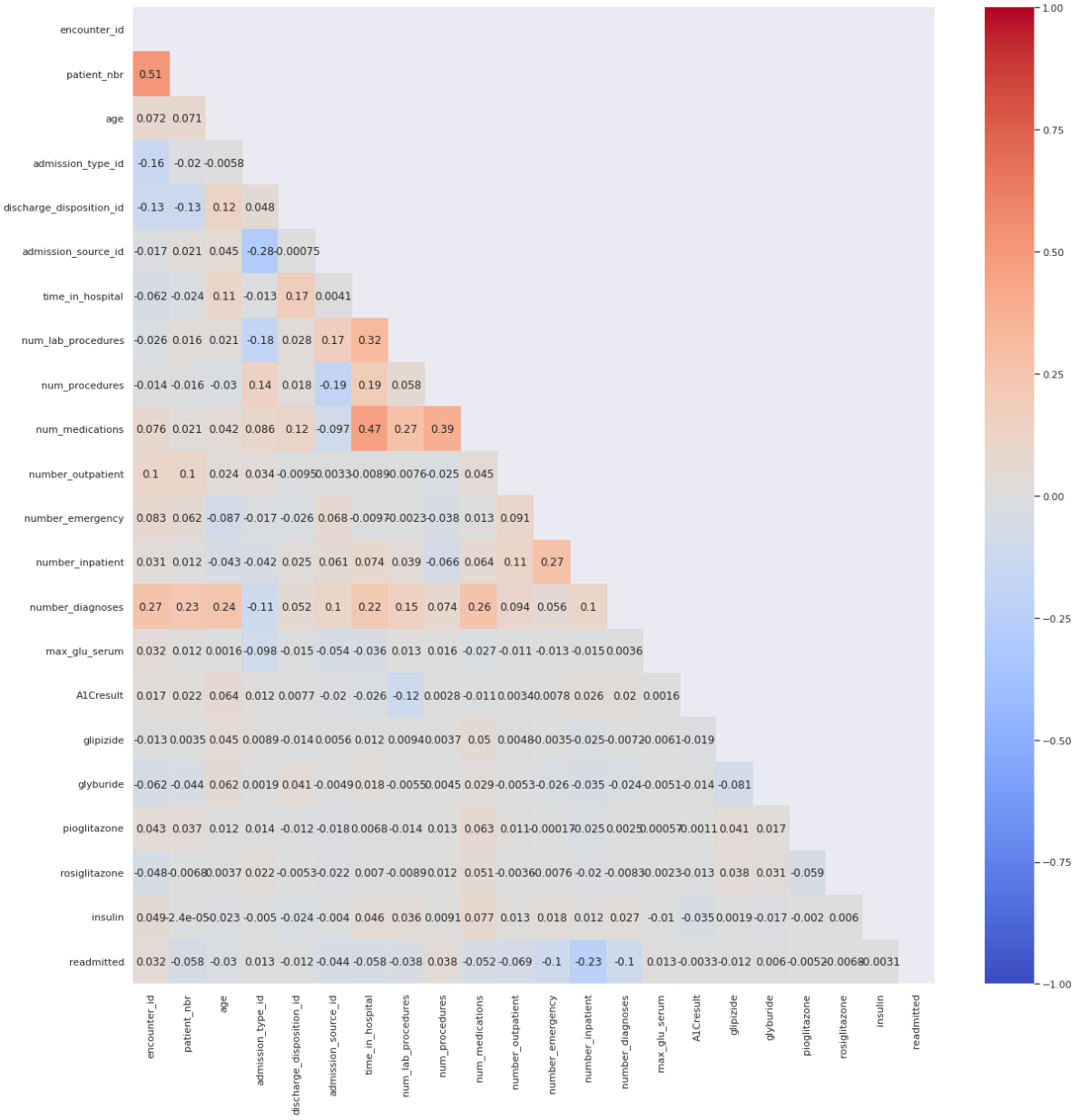


Ilustración 47. Mapa de calor.

Si dos variables se encuentran muy correlacionadas estarán cercanas al -1 o al 1. Mientras más cercanas al 0, menor correlación.

Se puede observar que la mayoría de las variables no se encuentran muy correlacionadas. Aunque hay excepciones como por ejemplo 'num_medications', el número de medicamentos que una persona recibe y 'time_in_hospital', tiempo en el hospital. A mayor tiempo en el hospital, mayor número de medicamentos recibe, por lo que es correcto que se obtenga esta correlación.

A continuación, se observa una figura donde se puede observar la distribución de las variables numéricos que se encuentran en nuestro dataset, después de haber hecho la limpieza y edición de los datos:

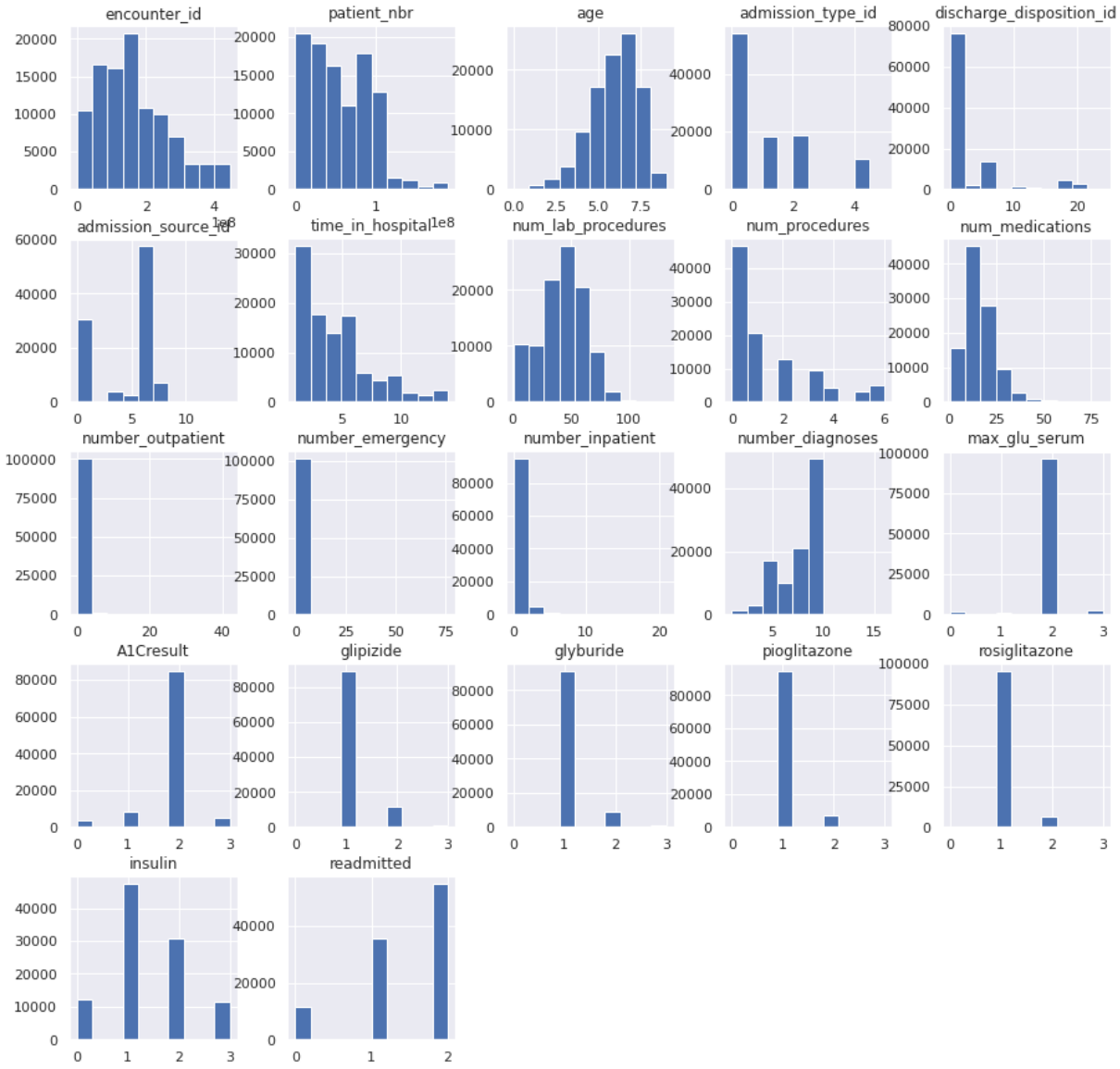


Ilustración 48. Histogramas de variables dataset.

2.2 Comenzando a clasificar

La curva ROC (Receiver Operating Characteristic) es una es una representación gráfica que ilustra la relación entre la sensibilidad y la especificidad de un sistema clasificador para diferentes puntos de corte.

Un modelo de clasificación es una función que permite decidir qué elementos de un conjunto de instancias están relacionados o no por pertenecer a un mismo tipo o clase. El resultado del clasificador o del diagnóstico puede ser un número real (valor continuo), en cuyo caso el límite del clasificador entre cada clase debe determinarse por un valor umbral o puede ser un resultado discreto que indica directamente una de las clases.

La elección se realiza mediante la comparación del área bajo la curva (AUC) de ambas pruebas. Esta área posee un valor comprendido entre 0,5 y 1, donde 1 representa un valor diagnóstico perfecto y 0,5 es una prueba sin capacidad discriminatoria diagnóstica.

Es decir, si AUC para una prueba diagnóstica es 0,8 significa que existe un 80% de probabilidad de que el diagnóstico realizado a un enfermo sea más correcto que el de una persona sana escogida al azar. Por esto, siempre se elige la prueba diagnóstica que presente una mayor área bajo la curva.

Para ello, se han de saber interpretar los valores de la curva en función de nuestros clasificadores:

- [0.5]: Resultado es como lanzar una moneda.
- [0.5, 0.6]: Resultado malo.
- [0.6, 0.75]: Resultado regular.
- [0.75, 0.9]: Resultado bueno.
- [0.9, 0.97]: Resultado muy bueno.
- [0.97, 1]: Resultado excelente.

2.2.1 Dividiendo el dataset

Con la siguiente sentencia dividimos el data set en: conjunto de entrenamiento y test, para poder realizar las clasificaciones correctamente.

```
[54] from sklearn import metrics
     from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=1)
```

Ilustración 49. División dataset.

Clasificador básico: **Dummy classifier**

DummyClassifier hace predicciones que ignoran las características de entrada. Este clasificador sirve como una línea de base simple para comparar con otros clasificadores más complejos. Para seleccionar el comportamiento especifico de la linea base, se utiliza el parametro 'strategy'.

Este clasificador implementa varias estrategias simples para la clasificación. En este caso se ha utilizado 'most_frequent' que siempre predice la etiqueta más frecuente en el conjunto de entrenamiento.

```
from sklearn.dummy import DummyClassifier
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(x_train, y_train)
y_score=dummy_clf.predict(x_test)
y_scores=dummy_clf.predict_proba(x_test)
print('ROC AREA: '+ str(metrics.roc_auc_score(y_test, y_scores, multi_class='ovr')))
print('ACCURACY CLASSIFIER: '+ str(metrics.accuracy_score(y_test,y_score)))
```

```
ROC AREA: 0.5
ACCURACY CLASSIFIER: 0.5425469195244178
```

Ilustración 50. Clasificador Dummy.

Primer clasificador: Naive Bayes

Los métodos Naive Bayes son un conjunto de algoritmos de aprendizaje supervisado basados en la aplicación del teorema de Bayes con la suposición "ingenua" de independencia condicional entre cada par de características dado el valor de la variable de clase.

En este caso se ha utilizado Gaussian Naive Bayes para la clasificación. Se supone que la probabilidad de las características es gaussiana:

NIVEL BASICO:

```
#NAIVE BAYES
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)
y_score = gnb.predict(x_test)
y_scores=gnb.predict_proba(x_test)
print('ROC AREA: '+ str(metrics.roc_auc_score(y_test, y_scores, multi_class='ovr')))
print('ACCURACY CLASSIFIER: '+ str(metrics.accuracy_score(y_test,y_score)))
```

```
ROC AREA: 0.6353681436453984
ACCURACY CLASSIFIER: 0.5588582096885133
```

Ilustración 51. Clasificador NaiveBayes.

El resultado que obtenemos del clasificador NaiveBayes con la curva ROC es de 0.635 y como se ha visto en la guia de interpretación, es un resultado regular.

Segundo clasificador: Árbol de decision

Los árboles de decisión (DT) son un método de aprendizaje supervisado no paramétrico que se utiliza para la clasificación y la regresión . El objetivo es crear un modelo que prediga el valor de una variable de destino mediante el aprendizaje de reglas de decisión simples deducidas de las características de los datos. Un árbol puede verse como una aproximación constante por partes.

```
#ARBOL DE DECISION
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=1000, max_depth=10,class_weight='balanced')
clf.fit(x_train, y_train)
y_score = clf.predict(x_test)
y_scores=clf.predict_proba(x_test)
print('ROC AREA: '+ str(metrics.roc_auc_score(y_test, y_scores, multi_class='ovr')))
print('ACCURACY CLASSIFIER: '+ str(metrics.accuracy_score(y_test,y_score)))
```

```
ROC AREA: 0.6548806973785422
ACCURACY CLASSIFIER: 0.4993613049032131
```

Ilustración 52. Clasificador Árbol de decisión.

El resultado que obtenemos del clasificador Árbol de decisión con la curva ROC es de 0.655 y como se ha visto en la guia de interpretación, es un resultado regular. Finalmente, se puede observar que es un clasificador aleatorio (similar a lanzar una moneda) y se concluye que no es un buen clasificador para este conjunto de datos.

Ninguno de los dos clasificadores nos ha dado un resultado bueno, aunque si es notable que el clasificador NaiveBayes es mejor que este último clasificador.

2.2.1 Test de Wilcoxon

Es muy común verificar si los resultados de dos experimentos distintos son equivalentes o si uno es mejor que otro, para ello es por lo que se utiliza esta prueba.

La prueba de los rangos con signo de Wilcoxon es una prueba no paramétrica para comparar las medianas de dos muestras relacionadas y determinar si existen diferencias entre ellas. Esta prueba debe cumplir:

- No necesita una distribución especifica por lo que la curva que se genera es totalmente libre.
- Nivel ordinal de la variable dependiente.

Su objetivo es determinar que la diferencia no se deba al azar, sino que la diferencia sea estadísticamente significativa.

Esta prueba se aplica cuando hay varios resultados, como es este caso, tras realizar la validación cruzada ya que tenemos 10 resultados, uno por cada una de las particiones. Por lo que para justificar que un algoritmo es mejor que otro, hay que ver los diez resultados y si mayoritariamente los diez resultados son unos mejores que otros.

Para realizarlo, se ha implantado la librería gráfica matplotlib , el clasificador model_selection, el test de significancia wilcoxon y warnings que se utiliza para poder filtrarlos y eliminarlos.

```
#test de wilcoxon
from sklearn import model_selection
from scipy.stats import wilcoxon
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

dtc= DecisionTreeClassifier(random_state=100, max_depth=10,criterion='gini')
resultados_dtc=np.round(model_selection.cross_val_score(dtc,x_train,y_train,cv=10),2)
qda = QuadraticDiscriminantAnalysis()
resultados_qda=np.round(model_selection.cross_val_score(qda,x_train,y_train,cv=10),2)
print(resultados_dtc)
print(resultados_qda)
wilcox_V, p_value =  wilcoxon(resultados_dtc, resultados_qda, alternative='greater', zero_method='wilcox', correction=False)

print('Resultado de Wilcoxon')
print(f'Wilcoxon V: {wilcox_V}, p-value: {p_value:.2f}')
plt.plot(range(len(resultados_dtc)),resultados_dtc,'ro-',label='dtc')
plt.plot(range(len(resultados_qda)),resultados_qda,'bs-',label='qda')
plt.ylabel('Aciertos')
plt.xlabel('Prueba')
plt.legend()
```

Ilustración 53. Implementación WIlcoxon.

En primer lugar, se ha utilizado el clasificador de árbol de decisión, ya explicado anteriormente, utilizando sus parámetros.

- random_state: controla la aleatoriedad del estimador.
- max_depth: la profundidad máxima del árbol.
- Criterion: la función para medir la calidad de una división. En este caso se ha utilizado 'gini' para la impureza de Gini.

Y se aplica a los resultados.

Por otro lado, se ha utilizado el clasificador QDA, explicado posteriormente, que ajusta densidades condicionales de clase a los datos y usando la regla de Bayes. Y se aplica a los resultados.

```
[0.58 0.57 0.59 0.58 0.59 0.59 0.59 0.58 0.58 0.59]
[0.56 0.56 0.57 0.56 0.57 0.56 0.57 0.56 0.56 0.56]
```

Ilustración 54. Folds

En la ilustración anterior encontramos los diez resultados, fruto de la validación cruzada, de los dos clasificadores, árbol de decisión y QDA, respectivamente.

A continuación, se aplica Wilcoxon. Para su aplicación, utilizamos los resultados de los clasificadores, previamente mostrados, y distintos parámetros:

- alternative: testea si los valores de dtc son mayores que los valores de qda por lo que se pasa por parámetro 'greater' indicando que sea mayor.
- zero_method: el método que se usa es 'wilcox'.
- correction: sin necesidad de corrección.

Esto devuelve dos valores. El primer valor, llamado 'Wilcoxon' devuelve un valor en su escala. El segundo valor, llamado 'p-value' ofrece una idea del solapamiento, es decir, como de distintas o parecidas son las distribuciones. Si el valor es muy bajo, las distribuciones serán distintas. A mayor valor, mayor parecido.

```
Resultado de Wilcoxon
Wilcoxon V: 55.0, p-value: 0.00
```

Ilustración 55. Resultado Wilcoxon y solapamiento.

Como se muestra en la ilustración 55, el valor de la escala de wilcoxon es 55.0 en la escala de aciertos y el 'p-value' es nulo, por lo que indica que no hay solapamiento y que las distribuciones son totalmente distintas entre sí. Como este valor es menor que 0.05, se rechaza la hipótesis de que son iguales. Por lo que se puede afirmar que los resultados del árbol de decision tiene mejores resultados que los resultados de QDA, es decir, los resultados del árbol de decisión son mayores.

Finalmente, se muestran los resultados en el gráfico.

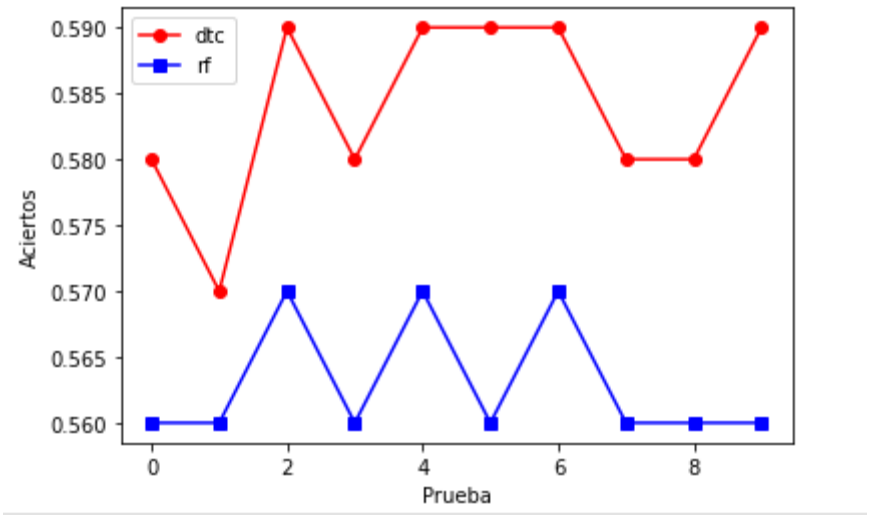


Ilustración 56. Representación escala Wilcoxon.

2.2.2 Mejorando un clasificador

El clasificador que se ha elegido para mejorar ha sido: Árbol de decisión.

```
#ARBOL DE DECISION
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=1000, max_depth=10)
clf.fit(x_train, y_train)
y_score = clf.predict(x_test)
y_scores=clf.predict_proba(x_test)
print('ROC AREA: '+ str(metrics.roc_auc_score(y_test, y_scores, multi_class='ovr')))
print('ACCURACY CLASSIFIER: '+ str(metrics.accuracy_score(y_test,y_score)))
```

```
ROC AREA: 0.6639523293158976
ACCURACY CLASSIFIER: 0.5877960106121647
```

Ilustración 57. Clasificador Árbol de decisión mejorado.

Para ello, se va a aumentar tanto la curva ROC como la precisión del clasificador. A continuación, se explica cómo se ha conseguido esta mejora.

Se ha añadido el siguiente parámetro al clasificador: 'max_depth', con un valor igual a 10, y como podemos observar el resultado de la curva ROC ha mejorado significativamente de 0.54 a 0.66 y en la precisión del clasificador ha aumentado un 10% pasando de 0.48 a 0.58.

Aplicar este parámetro lo que hace es determinar la máxima profundidad del árbol, con esto conseguimos que el árbol no sea puro, es decir, que todos sus nodos no sean puros. Consiguiendo así reducir el overfitting, es decir, el sobre aprendizaje del clasificador sobre los datos.

2.2.3 Equilibrar el número de instancias

La clasificación desequilibrada implica el desarrollo de modelos predictivos en conjuntos de datos de clasificación que tienen un desequilibrio de clase severo.

Un enfoque para abordar conjuntos de datos desequilibrados es oversampling la clase minoritaria y el otro es subsampling. El enfoque más simple consiste en duplicar ejemplos en la clase minoritaria, aunque estos ejemplos no agregan ninguna información nueva al modelo. En su lugar, se pueden sintetizar nuevos ejemplos a partir de los ejemplos existentes. Una de estas técnicas se conoce como SMOTE, Systhetic Minority Oversampling Technique. Para el caso de submuestreo, implica seleccionar aleatoriamente ejemplos de la clase mayoritaria y eliminarlos del conjunto de datos de entrenamiento.

SMOTE funciona seleccionando ejemplos que están cerca en el espacio de funciones, dibujando una línea entre los ejemplos en el espacio de funciones y dibujando una nueva muestra en un punto a lo largo de esa línea.

Como se puede observar en la siguiente ilustración, el número de instancias de la variable objetivo está muy desbalanceado, por lo tanto, se va a realizar oversampling para equilibrar el número de instancias.

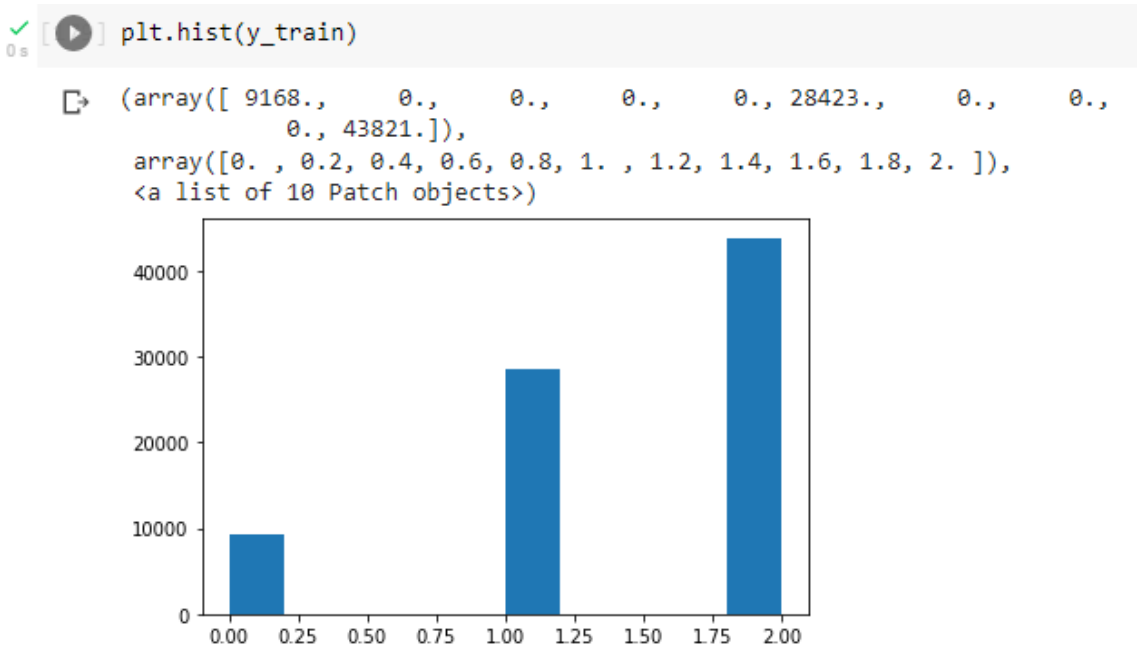


Ilustración 58. Histograma clasificación desbalanceado.

Se utiliza el clasificador SMOTE con el parámetro:

- random_state: controla la aleatoriedad del estimador.

```
from imblearn.over_sampling import SMOTE
sm= SMOTE(random_state=40)
x_train, y_train= sm.fit_resample(x_train,y_train)
plt.hist(y_train)
```

Ilustración 59. Implementación SMOTE.

```
(array([43821.,     0.,     0.,     0.,     0., 43821.,     0.,     0.,
            0., 43821.]),
 array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. ]),
 <a list of 10 Patch objects>)
```

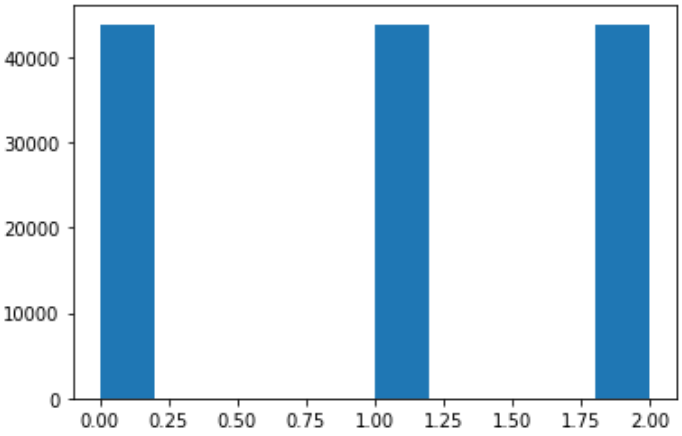


Ilustración 60. Resultado clasificación SMOTE.

Finalmente, se ha conseguido equilibrar el número de instancias de la variable objetivo.

Nivel intermedio

1.Feature selection

Primero probamos un clasificador para comprar después del feature selection:

```
X_scaled =  StandardScaler().fit_transform(x)
X_train, X_valid, y_train, y_valid = train_test_split(X_scaled,y,test_size = 0.2,stratify=y, random_state=1)
classifier = RandomForestClassifier()
classifier.fit(X_train,y_train)
preds = classifier.predict(X_valid)
print('Resultado del clasificador previo a feature selection: ')
accuracy_score(preds,y_valid)
```

```
Resultado del clasificador previo a feature selection:
0.6023386066620812
```

Ilustración 61. Probando clasificador.

Comenzamos con la selección de características:

Para instalar se implementa lo siguiente:

```
!pip install featurewiz
```

Ilustración 62. Instalación.

A continuación, se implementa el código, importando las librerías y utilizando el clasificador featurewiz. Featurewiz utiliza el algoritmo SULOV y Recursive XGBoost para reducir las características y seleccionar las mejores para el modelo.

SULOV (Searching for Uncorrelated List of Variables): realiza una búsqueda de una lista de variables no correlacionadas.

XGBoost recursivo. Se utiliza para encontrar las mejores características entre las restantes.

Para esto se han utilizado distintos parámetros:

- target: es la variable objetivo, en este caso es 'readmitted'.
- Df: es el dataframe entero, la variable objetivo es incluida, luego será separada.

```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from featurewiz import featurewiz
target = 'readmitted'

features, train = featurewiz(df, target, corr_limit=0.7, verbose=2, sep=",",
header=0,test_data="", feature_engg="", category_encoders="")
print(features)
```

Ilustración 63. Implementación código y librerías.

El resultado que se obtiene es el siguiente:

- features - una lista de características seleccionadas
- train - Este dataframe contiene sólo las características seleccionadas y la variable objetivo.

El resultado que nos brinda el algoritmo se puede ver en la siguiente ilustración:

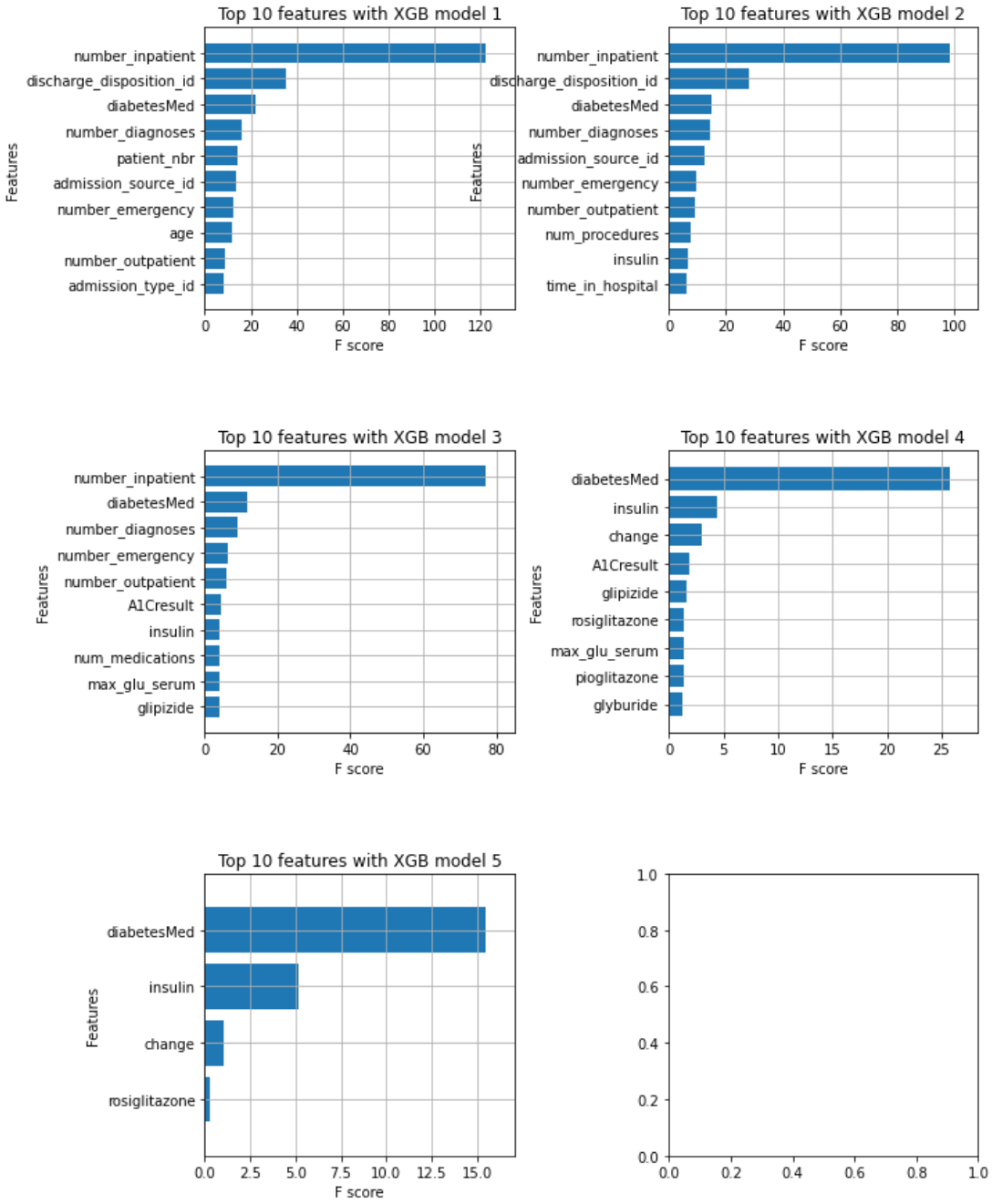


Ilustración 64. Resultado gráfico del algoritmo.

```
F score
        Total time taken for XGBoost feature selection = 13 seconds
    Alert: No ID variables ['encounter_id'] are included in selected features
#################################################################################
#####          F E A T U R E   S E L E C T I O N   C O M P L E T E D          #######
#################################################################################
Selected 21 important features:
['number_inpatient', 'discharge_disposition_id', 'diabetesMed', 'number_diagnoses', 'patient_nbr', 'admission_source_id',

    Time taken for feature selection = 15 seconds
    Reverted column names to original names given in train dataset
['number_inpatient', 'discharge_disposition_id', 'diabetesMed', 'number_diagnoses', 'patient_nbr', 'admission_source_id',
```

Ilustración 65. Resultado del algoritmo.

Donde, como se puede ver arriba, se obtiene una lista con las características que el algoritmo ha dejado en el dataframe, habiendo eliminado 4, antes teníamos 25 y ahora hemos obtenido 21.

Las características resultantes son las siguientes:

['number_inpatient', 'discharge_disposition_id', 'diabetesMed', 'number_diagnoses', 'patient_nbr', 'admission_source_id', 'number_emergency', 'age', 'number_outpatient', 'admission_type_id', 'num_procedures', 'insulin', 'time_in_hospital', 'A1Cresult', 'num_medications', 'max_glu_serum', 'glipizide', 'change', 'rosiglitazone', 'pioglitazone', 'glyburide']

A continuación, se crea de nuevo el conjunto de entrenamiento nuevo a partir del resultado anterior.

```
[139] X_new=train.drop(['readmitted'], axis=1)
      y_new= train.readmitted.values
      X_scaled =  StandardScaler().fit_transform(X_new)
      X_train, X_valid, y_train, y_valid = train_test_split(X_scaled,y,test_size = 0.2,stratify=y, random_state=1)
```

Ilustración 66. Conjunto de entrenamiento.

Se crea un clasificador como el inicial y se obtiene el resultado de este:

```
[140] classifier = RandomForestClassifier()
      classifier.fit(X_train,y_train)

      RandomForestClassifier()

      preds = classifier.predict(X_valid)
      print('Resultado del clasificador post feature selection')
      accuracy_score(preds,y_valid)

      Resultado del clasificador post feature selection
      0.5811142772919328
```

Ilustración 67. Clasificador Random Forest.

Como se puede observar, la precisión obtenida es peor que la inicial, por lo tanto, el algoritmo de selección de características no ha desempeñado un buen trabajo ya que, alguna de las variables que ha eliminado sí tenía importancia en el dataframe.

Sin embargo, esto ocurre para este clasificador 'de prueba', veamos los siguientes clasificadores y cómo estos han mejorado su resultado:

A continuación, se va a observar un ejemplo con otro clasificador:

KNN haciendo uso del dataframe antes de la feature selection:

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(15)
knn.fit(x_train, y_train)
y_score = knn.predict(x_test)
y_scores=knn.predict_proba(x_test)
print('ROC AREA: '+ str(metrics.roc_auc_score(y_test, y_scores, multi_class='ovr')))
print('ACCURACY CLASSIFIER: '+ str(metrics.accuracy_score(y_test,y_score)))
```

```
ROC AREA: 0.5001778490821067
ACCURACY CLASSIFIER: 0.49115652942910487
```

Ilustración 68. Resultado clasificador KNN.

KNN haciendo uso del dataframe después de la feature selection:

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(15)
knn.fit(X_train, y_train)
y_score = knn.predict(X_valid)
y_scores=knn.predict_proba(X_valid)
print('ROC AREA: '+ str(metrics.roc_auc_score(y_valid, y_scores, multi_class='ovr')))
print('ACCURACY CLASSIFIER: '+ str(metrics.accuracy_score(y_valid,y_score)))
```

```
ROC AREA: 0.6051791355694283
ACCURACY CLASSIFIER: 0.5508499557826472
```

Ilustración 69. Resultado clasificador KNN despues de feature selection.

Se puede observar, que la clasificación ha mejorado significativamente. El ROC área ha mejorado un 10% su valor y la precisión un 6%. Por lo que, se puede concluir que con el clasificador KNN, el uso del feature selection mejora el resultado.

2.Probando clasificadores

Al igual que anteriormente, se van a utilizar nuevos clasificadores, en este caso serán: KNN, Random Forest, Adaboost, QDA y Bagging.

2.1 KNN

KNeighbors Classifier es un tipo de aprendizaje basado en instancias o aprendizaje no generalizador: no intenta construir un modelo interno general, sino que simplemente almacena instancias de los datos de entrenamiento. La clasificación se calcula a partir de un voto de mayoría simple de los vecinos más cercanos de cada punto: a un punto de consulta se le asigna la clase de datos que tiene la mayor cantidad de representantes dentro de los vecinos más cercanos del punto.

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(15)
knn.fit(X_train, y_train)
y_score = knn.predict(X_valid)
y_scores=knn.predict_proba(X_valid)
print('ROC AREA: '+ str(metrics.roc_auc_score(y_valid, y_scores, multi_class='ovr')))
print('ACCURACY CLASSIFIER: '+ str(metrics.accuracy_score(y_valid,y_score)))

ROC AREA: 0.6051791355694283
ACCURACY CLASSIFIER: 0.5508499557826472
```

Ilustración 70. Clasificador KNN (KNeighbors).

El resultado que obtenemos del clasificador KNN con la curva ROC es de 0.60, por lo que, es un resultado regular.

2.2 RANDOM FOREST

Random Forest es un metaestimador que ajusta una serie de clasificadores de árboles de decisión en varias submuestras del conjunto de datos y utiliza el promedio para mejorar la precisión predictiva y controlar el sobreajuste.

En este clasificador, cada árbol del conjunto se construye a partir de una muestra extraída con reemplazo (es decir, una muestra de arranque) del conjunto de entrenamiento.

Se han usado dos parámetros:

- max_depth: controla el tamaño de los árboles. Si es *None*, los nodos se expanden hasta que todas las hojas sean puras. En este caso el número máximo del tamaño del árbol es de 2. Para reducir el consumo de la memoria, la complejidad y el tamaño de los arboles se debe controlar este parámetro.
- random_state: controla tanto la aleatoriedad del arranque de las muestras utilizadas al construir árboles como el muestreo de las caracteristicas a considerar cuando se busca la mejor division en cada nodo.

```
[154] #RANDOM FOREST
      from sklearn.ensemble import RandomForestClassifier
      clf = RandomForestClassifier(max_depth=10, random_state=0)
      clf.fit(X_train, y_train)
      y_score = clf.predict(X_valid)
      y_scores=clf.predict_proba(X_valid)
      print('ROC AREA: '+ str(metrics.roc_auc_score(y_valid, y_scores, multi_class='ovr')))
      print('ACCURACY CLASSIFIER: '+ str(metrics.accuracy_score(y_valid,y_score)))

      ROC AREA: 0.6788295665065641
      ACCURACY CLASSIFIER: 0.590056008646949
```

Ilustración 71. Clasificador Random Forest.

El resultado que obtenemos del clasificador Random Forest con la curva ROC es de 0.679, por lo que, es un resultado regular.

2.3 ADABOOST (TÉCNICA DE BOOSTING)

Un clasificador AdaBoost es un metaestimador que comienza ajustando un clasificador en el conjunto de datos original y luego ajusta copias adicionales del clasificador en el mismo conjunto de datos, pero donde los pesos de las instancias clasificadas incorrectamente se ajustan de modo que los clasificadores posteriores se centren más en casos difíciles.

Se han usado tres parametros distintos:

- n_estimators: El número máximo de estimadores en los que finaliza el refuerzo. En caso de ajuste perfecto, el procedimiento de aprendizaje se detiene antes de tiempo. En este caso se ha indicado que el número máximo de estimadores sea 100.
- random_state: Controla la semilla aleatoria dada en cada iteracion del impulso. En este caso es 0.
- algorithm = 'SAMME': utiliza el algoritmo de refuerzo discreto. Converge más lento que el algoritmo 'SAMME.R'.

```
#ADABOOST
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(n_estimators=100, random_state=0,algorithm='SAMME.R')
clf.fit(X_train, y_train)
y_score = clf.predict(X_valid)
y_scores=clf.predict_proba(X_valid)
print('ROC AREA: '+ str(metrics.roc_auc_score(y_valid, y_scores, multi_class='ovr')))
print('ACCURACY CLASSIFIER: '+ str(metrics.accuracy_score(y_valid,y_score)))
```

```
ROC AREA: 0.6720528937651883
ACCURACY CLASSIFIER: 0.587452097867741
```

Ilustración 72. Clasificador AdaBoost.

El resultado que obtenemos del clasificador Adaboost con la curva ROC es de 0.672, por lo que, es un resultado regular.

2.4 QDA

QDA (Análisis Discriminante Cuadrático) es un clasificador con un límite de decisión cuadrático, generado ajustando densidades condicionales de clase a los datos y usando la regla de Bayes.

Este clasificador es atractivo porque tiene soluciones de forma cerrada que se pueden calcular fácilmente, es inherentemente multiclase, han demostrado que funcionan bien en la práctica y no tienen hiperparámetros para ajustar.

```
[156] #QuadraticDiscriminantAnalysis
      from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
      clf = QuadraticDiscriminantAnalysis()
      clf.fit(X_train, y_train)
      y_score = clf.predict(X_valid)
      y_scores=clf.predict_proba(X_valid)
      print('ROC AREA: '+ str(metrics.roc_auc_score(y_valid, y_scores, multi_class='ovr')))
      print('ACCURACY CLASSIFIER: '+ str(metrics.accuracy_score(y_valid,y_score)))

      ROC AREA: 0.6323010046259273
      ACCURACY CLASSIFIER: 0.5545838655792473
```

Ilustración 73. Clasificador QDA.

El resultado que obtenemos del clasificador QDA con la curva ROC es de 0.632, por lo que, es un resultado regular.

2.5 BAGGING

Bagging es un metaestimador de conjunto que ajusta los clasificadores base a cada uno en subconjuntos aleatorios del conjunto de datos original y luego agrega sus predicciones individuales (ya sea por votación o por promedio) para formar una predicción final.

Se han usado dos parámetros:

- max_samples: es el número de muestras para extraer de X para entrenar cada estimador base. En este caso, al ser float, 0.5, toma las muestras. Si fuera un entero, extraería las muestras.
- max_features: es la cantidad de características que se extraerán de X para entrenar a cada estimador base. Como también es float, 0.5, dibuja entidades; si fuera entero dibujaría características.

```
#BAGGING
from sklearn.ensemble import BaggingClassifier
clf = BaggingClassifier(KNeighborsClassifier(),max_samples=0.7, max_features=0.5)
clf.fit(X_train, y_train)
y_score = clf.predict(X_valid)
y_scores=clf.predict_proba(X_valid)
print('ROC AREA: '+ str(metrics.roc_auc_score(y_valid, y_scores, multi_class='ovr')))
print('ACCURACY CLASSIFIER: '+ str(metrics.accuracy_score(y_valid,y_score)))

ROC AREA: 0.634561068269644
ACCURACY CLASSIFIER: 0.5698142871180112
```

Ilustración 74. Clasificador Bagging.

El resultado que obtenemos del clasificador KNN con la curva ROC es de 0.635, por lo que, es un resultado malo.

3.Alterando parámetros para mejorar la clasificación

Se va a realizar la alteración de los parámetros de cada uno de los cinco clasificadores anteriores para tratar de mejorar su precisión y el valor del ROC área.

3.1 KNeighborsClassifier GridSearchCV

GridSearchCV implementa un método de "ajuste" y "puntuación". Realiza una búsqueda exhaustiva sobre valores de parámetros específicos para un estimador.

Los parámetros del estimador utilizados para aplicar estos métodos se optimizan mediante una búsqueda de cuadrícula con validación cruzada sobre una cuadrícula de parámetros.

El número de vecinos idóneo se ha determinado realizando una búsqueda en grid, es decir, aplicando la siguiente sentencia:

```
knn1 = KNeighborsClassifier()
from sklearn.model_selection import GridSearchCV
k_range = list(range(1, 31))
param_grid = dict(n_neighbors=k_range)


grid = GridSearchCV(knn1, param_grid, cv=5, scoring='accuracy', return_train_score=False,verbose=1)


grid_search=grid.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
```

Ilustración 75. Obtención número de vecinos.

El objetivo de esta búsqueda en grid es encontrar el parámetro idóneo para mejorar el resultado del clasificador lo máximo posible. Primero creamos una instancia de clasificador KNN y luego preparamos un rango de valores del hiperparámetro k_range del 1 al 31 que será utilizado por GridSearchCV para encontrar el mejor valor de este.

Además, establecemos nuestros tamaños de lote de validación cruzada cv = 5.

A continuación, mostramos el parámetro que nos brinda la búsqueda en grid:

```
print(grid_search.best_params_)
```

Ilustración 76. Obtención número de vecinos.

Y El resultado obtenido es el siguiente:

```
{'n_neighbors': 26}
```

Ilustración 77. Número de vecinos obtenido.

En la siguiente ilustración se puede observar que hemos aumentado el número de vecinos a 26, lo que nos da un mejor resultado tanto en área ROC como en la accuracy o precisión del clasificador.

Ilustración 78. Clasificador KNN mejorado.

En el resultado anterior encontrábamos un valor del ROC área de 0.616 mientras que ahora ha aumentado a 0.621. Por otro lado, la precisión ha mejorado de 0.555 a 0.561.

3.2 Random Forest

Se ha cambiado el parámetro: *max_depth* el cual indica: la profundidad máxima del árbol.

Para no aumentar la complejidad y el consumo de memoria en exceso, se ha aumentado el tamaño del árbol a 10. Esto nos permite tener una mayor profundidad sin que los nodos sean puros y así reducir el overfitting lo máximo posible.

```
#RANDOM FOREST
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(max_depth=10, random_state=0)
clf.fit(x_train, y_train)
y_score = clf.predict(x_test)
y_scores=clf.predict_proba(x_test)
print(metrics.roc_auc_score(y_test, y_scores, multi_class='ovr'))#roc area
print(metrics.accuracy_score(y_test,y_score))#resultado del clasificador
```

```
0.6931396695842359
0.6026333890144443
```

Ilustración 79. Clasificador Random Forest mejorado.

El resultado obtenido es bastante satisfactorio atendiendo al ROC área ya que es el mayor valor conseguido hasta el momento. Este valor ha aumentado más de un 4% su valor pasando de 0.651 a 0.693, aunque manteniéndose en un valor regular. Por otro lado, la precisión ha aumentado un 5% su valor de 0.553 a 0.603.

3.3 Adaboost

A continuación, vamos a alterar el algoritmo que utiliza Adaboost. Adaboost utiliza técnicas de boosting que consiste en: ponderar el conjunto de entrenamiento y entrenar varios clasificadores. A diferencia del bagging, el entrenamiento no es en paralelo sino en serie (usando el resultado de un clasificador para crear el siguiente).

Para mejorar el clasificador simplemente se ha cambiado el parámetro algorithm de 'SAMME' a 'SAMME.R'. Este es el algoritmo de impulso real y en comparación con el algoritmo que se usó al inicio, este converge más rápido logrando un error de prueba más bajo con menos iteraciones de esfuerzo.

```
#ADABOOST
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(n_estimators=100, random_state=0,algorithm='SAMME.R')
clf.fit(x_train, y_train)
y_score = clf.predict(x_test)
y_scores=clf.predict_proba(x_test)
print(metrics.roc_auc_score(y_test, y_scores, multi_class='ovr'))#roc area
print(metrics.accuracy_score(y_test,y_score))#resultado del clasificador
```

```
0.6854072221973544
0.6008646949002653
```

Ilustración 80. Clasificador AdaBoost mejorado.

El valor del ROC área ha aumento de 0.666 a 0.684 mientras que la precisión ha aumentado de 0.589 a 0.601.

3.4 QuadraticDiscriminantAnalysis

Como se ha comentado anteriormente, el análisis discriminante cuadrático no tiene hiperparámetros para ajustar, por lo que este clasificador no ha sufrido ningún cambio respecto al primer caso. No se ha podido mejorar sus resultados.

```
#QuadraticDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
clf = QuadraticDiscriminantAnalysis()
clf.fit(x_train, y_train)
y_score = clf.predict(x_test)
y_scores=clf.predict_proba(x_test)
print(metrics.roc_auc_score(y_test, y_scores, multi_class='ovr'))
print(metrics.accuracy_score(y_test,y_score))#resultado del clasificador
```

```
0.645633249651152
0.5642134224231109
```

Ilustración 81. Clasificador QDA.

3.5 BaggingClassifier

El algoritmo ahora utilizado es el KNeighborsClassifier(). Y, por otro lado, además, se ha aumentado el 'max_samples' a 0.7, lo que nos proporciona una mayor toma de muestras.

estos cambios nos brindan un mejor resultado en ambos aspectos a estudiar.

```
#BAGGING
from sklearn.ensemble import BaggingClassifier
clf = BaggingClassifier(KNeighborsClassifier(),max_samples=0.7, max_features=0.5)
clf.fit(x_train, y_train)
y_score = clf.predict(x_test)
y_scores=clf.predict_proba(x_test)
print(metrics.roc_auc_score(y_test, y_scores, multi_class='ovr'))
print(metrics.accuracy_score(y_test,y_score))#resultado del clasificador
```

```
0.616566010069076
0.5609216861550556
```

Ilustración 82. Clasificador Bagging mejorado.

Con la mejora de este clasificador, se ha mejorado el ROC área de 0.599 a 0.617, mejorando su resultado un nivel según la guía explicada anteriormente. En un principio el valor era un resultado malo, pero se ha conseguido mejorar y llegar a un resultado regular. Por otro lado, la precisión ha mejorado cerca de un 2% de 0.543 a 0.561.

4.Test estadísticos

Para este punto se han usado distintas librerías. Para el tratamiento de datos, 'pandas' y 'numpy' como en el transcurso del proyecto. Para los gráficos, 'matplot' y 'seaborn'. Para el preprocesado y análisis, 'statsmodels' y 'scipy'. Y, finalmente, para la configuración de warnings, 'warnings'.

En la variable objetivo, 'readmitted', se tiene una distribución **no normal**, a continuación, se verá por qué.

4.1 Test de kurtosis y Skewness (estudio de la normalidad)

Los estadísticos de asimetría (Skewness) y kurtosis pueden emplearse para detectar desviaciones de la normalidad.

1. Un valor de curtosis y/o coeficiente de asimetría entre -1 y 1, es generalmente considerada una ligera desviación de la normalidad.
2. Entre -2 y 2 hay una evidente desviación de la normal pero no extrema.

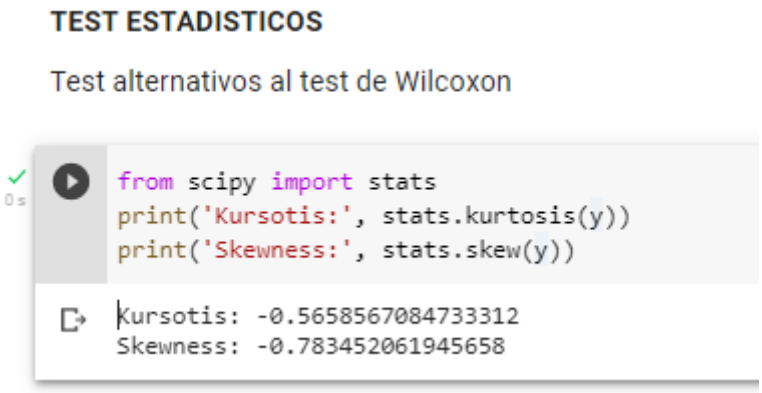


Ilustración 83. Resultado test Kurtosis y Skewness.

Como se observa, ambos valores entran en el rango de una ligera desviación de la normalidad ya que se encuentran entre -1 y 1. El valor de Kurtosis tiene un valor de -0.566 y el coeficiente de asimetría de -0.783.

4.2 Shapiro test

El Shapiro-Wilk test son dos de los test de hipótesis más empleados para analizar la normalidad. En ambos, si los datos proceden de una distribución normal, se considera una hipótesis nula.

El p-value de estos test indica la probabilidad de obtener unos datos como los observados, si realmente procediesen de una población con una distribución normal con la misma media y desviación que estos. Por lo tanto, si el p-value es menor que un determinado valor (típicamente 0.05), entonces se considera que hay evidencias suficientes para rechazar la normalidad.

```
# Shapiro-Wilk test
shapiro_test = stats.shapiro(y)
shapiro_test
print('Resultado de shapiro_test')
print(f'Wilcoxon V: {shapiro_test[0]}, p-value: {shapiro_test[1]:.2f}')
```

```
Resultado de shapiro_test
Wilcoxon V: 0.735634446144104, p-value: 0.00
```

Ilustración 84. Resultado test Shapiro-Wilk.

4.2.1 Demostración distribución NO normal

Como es obvio y era de esperar, estos datos no siguen una distribución normal porque la variable 'readmitted' ha pasado de ser una variable distribuida por intervalos:

```
[176] #label encoder
      l = LabelEncoder()
      read=df['readmitted']
      read.value_counts()
```

```
NO     54864
>30    35545
<30    11357
Name: readmitted, dtype: int64
```

Ilustración 85. Variable a intervalos.

A lo siguiente:

```
y.value_counts()
```

```
2    54864
1    35545
0    11357
Name: readmitted, dtype: int64
```

Ilustración 86. Variable sin intervalos.

Por lo tanto, nunca seguirán una distribución normal.

4.3 Test de Brunner-Munzel

La prueba de Brunner-Munzel es una prueba no paramétrica de la hipótesis nula de que cuando se toman valores uno por uno de cada grupo, las probabilidades de obtener valores grandes en ambos grupos son iguales.

```
from sklearn import model_selection
from scipy.stats import wilcoxon
from scipy import stats
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

dtc= KNeighborsClassifier(n_neighbors=26)
resultados_dtc=np.round(model_selection.cross_val_score(dtc,x_train,y_train,cv=10),2)
qda = QuadraticDiscriminantAnalysis()
resultados_qda=np.round(model_selection.cross_val_score(qda,x_train,y_train,cv=10),2)

w, p_value = stats.brunnermunzel(resultados_dtc, resultados_qda)
print(w)
print(p_value)
```

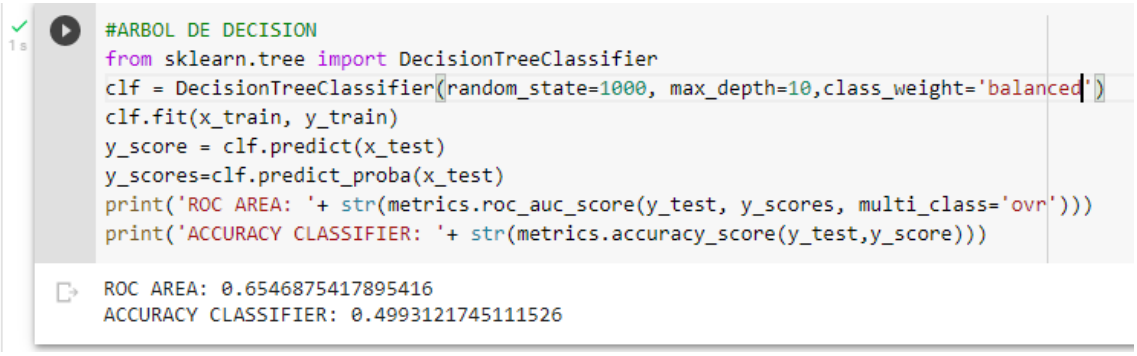
```
3.1672906063473705
0.01082491341852565
```

Ilustración 87. Resultados test de Brunner-Munzel.

5.Clasificadores sensibles al coste

El aprendizaje sensible a costes es un subcampo del aprendizaje automático que tiene en cuenta los costes de los errores de predicción (y potencialmente otros costes) al entrenar un modelo de aprendizaje automático. Como tal, muchas conceptualizaciones y técnicas desarrolladas y utilizadas para el aprendizaje sensible a costes pueden adoptarse para los problemas de clasificación desequilibrada, como es el caso de nuestro modelo.

En el siguiente ejemplo se ha incluido el parámetro 'class_weight'. Este parámetro indica los pesos asociados a las clases en el formulario. Al ser el modo 'balanced' utiliza los valores de y para ajustar automáticamente las ponderaciones de forma inversamente proporcional a las frecuencias de las clases en los datos de entrada.



```
#ARBOL DE DECISION
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=1000, max_depth=10,class_weight='balanced')
clf.fit(x_train, y_train)
y_score = clf.predict(x_test)
y_scores=clf.predict_proba(x_test)
print('ROC AREA: '+ str(metrics.roc_auc_score(y_test, y_scores, multi_class='ovr')))
print('ACCURACY CLASSIFIER: '+ str(metrics.accuracy_score(y_test,y_score)))
```

```
ROC AREA: 0.6546875417895416
ACCURACY CLASSIFIER: 0.4993121745111526
```

Ilustración 88. DecisionTree Classifier sensible al coste.

Como se puede observar ha disminuido tanto el ROC área como la precisión del clasificador. En el caso del valor del ROC área ha sido una bajada muy leve ya que ha sido menos de un 1%, de 0.664 a 0.655. En cambio, la precisión del clasificador ha bajado de forma considerable, cerca de un 10%, de 0.588 a 0.499.

Bibliografía

Python: qué es y por qué deberías aprender a utilizarlo. (s/f). Becas-santander.com. Recuperado el 21 de abril de 2022, de https://www.becas-santander.com/es/blog/python-que-es.html

Varoquaux, G., Buitinck, L., Louppe, G., Grisel, O., Pedregosa, F., & Mueller, A. (2015). Scikit-learn: Machine learning without learning the machinery. GetMobile Mobile Computing and Communications, 19(1), 29–33. https://doi.org/10.1145/2786984.2786995

Redacci, Espa, O., & ntilde;a. (2020, 9 de julio). ¿Qué es Weka y qué tiene que ver con Big Data? Agenciab12.com. https://agenciab12.com/noticia/que-es-weka-que-tiene-que-ver-big-data

Bugarin, J. L. (s/f). Convertir Excel o CSV a formato ARFF en WEKA. Consultorjava.com. Recuperado el 21 de abril de 2022, de https://consultorjava.com/blog/convertir-excel-o-csv-a-formato-arff-en-weka/

Curvas ROC. (s/f). Inf.um.es. Recuperado el 21 de abril de 2022, de http://ares.inf.um.es/00Rteam/pub/mamutCola/modulo2.html

David, D. (2021, julio 20). Automatic feature Selection in python: An essential guide. Hackernoon.Com. https://hackernoon.com/automatic-feature-selection-in-python-an-essential-guide-uv3e37mk

Statistical functions (scipy.stats) — SciPy v1.8.0 Manual. (s/f). Scipy.org. Recuperado el 28 de abril de 2022, de https://docs.scipy.org/doc/scipy/reference/stats.html

Análisis de normalidad con Python. (s/f). Cienciadedatos.net. Recuperado el 28 de abril de 2022, de https://www.cienciadedatos.net/documentos/pystats06-analisis-normalidad-python.html

scipy.stats.brunnermunzel — SciPy v1.8.0 Manual. (s/f). Scipy.org. Recuperado el 30 de abril de 2022, de https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.brunnermunzel.html

Sánchez, C. A. (s/f). Tema 3. Contraste de la normalidad multivariante. Usc.es. Recuperado el 30 de abril de 2022, de http://eio.usc.es/eipc1/base/BASEMASTER/FORMULARIOS-PHP/MATERIALESMASTER/Mat_142400_mmulti1011tema3.pdf