Práctica Final
Curso 2021-2022

Martínez Mañas , Juan José.
Tercer año Ingeniería biomédica.

Índice.

Introducción

En la práctica final de Análisis de Datos, se van a estudiar, limpiar y manipualr una serie de datos recogidos en el fichero 'diabetic_data.csv', donde se recogen los datos de un estudio sobre readmisión de pacientes con diabetes. Sepuede automatizar el proceso de manipular todos estos datos con Python que será el lenguaje que se utilizará durante toda la práctica. Cabe destacar que el nombre que se le asignará a el dataset será de df durante toda la práctica.

Librerías.

Las librerías en Python contienen un conjunto de herramientas (llamadas funciones) que hacen tareas en nuestros datos. Una vez que la biblioteca está instalada, puede ser usada y llamadas para hacer muchas tareas. Las que fueron usadas en la práctica se resumen en:

- **Numpy:** Como paquete básico de manipulación de datos vectoriales.
- **Panda:** Como paquete básico de manipulación de datos de diferentes tipos y con estructura de tabla también sirve para representar gráficamente su contenido, o analizar estadísticamente de una forma sencilla. Está construido sobre NumPy y MatPlotLib.
- **MatPlotLib**: Es el paquete de representaciones gráficas más importante en Python. Tiene una orientación descriptiva donde se necesita indicar, el cómo, y necesita definir cada parte del gráfico con código. Existe otra orientación de alto nivel llamada declarativa como la del paquete Plotnine que permite definir, el qué, y no necesita especificar cada parte del gráfico.
- **Scikit-learn:** Es el principal paquete de aprendizaje automático (Machine Learning) de propósito general en Python. Tiene gran cantidad de algoritmos y módulos para el pre-procesamiento, validación cruzada y ajuste de hiper-parámetros de modelos, clasificación/regresión, etc.

Manipulación de los datos.

A la hora de seguir un orden establecido con el que poder preparar y manipular los datos guardados se ha seguido de guía ciertos apartados del libro 'Introducción a la Minería de Datos', escrito por José Hernández Orallo, Mª José Ramírez Quintana y Cèsar Ferri Ramírez, entre otras muchas fuentes indicadas en la bibliografía.

Martínez Mañas , Juan José. *jjmm23@gcloud.ua.es*

Limpieza de los datos.

Uno de los aspectos más importantes al trabajar con datos, consiste en el pre-procesamiento de estos. La limpieza de datos es un paso necesario antes de realizar cualquier tarea relevante en análisis/ciencia de datos, desde elaborar un simple análisis gráfico, hasta implementar un modelo de aprendizaje automático. Como bien se expone en el 'Capítulo 4 , Limpieza y Transformación.', *los procesos de limpieza reciben nombres bastante variados: preparación de datos, data cooking, preprocesamiento, etc. Conjuntamente, la preparación de datos tiene como objetivo la eliminación del mayor número posible de datos erróneos o inconsistentes (limpieza) e irrelevantes (criba), y trata de presentar los datos de la manera más apropiada para la minería de datos.*

Primera Limpieza

Trabajar datos Nulos

Algunos métodos de minería de datos que se utilizan pueden no tratar bien los campos faltantes. Los campos nulos, pueden deberse a falta de información o bien a no haber podido estimar la variable, un error de entrada, o bien hubo un fenómeno externo que impidió la recolección de datos. Para detectar estos campos faltantes y tratarlos, se realizó de varias formas:

- De manera visual, los datos al proceden de una base de datos, solo bastó con mirar en la tabla de resumen de atributos/características y ver la cantidad de nulos que tiene cada atributo.
- De una manera más exacta utilizando funciones de pandas como df.isnull().sum(), que devuelve el número de valores faltantes en el conjunto de datos.

El resultado de la última interacción quedaría en resumidas cuentas tal que así:

```
encounter_id                0
patient_nbr                 0
race                     2273
gender                      0
age                         0
weight                  98569
admission_type_id           0
discharge_disposition_id    0
admission_source_id         0
time_in_hospital            0
payer_code              40256
medical_specialty       49949
num_lab_procedures          0
num_procedures              0
num_medications             0
number_outpatient           0
number_emergency            0
number_inpatient            0
diag_1                     21
diag_2                    358
diag_3                   1423
```

En este caso como se muestra en la imagen anterior, existen varios campos donde los nulos comprenden un gran número de los datos totales, como son las columnas de weight, payer_code, medical_specialty o race. Estas serán las primeras columnas que se modificarán en una primera limpieza.

Borrado de las columnas y filas con nulos.

Para tratar estos casos en primer lugar se borró las columnas conflictivas que se mencionaron en el apartado anterior con la función de pandas df.pop(). Tras estos cambios se puede verificar si se han borrado las columnas con display df.head().

Aún así al estudiar denuevo los nulos vuelven a salir ciertos valores en algunas columnas, como se muestra en la siguiente imagen:

```
(101766, 46)
encounter_id                0
patient_nbr                 0
gender                      0
age                         0
admission_type_id           0
discharge_disposition_id    0
admission_source_id         0
time_in_hospital            0
num_lab_procedures          0
num_procedures              0
num_medications             0
number_outpatient           0
number_emergency            0
number_inpatient            0
diag_1                     21
diag_2                    358
diag_3                   1423
number_diagnoses            0
max_glu_serum               0
A1Cresult                   0
```

Para poder solventar esto se ha usado la función dropna() de pandas con la que elimina los valores nulos del conjunto de datos, dejando caer las filas o columnas que contienen los valores nulos.

Cambiar los tipos de datos.

En el fichero de diabetes a analizar, se puede observar que muchas de las variables que encontramos son categóricas, es decir, variables no numéricas que adquieren valores de un número limitado de clases o categorías. Estas variables categóricas pueden ser de dos tipos:

- **Variables ordinales**: sus valores pueden ser ordenados jerárquicamente, como, por ejemplo, el nivel de educación de una persona , o en los datos propuestos el 'age', que está ordenado según unos intervalos((0-10),(10,20)...)
- **Variables nominales:** no se puede establecer un orden en sus categorías. Algunos ejemplos en el conjunto de datos son el gender (male or female) o race (Caucasian, AfricanAmerican ,etc.).

El problema es que la mayoría de los modelos de machine learning en Python, entre ellos las redes neuronales, solo pueden leer valores numéricos puesto que hacen uso de diversas operaciones matemáticas. Por esto mismo, en estos casos es importante convertir este tipo de variables en numéricas para que los modelos puedan utilizarlas.

Antes de comenzar con el cambio en primer lugar vamos a estudiar los diferentes tipos de datos de nuestro conjunto de datos gracias a la función de pandas df.dtypes, lo que genera una lista con los diferentes tipos de datos tal que así:

encounter_id	int64
patient_nbr	int64
race	object
gender	object
age	object
weight	object
admission_type_id	int64
discharge_disposition_id	int64
admission_source_id	int64
time_in_hospital	int64
payer_code	object
medical_specialty	object
num_lab_procedures	int64
num_procedures	int64
num_medications	int64
number_outpatient	int64
number_emergency	int64
number_inpatient	int64
diag_1	object
diag_2	object
diag_3	object
number_diagnoses	int64
max_glu_serum	object
A1Cresult	object
metformin	object
repaglinide	object
nateglinide	object
chlorpropamide	object
glimepiride	object
acetohexamide	object
glipizide	object
glyburide	object
tolbutamide	object
pioglitazone	object
rosiglitazone	object
acarbose	object
miglitol	object
troglitazone	object
tolazamide	object
examide	object
citoglipton	object
insulin	object
glyburide-metformin	object
glipizide-metformin	object
glimepiride-pioglitazone	object
metformin-rosiglitazone	object
metformin-pioglitazone	object
change	object
diabetesMed	object
readmitted	object

En el análisis podemos ver que existen varios tipos diferentes de datos desde int64(que luego modificaremos), hasta objetos.

- Para tratar los datos objeto a numéricos en primer lugar se utilizó la función de pandas display(df[].unique()), con la que conocer los diferentes valores para cada columna. En el caso de la raza el resultado sería el siguiente:

  array(['Caucasian', 'AfricanAmerican', nan, 'Other', 'Asian', 'Hispanic']

  Conociendo los diferentes valores los podemos modificar asignados a un entero gracias a la función de pandas df.replace. Pasándole los valores como parámetro se pueden modificar las referencias.

  ```
  valores={"race":{'Caucasian':1, 'AfricanAmerican':2, 'Other':3,'Asian':4,
  'Hispanic':5}}
  df.replace(valores,inplace=True)
  ```

  Este paso se repetirá con todas las columnas de clase objeto del conjunto de datos. Tras realizar estos cambios al mostrar esa columna se mostrarán los valores numéricos asignados como se muestra a continuación:

  ```
        dtype=object)
  0         1.0
  1         1.0
  2         2.0
  3         1.0
  4         1.0
            ...
  101761    2.0
  101762    2.0
  101763    1.0
  101764    1.0
  101765    1.0
  Name: race, Length: 101766, dtype: float64
  ```

- Mediante un pequeño bucle se cambia el tipo de dato numérico de int64 a int 32. Esto se realiza ya que en muchos casos se pueden realizar más proyectos con mayor compatibilidad.

  ```
  pares = {'int64':'int32', 'float64':'float32'}
  for i in df.columns:
    pareja = pares.get(str(df[i].dtype))
    if pareja != None:
      df[i]= df[i].astype(pareja)

  df.dtype
  ```

Al ejecutar de nuevo data.types para conocer los tipos de datos nos muestra la siguiente información cambiada:

```
metformin                 int32
repaglinide               int32
nateglinide               int32
chlorpropamide            int32
glimepiride               int32
acetohexamide             int32
glipizide                 int32
glyburide                 int32
tolbutamide               int32
pioglitazone              int32
rosiglitazone             int32
acarbose                  int32
miglitol                  int32
troglitazone              int32
tolazamide                int32
examide                   int32
citoglipton               int32
```

- Por último pasamos las variables objetivo a categórica, en este caso la variable a estudiar 'readmitted', mediante el comando:

```
df['readmitted'] = df['readmitted'].astype('int64').astype('category')
```

Creación del vector plano.

Para poder usar los clasificadores se necesita crear un vector plano con todos los datos y aparte un vector plano con la variable target, en este caso 'readmitted'. Se hace de la siguiente manera:

```
X =
df[['encounter_id','patient_nbr','gender','age','admission_type_id','di
scharge_disposition_id',

'admission_source_id','time_in_hospital','num_lab_procedures','num_proc
edures','num_medications','number_outpatient',

'number_emergency','number_inpatient','diag_1','diag_2','diag_3',

'number_diagnoses','max_glu_serum','A1Cresult','metformin','repaglinide
','nateglinide','chlorpropamide',

'glimepiride','acetohexamide','glipizide','glyburide','tolbutamide','pi
oglitazone','rosiglitazone','acarbose','miglitol',

'troglitazone','tolazamide','examide','citoglipton','insulin','glyburid
e-metformin','glipizide-metformin',

'glimepiride-pioglitazone','metformin-rosiglitazone','metformin-pioglit
azone','metformin-pioglitazone','change','diabetesMed']]

y = df[['readmitted']].values.ravel()
```

Primer estudio.

Tras una primera limpieza de los datos se procedió a hacer una pequeña prueba con los clasificadores para ver si los resultados eran los deseados. Para ello se usó una lista de clasificadores con los que calcular los resultados y conseguir la media de ellos. Esta lista usa clasificadores desde el Random Forest hasta el AdaBoost, pasando por los Nearest Neighbors. Para el estudio utilizaremos model_selection que proporciona índices de entrenamiento/prueba para dividir los datos en conjuntos de entrenamiento/prueba. El resultado de esta medición quedaría tal que así:

```
Baseline              media aciertos: 0.54 resultados: [0.54 0.54 0.54 0.54 0.54 0.54 0.54 0.54 0.54 0.54]
Nearest Neighbors (1) media aciertos: 0.46 resultados: [0.46 0.45 0.46 0.46 0.46 0.45 0.45 0.46 0.46 0.45]
Nearest Neighbors (3) media aciertos: 0.51 resultados: [0.51 0.51 0.5  0.52 0.5  0.51 0.5  0.5  0.51 0.5 ]
Nearest Neighbors (5) media aciertos: 0.52 resultados: [0.52 0.52 0.52 0.53 0.52 0.53 0.52 0.52 0.52 0.52]
Nearest Neighbors (7) media aciertos: 0.53 resultados: [0.52 0.52 0.53 0.53 0.53 0.53 0.53 0.52 0.52 0.53]
Decision Tree         media aciertos: 0.49 resultados: [0.49 0.49 0.5  0.49 0.49 0.49 0.49 0.48 0.49 0.49]
Random Forest         media aciertos: 0.60 resultados: [0.6  0.6  0.6  0.6  0.6  0.6  0.61 0.6  0.59 0.6 ]
Neural Net            media aciertos: 0.47 resultados: [0.54 0.54 0.45 0.35 0.5  0.35 0.35 0.54 0.54 0.54]
AdaBoost              media aciertos: 0.59 resultados: [0.59 0.59 0.59 0.59 0.6  0.59 0.59 0.59 0.58 0.59]
Naive Bayes           media aciertos: 0.54 resultados: [0.54 0.54 0.54 0.54 0.54 0.54 0.54 0.54 0.54 0.54]
Logistic Regression   media aciertos: 0.54 resultados: [0.54 0.53 0.54 0.54 0.54 0.54 0.54 0.54 0.54 0.54]
```

Como se buscan valores mayores procedemos a una segunda limpieza antes de proceder con los demás clasificadores.

Segunda limpieza

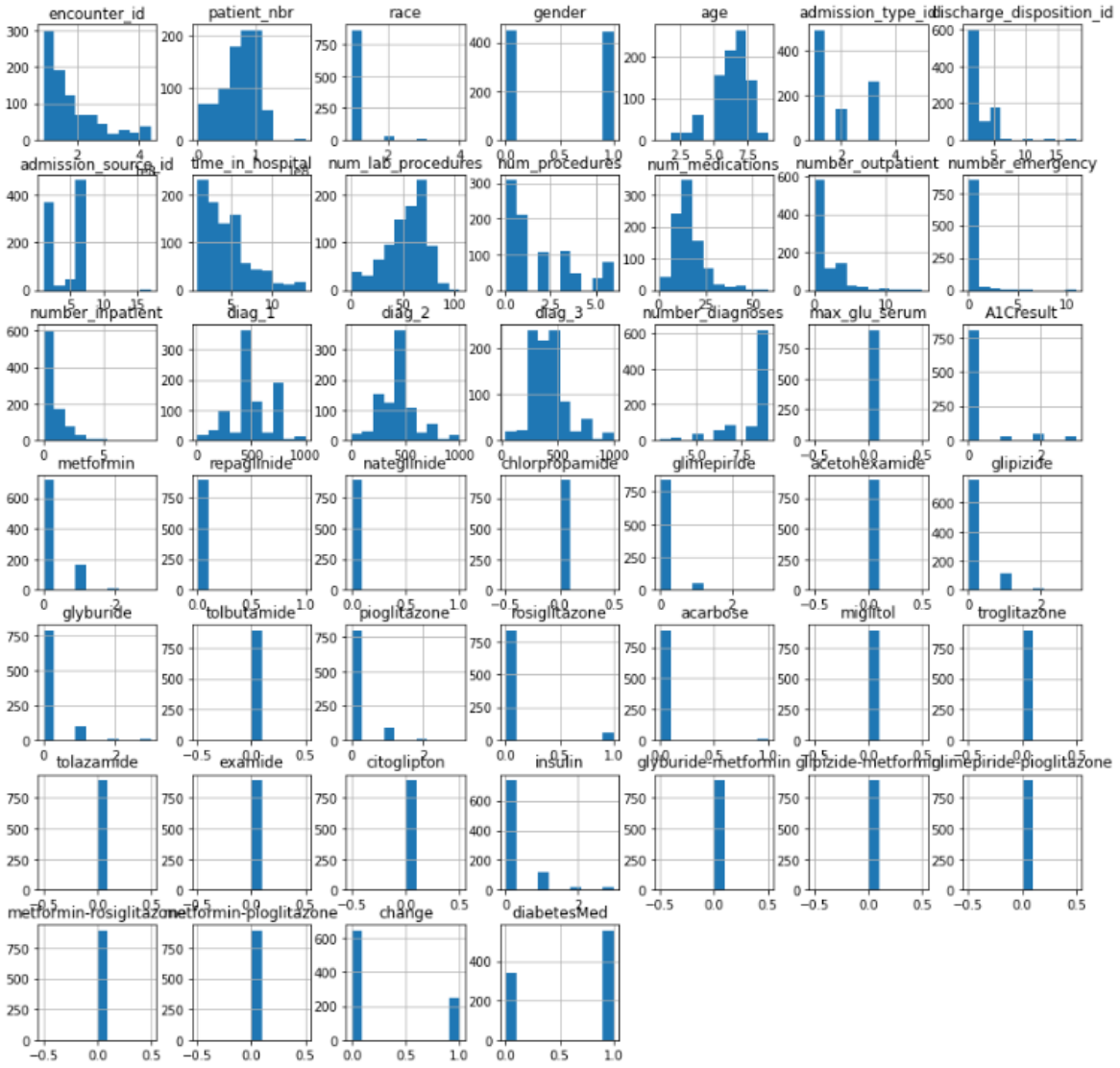
- En primer lugar se estudió los valores de las columnas más conflictivas de las que quedaban y se descubrió que las columnas de diag_1, diag_2 y diag_3 existían valores erróneos, una mezcla de enteros con daots objetivos y demás por eso se procedió a borrar esos datos mediante la función de pandas:

  df['diag_1']=pd.to_numeric(df['diag_1'],errors='coerce',downcast='float')

  En este caso eliminar los valores erróneos y pasar los datos de object a float. Mediante el comando errores=coerce se indica que se quiere tener en cuenta los valores erróneos que encuentre y los tome como nulos. Después se puede hacer un df=df.dropna(how='any') para que borre los nulos y pueda así eliminar por todas esos valores.

  Finalmente se decidió eliminar estas columnas en función de mejorar los resultados.

- Después se probó con los histogramas. Mediante la función df.hist(figsize=(15,15)) en este caso se ajustó el tamaño del mismo por motivos de presentación. El resultado quedó tal que así:



El histograma tiene en cuenta la cantidad de veces que aparece cada valor para cada dato por eso se optó por eliminar las columnas que tengan una diferencia de valores muy elevadas como el caso de number _ outpatient, acarbose o miglitol entre otras.

Aún así se buscó limpiar las filas en su mayoría tambíen utilizando la función df.drop_duplicates() con el que eliminar las filas con los datos duplicados del conjunto de datos.

```
df.drop_duplicates(['patient_nbr'], keep='first', inplace=True)
df.drop_duplicates(['encounter_id'], keep='first', inplace=True)
df.shape
```

El resultado de esta criba lo podemos ver con la función df.shape, donde se muestran el tamaño del Data set. El resultado sería tal que así:

```
df.shape
(65740, 16)
```

Como resultado vemos que el dataset que se va a clasificar se redujo en 64520 filas y 16 columnas.

Clasificación

Tras varios estudios y limpieza del Dataset se procedió a la clasificación final de los datos. En primer lugar se prepararon los datos de nuevo con el vector plano:

```
X=df[['encounter_id','patient_nbr','gender','age','admission_type
_id',

'admission_source_id','time_in_hospital','num_lab_procedures','nu
m_procedures','num_medications',

'number_diagnoses','A1Cresult','insulin','change','diabetesMed']]

y = df[['readmitted']].values.ravel()
```

En primer lugar se va a calcular la cantidad de aciertos para una serie de clasificadores de distintas naturalezas. Estos clasificadores se recogen en una lista que iremos recorriendo y calculando estos resultados y sus correspondientes medias de los aciertos tras hacer la validación cruzada. El objetivo de la validación cruzada es garantizar que los resultados que se obtengan sean independientes de la partición entre datos de entrenamiento y datos de validación, y por eso se usa mucho en para validar modelos generados en proyectos de matching learning e IA.
La lista de clasificador a utilizar sería la siguiente:

```
classifiers_list = [
  ( 'Baseline',              DummyClassifier() ),
  ( 'Nearest Neighbors (1)', KNeighborsClassifier(1) ),
  ( 'Nearest Neighbors (3)', KNeighborsClassifier(3) ),
  ( 'Nearest Neighbors (5)', KNeighborsClassifier(5) ),
  ( 'Nearest Neighbors (7)', KNeighborsClassifier(7) ),
  ( 'Decision Tree',         DecisionTreeClassifier() ),
  ( 'Random Forest',         RandomForestClassifier() ),
  ( 'Neural Net',            MLPClassifier() ),
  ( 'AdaBoost',              AdaBoostClassifier() ),
  ( 'Naive Bayes',           GaussianNB() ),
  ( 'Logistic Regression',   LogisticRegression() ),
]
```

Mediante el siguiente bucle podemos calcular los aciertos del conjunto de datos con una validación cruzada de 10, para cada clasificador de la lista:

```
cv = model_selection.StratifiedKFold(n_splits=10, random_state=123,
shuffle=True)
for name, model in classifiers_list:
  results = np.round(model_selection.cross_val_score(model, X, y,
cv=cv),2)
  print(f'{name:22s}  resultados: {results}')
```

El resultado de la cross validation sería el siguiente:

```
Baseline               resultados: [0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6
0.6 0.6]
Nearest Neighbors (1)  resultados: [0.5  0.5  0.5  0.51 0.5  0.5  0.5
0.5  0.5  0.5 ]
Nearest Neighbors (3)  resultados: [0.56 0.56 0.55 0.56 0.56 0.56 0.56
0.55 0.55 0.55]
Nearest Neighbors (5)  resultados: [0.58 0.57 0.57 0.58 0.58 0.57 0.58
0.58 0.57 0.57]
Nearest Neighbors (7)  resultados: [0.58 0.58 0.58 0.58 0.59 0.58 0.58
0.58 0.57 0.58]
Decision Tree          resultados: [0.5  0.5  0.5  0.49 0.5  0.5  0.5
0.5  0.51 0.5 ]
Random Forest          resultados: [0.61 0.61 0.61 0.62 0.62 0.61 0.61
0.61 0.61 0.62]
Neural Net             resultados: [0.31 0.31 0.58 0.6  0.58 0.38 0.59
0.55 0.31 0.54]
AdaBoost               resultados: [0.61 0.61 0.61 0.61 0.61 0.61 0.61
0.61 0.61 0.61]
Naive Bayes            resultados: [0.6 0.6 0.6 0.6 0.6 0.6 0.6 0.6
0.6 0.6]
Logistic Regression    resultados: [0.6  0.6  0.6  0.6  0.6  0.6  0.6
0.59 0.6  0.6 ]
```

	tipos de clasificadores	Media de los aciertos
1	'Baseline'	0.6
2	'Nearest Neighbors (1)'	0.5
3	'Nearest Neighbors (3)'	0.56
4	'Nearest Neighbors (5)'	0.57
5	'Nearest Neighbors (7)'	0.58
6	'Decision Tree'	0.5
7	'Random Forest'	0.61
8	Neural Net'	0.46
9	'AdaBoost'	0.61
10	'Naive Bayes'	0.6
11	'Logistic Regression'	0.6

Tabla 1/ Media de los aciertos.

La siguiente medición que se llevó a cabo fue medir el área de la curva ROC. La curva ROC es una gráfica que enfrenta el ratio de falsos positivos (eje x) con el ratio de falsos negativos (eje y). El área descrita por esta curva es interesante para sacar conclusiones sobre los datos. El método que se llevará a cabo para conocer los valores del área será similar al código efectuado con anterioridad, cálculo del área bajo la curva ROC sobre una validación cruzada 10 :

```
cv = model_selection.StratifiedKFold(n_splits=10, random_state=123,
shuffle=True)
for name, model in classifiers_list:
  results = np.round(model_selection.cross_val_score(model, X, y,
scoring='roc_auc_ovr', cv=cv),2)
  print(f'{name:22s}  resultados: {results}')
```

```
Baseline               resultados: [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5]
Nearest Neighbors (1)  resultados: [0.53 0.52 0.53 0.53 0.53 0.53 0.53
0.52 0.53 0.53]
Nearest Neighbors (3)  resultados: [0.55 0.55 0.55 0.55 0.56 0.55 0.55
0.55 0.54 0.55]
Nearest Neighbors (5)  resultados: [0.58 0.56 0.56 0.56 0.57 0.57 0.55
0.56 0.55 0.55]
Nearest Neighbors (7)  resultados: [0.59 0.57 0.57 0.57 0.58 0.57 0.56
0.57 0.56 0.56]
Decision Tree          resultados: [0.53 0.54 0.54 0.53 0.53 0.53 0.54
0.54 0.54 0.53]
Random Forest          resultados: [0.63 0.63 0.63 0.64 0.63 0.62 0.62
0.63 0.63 0.63]
Neural Net             resultados: [0.5  0.51 0.5  0.5  0.54 0.48 0.51
0.51 0.5  0.5 ]
AdaBoost               resultados: [0.64 0.64 0.63 0.64 0.63 0.62 0.62
0.64 0.63 0.63]
Naive Bayes            resultados: [0.6  0.59 0.58 0.59 0.59 0.59 0.59
0.57 0.58 0.59]
Logistic Regression    resultados: [0.58 0.57 0.56 0.57 0.58 0.57 0.57
0.56 0.56 0.57]
```

	tipos de clasificadores	Media de los valores del AUC
1	'Baseline'	0.5
2	'Nearest Neighbors (1)'	0.53
3	'Nearest Neighbors (3)'	0.55
4	'Nearest Neighbors (5)'	0.56
5	'Nearest Neighbors (7)'	0.57
6	'Decision Tree'	0.54
7	'Random Forest'	0.63
8	Neural Net'	0.51
9	'AdaBoost'	0.63
10	'Naive Bayes'	0.59
11	'Logistic Regression'	0.57

Tabla 2/ Media de los valores de AUC

Los valores recogidos en la tabla anterior representan las medias de los valores de AUC para cada clasificador. Esta información se utiliza como resumen del rendimiento del modelo.

Estos datos guardan la relación de que cuanto más esté hacia la izquierda la curva, más área habrá contenida bajo ella y por ende, mejor será el clasificador, ya que los valores a la izquierda representan los valores de diagnóstico perfecto. Por ello el clasificador perfecto tendría un AUC de 1, y los clasificadores más acertados deberían acercarse a este valor y no a valores como 0,5. En el último caso, el modelo no tendrá la suficiente capacidad de descriminación para distinguir entre las clases.

Ajustándose a los resultados los valores más cercanos a la unidad son los calculados por los clasificadores de Adaboost y Random Forest, los cuales han conseguido distinguir de mejor manera la diferencia entre las clases positivas y negativas.

A continuación se va a tratar el desbalanceamiento de clases con **Oversampling**. En muestras desbalanceadas, donde existen clases que contienen más registros que otros y es necesario balancear los registros es necesario usar estas técnicas. En este caso vamos a utilizar Oversampling que en lugar de tomar una muestra de los datos mayoritarios se duplican los datos minoritarios hasta que se lleguen a equilibrar. El código es el siguiente, similar a l anterior una validación cruzada utilizando pipeline:

```
# Oversampling

cv = model_selection.StratifiedKFold(n_splits=10, random_state=123,
shuffle=True)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

for name, model in classifiers_list:
  pipe = Pipeline([('oversampling', RandomOverSampler()), ('Model',
model)])
  results = np.round(model_selection.cross_val_score(pipe, X, y,
scoring='roc_auc_ovr', cv=cv),2)
  print(f'{name:22s} resultados: {results}')
```

El resultado de este estudio sería el siguiente:

```
Baseline               resultados: [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5]
Nearest Neighbors (1)  resultados: [0.53 0.52 0.53 0.53 0.53 0.53 0.53
0.52 0.53 0.53]
Nearest Neighbors (3)  resultados: [0.54 0.54 0.54 0.54 0.55 0.54 0.54
0.54 0.54 0.54]
Nearest Neighbors (5)  resultados: [0.56 0.55 0.54 0.55 0.55 0.55 0.54
0.54 0.54 0.54]
Nearest Neighbors (7)  resultados: [0.56 0.55 0.55 0.55 0.56 0.56 0.54
0.54 0.55 0.54]
Decision Tree          resultados: [0.53 0.53 0.53 0.54 0.54 0.53 0.53
0.53 0.53 0.53]
Random Forest          resultados: [0.63 0.63 0.63 0.64 0.63 0.62 0.61
0.63 0.63 0.62]
Neural Net             resultados: [0.51 0.54 0.5  0.5  0.5  0.51 0.5
0.5  0.52 0.52]
```

```
AdaBoost                resultados: [0.64 0.64 0.63 0.64 0.63 0.63 0.62
0.63 0.63 0.63]
Naive Bayes             resultados: [0.59 0.58 0.57 0.59 0.59 0.58 0.59
0.57 0.58 0.58]
Logistic Regression     resultados: [0.58 0.59 0.58 0.59 0.59 0.59 0.58
0.58 0.58 0.59]
```

	tipos de clasificadores	Media de los valores del AUC tras oversampling
1	'Baseline'	0.5
2	'Nearest Neighbors (1)'	0.53
3	'Nearest Neighbors (3)'	0.54
4	'Nearest Neighbors (5)'	0.55
5	'Nearest Neighbors (7)'	0.55
6	'Decision Tree'	0.53
7	'Random Forest'	0.63
8	Neural Net'	0.51
9	'AdaBoost'	0.63
10	'Naive Bayes'	0.58
11	'Logistic Regression'	0.58

Tabla 2/ Media de los valores de AUC tras el Oversampling.

Los resultados no varían en gran medida, incluso en algunos clasificadores disminuyen el valor, por eso se decidió utilizar los datos sin esta agrupación. Esta poca diferencia puede ser causada por la anterior limpieza de los datos donde se eliminaron las columnas que podían enriquecerse de un futuro Oversampling.

Una de las funciones hábiles en python es la posibilidad de estudiar las modificación de los parámetros por defecto para mejorar los resultados para ello se va a utilizar la función del paquete de scikit-learn que tiene funciones específicas como Grid Search CV para realizar pruebas exhaustivas de parámetros. Gracias a él se puede evaluar y seleccionar de forma sistemática los parámetros de un modelo. Indicando un modelo y los parámetros a probar, puede evaluar el rendimiento del primero en función de los segundos. La lista de parámetros a modificar sería:

```
params= {'max_features':['auto''log2'],'max_depth':[10,6,7]}
```

La sentencia sería tal que así:

```
grid_search.fit(X_train, y_train)
cv_score=grid_search.best_score_
test_score = grid_search.score(X_test, y_test)
print(f'AUC score: {cv_score}')
print(f'Accuracy:{test_score}')
```

El resultado es el mejor porcentaje conseguido de las modificaciones de los parámetros introducidos al clasificador , en este caso DecisionTreeClassifier(). En este caso sería:

```
Accuracy:0.6030650997116903
```

Test de Wilcoxon

Para completar el análisis se utilizaron varios test estadísticos para comprobar los resultados de los clasificadores entre sí y conseguir distinguir el mejor clasificador para los datos introducidos. Para ello se utilizó el test de Wilcoxon. El test es un test estadístico no paramétrico que permite contrastar la hipótesis nula de que las medias de dos poblaciones son iguales, frente a la hipótesis alternativa de que no lo son.

El método utiliza los 10 resultados de AUC que obtenemos de la cross validation para 2 experimentos, y compara estos 10 resultados de las 10 particiones, con los del otro experimento para considerar cual es más acertado.

En mi caso vamos a comprar los 4 clasificadores con más media a la hora de calcular la AUC. El estudio será de cada clasificador con todos los restantes.
Las sentencias serían tal que así, únicamente modificando el array de AUCs de cada clasificador.

```
x=np.array([0.6 , 0.6 , 0.69, 0.65, 0.61 ,0.57, 0.66 ,0.62, 0.62
,0.48])
y=np.array([0.57 ,0.58 ,0.66, 0.5 , 0.57 ,0.54 ,0.54, 0.55, 0.62,
0.49])
warnings.filterwarnings('ignore')
wilcox_V, p_value =  wilcoxon(x, y, alternative='greater',
zero_method='wilcox', correction=False)
print('Resultado completo del test de Wilcoxon')
print(f'Wilcox V: {wilcox_V}, p-value: {p_value:.2f}')
```

Lo que nos interesa de este método es el p_value. Esta variable recoge el porcentaje de solapamiento. Si este valor es mayor que 0.05 se acepta la hipótesis de que son iguales, y se determina que la información en la variable X es más confiable que en Y.
A continuación se muestra una tabla con los resultados de enfrentar ambos resultados de los clasificadores al test:

	tipos de clasificadores	tipos de clasificadores a comparar	P_value
1	Nearest Neighbors (7)		
		Random Forest	1.00
		AdaBoost	1.00
		Naive Bayes	1.00
2	Random Forest		
		Nearest Neighbors (7)	0.00
		AdaBoost	0.96
		Naive Bayes	0.00
3	AdaBoost		
		Nearest Neighbors (7)	0.00
		Random Forest	0.04
		Naive Bayes	0.00
4	Naive Bayes		
		Nearest Neighbors (7)	0.00
		Random Forest	1.00
		AdaBoost	1.00

Tabla 3/ Recopilación de los valores de semejanza.

Lo interesante de estos resultados son los menores de 0.05 como es el caso de comparar Adaboost vs Random forest, son muy similares.

Bibliografía Consultada.

- Introducción a la Minería de Datos - Hernández, Ramírez, Ferri.pdf
- https://datacarpentry.org/python-ecology-lesson-es/02-starting-with-data/
- https://www.analyticslane.com/2021/07/15/pandas-cambiar-los-tipos-de-datos-en-los-dataframes/
- https://interactivechaos.com/es/python/function/sklearnmodelselectiontraintestsplit#:~:text=La%20funci%C3%B3n%20sklearn.,el%20nombre%20de%20la%20funci%C3%B3n)
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

- https://www.cienciadedatos.net/documentos/py06_machine_learning_python_scikitlearn.htm

- https://www.themachinelearners.com/curva-roc-vs-prec-recall/