

Máster Universitario en Ingeniería Informática

TRATAMIENTO INTELIGENTE DE DATOS

**PRÁCTICA 1: Preprocesamiento
ACCIDENTES**



**UNIVERSIDAD
DE GRANADA**



Carlos Santiago Sánchez Muñoz

Grupo de prácticas 1 - Jueves

Email: carlossamu7@correo.ugr.es

4 de noviembre de 2020

Índice

1. Descripción del problema	2
1.1. Atributos imputados de la base de datos	2
1.2. Construcción de la variable a predecir	3
2. Discretización	4
3. Valores perdidos	6
3.1. Imputando valores con media y moda	6
3.2. Eliminando instancias	7
3.3. Eliminando atributos	7
4. Selección de características	8
5. Selección de instancias	9
5.1. Muestreo	9
5.2. Balanceando los datos	9
6. Resultados	11
6.1. Datos sin imputar	11
6.2. Datos imputados	12

1. Descripción del problema

En EE.UU., el General Estimate System (GES) es un componente del National Automotive Sampling System (NASS) mantenido por la Administración Nacional de Seguridad del Tráfico en Carreteras. El GES obtiene sus datos de una muestra representativa a nivel nacional de los aproximadamente 6,4 millones de accidentes informados por la policía que se producen anualmente. Estos accidentes incluyen aquellos que resultan mortales o causan lesiones y los relacionados con daños materiales. Al restringir la atención a los accidentes informados por la policía, el GES se concentra en los accidentes de mayor preocupación para la comunidad de seguridad vial y el público en general. GES se utiliza para identificar áreas con problemas de seguridad vial, proporcionar una base para iniciativas de información al consumidor y normativas, así como facilitar el análisis de costes y beneficios de las iniciativas de seguridad en carretera.

En esta práctica, utilizaremos como ejemplo el conjunto de datos tomado como muestra en el año 2001 (55.964 datos). Para facilitar las pruebas durante la realización de la práctica, dada la cantidad de datos, se facilitará el archivo completo .xls, y una versión con un número reducido de instancias. Ambos archivos .xls contienen dos hojas: una, con la descripción de las variables y los posibles valores que pueden tomar, y, otra, con los datos recogidos.

El objetivo del problema es predecir la gravedad del daño causado en el accidente en función de una serie de características como la ingesta de alcohol del conductor, hora del accidente, condición de la carretera, etc. Este tipo de predicción podría ser útil, por ejemplo, para priorizar la dotación de recursos en respuesta a un accidente. Sin embargo, los datos disponibles presentan importantes deficiencias (valores perdidos, características con excesivas categorías, características e instancias prescindibles, etc.) que sugieren la necesidad de ser preprocesados antes de aplicar otras técnicas de Minería de Datos.

1.1. Atributos imputados de la base de datos

Algunos atributos de esta base de datos tienen valores perdidos y existe un atributo análogo (normalmente contiene un _I). A la hora de hacer cualquier tarea, ya sea preprocesar o aplicar una técnica de clasificación se debe usar una sola de esas variables, nunca ambas. Las parejas de estos atributos son las siguientes:

Variable	Variable imputada	Variable	Variable imputada
WEEKDAY	WKDY_I	SUR_COND	SURCON_I
HOUR	HOUR_I	TRAF_CON	TRFCON_I
MAN_COL	MANCOL_I	SPD_LIM	SPDLIM_H
REL_JCT	RELJCT_I	LGHTE_CON	LGTCON_I
ALIGN	ALIGN_I	WEATHER	WEATHR_I
PROFILE	PROFIL_I	ALCOHOL	ALCHL_I

Tabla 1: Tabla de parejas variable-variable imputada

En mi caso manejaré dos conjuntos de datos a la vez, los normales y los imputados. A continuación muestro la lectura de estos datos y como se distingue entre la base de datos

con atributos sin imputar y los imputados.

```
""" Lectura de datos. Devuelve dos df, uno de ellos con las columnas imputadas.
- mini (op): indica si leer 'accidentes_mini'. Por defecto 'True'.
"""
def read_data(mini=True):
    if(mini):
        df = pd.read_excel(r'accidentes_mini.xls', sheet_name='datos')
    else:
        df = pd.read_excel(r'accidentes.xls', sheet_name='datos')

    df_I = df.copy()
    df = df.drop(columns = ['WKDY_I', 'HOUR_I', 'MANCOL_I', 'RELJCT_I', 'ALIGN_I',
                           'PROFIL_I', 'SURCON_I', 'TRFCON_I', 'SPDLIM_H',
                           'LGTCON_I', 'WEATHR_I', 'ALCHL_I'])
    df_I = df_I.drop(columns = ['WEEKDAY', 'HOUR', 'MAN_OOL', 'REL_JCT', 'ALIGN',
                               'PROFILE', 'SUR_COND', 'TRAF_CON', 'SPD_LIM',
                               'LGHT_CON', 'WEATHER', 'ALCOHOL'])
    if(IMPRIME_INFO):
        print()
        print(df.info())
        print()
        print(df_I.info())
    return df, df_I
```

1.2. Construcción de la variable a predecir

La construcción de esta variable, la que predice el accidente, se tiene que hacer en función de las variables FATALITIES, INJURY_CRASH y PRPTYDMG_CRASH. Tras hacer algunas pruebas desde Python y visualizar estas variables me di cuenta de que nunca las tres estaban activas. Por tanto sólo hay tres tipos de accidentes: daños materiales, lesiones y accidente mortal.

He llamado a dicha variable CRASH_TYPE. ¿Cómo construirla eficientemente en función de las tres anteriores? Pues bien las tres anteriores toman el valor 0 o 1 y como mucho sólo una está activada a la vez. La construcción más sencilla es la siguiente:

$$\text{CRASH_TYPE} = 0 \cdot \text{PRPTYDMG_CRASH} + 1 \cdot \text{INJURY_CRASH} + 2 \cdot \text{FATALITIES}$$

Trasladando esto a código y eliminando posteriormente dichas tres variables sería así:

```
""" Construyendo la variable de clase 'CRASH_TYPE'. Devuelve el df.
- df: dataframe.
"""
def construct_class_variable(df):
    df['CRASH_TYPE'] = df['INJURY_CRASH'] + 2*df['FATALITIES']
    return df.drop(columns = ['PRPTYDMG_CRASH', 'INJURY_CRASH', 'FATALITIES'])
```

2. Discretización

Algunas características contienen valores discretos según una escala ordenada y con suficiente cardinalidad como para plantearse la conveniencia de discretizarlos. De esto modo, al reducirlo a un menor número de posibles valores, simplificamos la clasificación haciendo también más sencillo el proceso de aprendizaje.

La primera variable que podemos discretizar es HOUR. Si la hora está en [7, 14] diremos que es por la mañana y le asignaremos un 0. Si está en [15, 23] diremos que es por la tarde y le asignaremos un 1 y si está fuera de esos intervalos es por la noche y le asignaremos un 2.

```
""" Discretizando la variable 'HOUR'. Devuelve el df.
- df: dataframe.
- imputed (op): si es el imputado. Por defecto 'False'.
"""

def discretize_HOUR(df, imputed=False):
    if imputed:
        attribute = 'HOUR_I'
    else:
        attribute = 'HOUR'
    for i in range(len(df[attribute])):
        # Si es por la mañana asigno un 0
        if(df[attribute][i]>=7 and df[attribute][i]<=14):
            df[attribute][i] = 0
        # Si es por la tarde asigno un 1
        elif(df[attribute][i]>=15 and df[attribute][i]<=23):
            df[attribute][i] = 1
        # Si es por la noche o es desconocido asigno un 2
        else:
            df[attribute][i] = 2
    return df
```

La segunda variable que se va a discretizar es WEEKDAY, el día de la semana. Si es entre semana (el número del día está en [2, 6]) le asigno un 0 y si es en fin de semana (fuera de ese intervalo) le asigno un 1.

```
""" Discretizando la variable 'WEEKDAY'. Devuelve el df.
- df: dataframe.
- imputed (op): si es el imputado. Por defecto 'False'.
"""

def discretize_WEEKDAY(df, imputed=False):
    if imputed:
        attribute = 'WKDY_I'
    else:
        attribute = 'WEEKDAY'
    for i in range(len(df[attribute])):
        # Si es entre semana asigno 0
        if(df[attribute][i]>=2 and df[attribute][i]<=6):
            df[attribute][i] = 0
        # Si es entre semana asigno 1
        else:
            df[attribute][i] = 1
    return df
```

La última variable que se va a discretizar es la velocidad, SPD_LIM. Teniendo en cuenta la distribución de los datos voy a distinguir entre una velocidad baja (menos de 40) a la que le asignaré un 0, una velocidad media (en el intervalo [44, 66)) en donde asigno un 1 y por último velocidad alta (mayor o igual a 66) en donde asigno un 2.

```

"""
    Discretizando la variable 'SPD_LIM'. Devuelve el df.
- df: dataframe.
- imputed (op): si es el imputado. Por defecto 'False'.
"""

def discretize_SPD_LIM(df, imputed=False):
    if imputed:
        attribute = 'SPDLIM_H'
    else:
        attribute = 'SPD_LIM'
    for i in range(len(df[attribute])):
        # Si es menor a 40 asigno 0
        if(df[attribute][i]<40):
            df[attribute][i] = 0
        # Si entre 40 y 65 asigno 1
        elif(df[attribute][i]>=40 and df[attribute][i]<66):
            df[attribute][i] = 1
        # Si es mayor a 65 asigno 2
        else:
            df[attribute][i] = 2
    return df

```

Estas tres discretizaciones se las voy a aplicar al conjunto de datos imputado.

3. Valores perdidos

El conjunto de datos contiene valores perdidos, es decir, valores desconocidos en algunas instancias para algunas características. En esta sección se van a examinar diferentes maneras para el tratamiento de valores perdidos. Este tratamiento se realiza sobre el conjunto de datos ya discretizado.

3.1. Imputando valores con media y moda

La primera forma de imputar valores va a ser usando la media o la moda. Se hace la media (o moda) del atributo en cuestión y se sustituye por el valor calculado. No obstante, es fundamental en este paso quitar de alguna manera los valores perdidos debido a que en este conjunto de datos el valor perdido tiene un valor numérico (normalmente 9 o 99) y no un NaN o NA por lo que computaría en el cálculo.

Para ello simplemente he reemplazado en el atributo en cuestión el valor perdido por `None`, un valor que la función media y moda en Python no es tenido en cuenta. Como detalle decir que el atributo `PED_ACC` tiene dos valores perdidos: 9998 y 9999. Simplemente se reemplaza previamente 9998 por 9999.

```
""" Imputando valores con la media. Devuelve el df.
- df: dataframe.

def imput_mean(df):
    atr_imputed = ['WEEKDAY', 'HOUR', 'MAN_COL', 'INT_HWY', 'REL_JCT', 'ALIGN',
                   'PROFILE', 'SUR_COND', 'TRAF_CON', 'SPD_LIM',
                   'LGHT_CON', 'WEATHER', 'PED_ACC', 'ALCOHOL']
    unknown_val = [9, 99, 99, 9, 99, 9, 9, 9, 99, 99, 9, 9, 9999, 9]
    df['PED_ACC'] = df['PED_ACC'].replace(to_replace = 9998, value = 9999)
    for i in range(len(atr_imputed)):
        df[atr_imputed[i]] = df[atr_imputed[i]].replace(to_replace = unknown_val[i],
                                                       value = None)
        mean = int(df[atr_imputed[i]].mean())
        if (IMPRIME_INFO):
            print(" Cambiando en {} el valor {} por su media: {}"
                  .format(atr_imputed[i], unknown_val[i], mean))
        df[atr_imputed[i]] = df[atr_imputed[i]].replace(to_replace = unknown_val[i],
                                                       value = mean)
    return df

""" Imputando valores con la moda. Devuelve el df.
- df: dataframe.

def imput_mode(df):
    atr_imputed = ['WEEKDAY', 'HOUR', 'MAN_COL', 'INT_HWY', 'REL_JCT', 'ALIGN',
                   'PROFILE', 'SUR_COND', 'TRAF_CON', 'SPD_LIM', 'LGHT_CON',
                   'WEATHER', 'PED_ACC', 'ALCOHOL']
    unknown_val = [9, 99, 99, 9, 99, 9, 9, 9, 99, 99, 9, 9, 9999, 9]
    df['PED_ACC'] = df['PED_ACC'].replace(to_replace = 9998, value = 9999)
    for i in range(len(atr_imputed)):
        df[atr_imputed[i]] = df[atr_imputed[i]].replace(to_replace = unknown_val[i],
                                                       value = None)
        mode = int(df[atr_imputed[i]].mode())
        if (IMPRIME_INFO):
            print(" Cambiando en {} el valor {} por su moda: {}"
                  .format(atr_imputed[i], unknown_val[i], mode))
        df[atr_imputed[i]] = df[atr_imputed[i]].replace(unknown_val[i], mode)
    return df
```

3.2. Eliminando instancias

La segunda manera que se propone para abordar este problema es eliminar las instancias en donde hay presentes valores perdidos. La siguiente función que he implementado detecta aquellas instancias que cumplen la condición de que tienen un valor perdido (que para cada atributo este valor cambia) y almacenan en una lista los índices de aquellas instancias que satisfacen la condición. Posteriormente elimino dichos índices.

```
""" Borrando intancias con valores desconocidos. Devuelve el df.
- df: dataframe.

def delete_instances(df):
    # No considero SPEED_LIM que siempre es desconocido.
    atr_imputed = [ 'WEEKDAY', 'HOUR', 'MAN_COL', 'INT_HWY', 'REL_JCT', 'ALIGN',
                    'PROFILE', 'SUR_COND', 'TRAF_CON', 'LGHT_CON',
                    'WEATHER', 'PED_ACC', 'PED_ACC', 'ALCOHOL' ]
    unknown_val = [9, 99, 99, 9, 99, 9, 9, 99, 9, 9, 9998, 9999, 9]
    to_delete = []
    for i in range(len(df)):
        cond = False
        for j in range(len(atr_imputed)):
            if(df.iloc[0][i][atr_imputed[j]]==unknown_val[j]):
                cond = True
        to_delete.append(i)
    if(IMPRIME_INFO):
        print("Numero de instancias a eliminar: {}".format(len(to_delete)))
    df = df.drop(to_delete, axis=0)
    df=df.reset_index()
    del df['index']
    return df
```

3.3. Eliminando atributos

Este modo de tratar los datos imputados es más rápido y eficiente. Sencillamente se borran aquellos atributos que contienen estos valores perdidos. El código es el siguiente:

```
""" Borrando ícaractersticas que contienen valores desconocidos. Devuelve el df.
- df: dataframe.

def delete_attributes(df):
    # No considero SPEED_LIM que siempre es desconocido.
    atr_imputed = [ 'WEEKDAY', 'HOUR', 'MAN_COL', 'INT_HWY', 'REL_JCT', 'ALIGN',
                    'PROFILE', 'SUR_COND', 'TRAF_CON', 'SPD_LIM', 'LGHT_CON',
                    'WEATHER', 'PED_ACC', 'ALCOHOL' ]
    if(IMPRIME_INFO):
        print("Numero de atributos a eliminar: {}".format(len(atr_imputed)))
    df = df.drop(columns = atr_imputed)
    return df
```

Para distinguir entre los distintos modos de imputación ya explicados tengo una constante denominada `INPUT_MODE`. Se actúa de la siguiente manera:

- Para el valor 0 se imputa con la media.
- Para el valor 1 con la moda.
- Para el valor 2 con la eliminación de instancias.
- Para el valor 3 con la eliminación de atributos.

Por defecto, uso el 2 que es el que mejor resultados me ha dado. Se eliminan 50 instancias.

4. Selección de características

El conjunto de datos contiene un gran número de características, algunas de las cuales podrían prescindirse en aras de facilitar la interpretabilidad del conocimiento extraído e incluso mejorar la precisión. En mi caso he trabajado sobre el conjunto de datos con características discretizadas para reducir el esfuerzo computacional. De este modo el preprocesamiento es algo que se va construyendo sobre lo que se ha elegido en el paso anterior.

Para ello he usado un selector que hay en Python basado en RFE (Recursive Feature Elimination). Se indica a este algoritmo el número de atributos a conservar, en mi caso 15. A través del nombre de los atributos seleccionados hago posteriormente el borrado. Muestro algunos datos que arroja la técnica y finalmente devuelvo el conjunto de datos obtenido.

```
""" Selecciona las caracteristicas mas relevantes usando
Recursive Feature Elimination (RFE).
Devuelve el df.
- X: datos de entrada.
- y: etiquetas.
- n_features: numero de caracteristicas.
- feature_cols: nombres de las columnas.
"""
def select_features(X, y, n_features, feature_cols):
    # Selección de características
    selector = RFE(DecisionTreeClassifier(), n_features)
    selector = selector.fit(X, y)
    to_select = selector.support_
    to_delete_attributes = []
    features = []
    for i in range(len(feature_cols)):
        if(not to_select[i]):
            to_delete_attributes.append(feature_cols[i])
        else:
            features.append(feature_cols[i])
    if(IMPRIME_INFO):
        print("\nNum Features: %d" % (selector.n_features_))
        print("Selected Features: %s" % (selector.support_))
        print("Feature Ranking: %s" % (selector.ranking_))
        print(to_delete_attributes)
    return X.drop(columns = to_delete_attributes), features
```

Sobre el conjunto de datos sin imputar las características eliminadas son: VEH_INVL, NON_INVL, INT_HWY, NUM_LAN, SPD_LIM, SCHL_BUS, ALCOHOL, REGION, WRK_ZONE.

Sobre el conjunto de datos imputados las características eliminadas son: NON_INVL, INT_HWY, TRAF_WAY, WEATHR_I, SCHL_BUS, PED_ACC, ALCHL_I, REGION, WRK_ZONE.

5. Selección de instancias

El conjunto de datos contiene un número considerable de instancias. Reducir este número no solo ayuda a mejorar el coste computacional del algoritmo de aprendizaje, sino que también puede ayudar a generar modelos más legibles (por ejemplo, árboles con menos nodos) e incluso más precisos. He trabajado con el conjunto de datos obtenido hasta el momento para reducir el esfuerzo computacional.

5.1. Muestreo

En esta subsección vamos a aplicar técnicas de muestreo para comprobar el efecto sobre el porcentaje de clasificación obtenido por el algoritmo de prueba, en este caso un árbol de decisión.

```
""" Seleccionando instancias con muestreo aleatorio. Devuelve el df.
- df: dataframe.
- frac: fraccion entre 0 y 1 de los datos a coger.
"""
def select_instances(df, frac):
    df = df.sample(frac=frac, random_state=1)
    if IMPRIME_INFO:
        print(" Seleccion de {} instancias aleatorias ".format(df.shape[0]))
    df=df.reset_index()
    del df[ 'index' ]
    return df
```

5.2. Balanceando los datos

El conjunto de datos contiene una categoría de la clase mucho más infrecuente que el resto, lo que hace que los datos no estén balanceados. Esta clase es cuando el tipo de accidente es mortal. Para hacer frente a esto y tener un equilibrio de frecuencia de clase vamos a enfrentar dicha clase contra el resto. Para ello, como el resto de clases contienen más instancias, se va a seleccionar un número de instancias similar al que hay en la clase minoritaria. De este modo se elimina el balanceo.

```
""" Borra una clase para testear el desbalanceo de datos.
- df: dataframe.
- clas: clase a borrar.
"""
def delete_class(df, clas):
    to_delete = []
    for i in range(len(df)):
        if(df.iloc(0)[i][ "CRASH_TYPE"]==clas):
            to_delete.append(i)
    if IMPRIME_INFO:
        print("Número de instancias a eliminar: {}".format(len(to_delete)))
    df = df.drop(to_delete, axis=0)
    df=df.reset_index()
    del df[ 'index' ]
    return df

""" Elige aleatoriamente tantas instancias de la clase distinta a la clase '2' como instancias tiene
- df: dataframe.
"""
def select_n_instances(df):
    #selecciónamos un subconjunto
```

```

num = df.groupby(['CRASH_TYPE']).size()[2] # número de instancias de la clase 2
df_copy = delete_class(df, 2)
df_copy = df_copy.sample(n=num, random_state=1)
if(IMPRIME_INFO):
    print("Sección de {} instancias aleatorias".format(df_copy.shape[0]))
df_copy=df_copy.reset_index()
del df_copy['index']
df_2 = df[df['CRASH_TYPE'] == 2]
df = pd.concat([df_2, df_copy])
df = df.reset_index()
del df['index']
return df

```

El árbol obtenido para los datos sin imputar en este desbalanceo es:

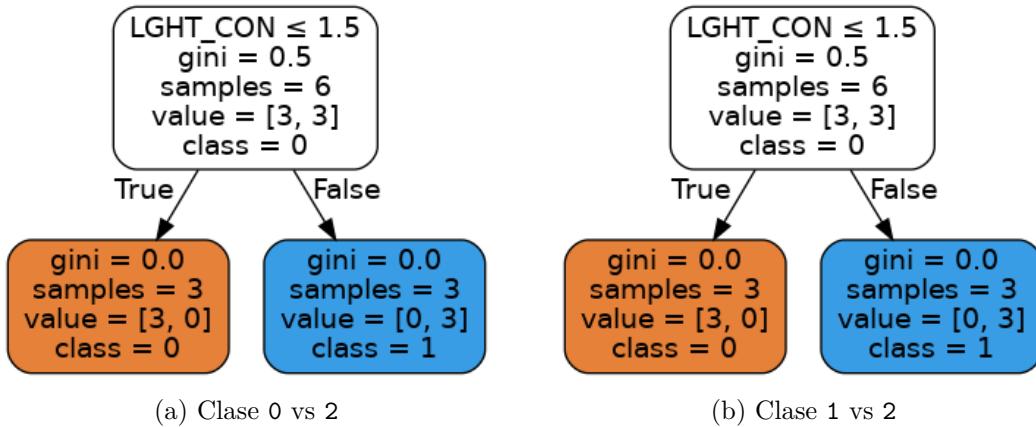


Imagen 1: Desbalanceo en los datos sin imputar

El accuracy en ambos casos es del 50 %. Sobre los datos imputados se ha obtenido un accuracy del 100 %, veamos el árbol:

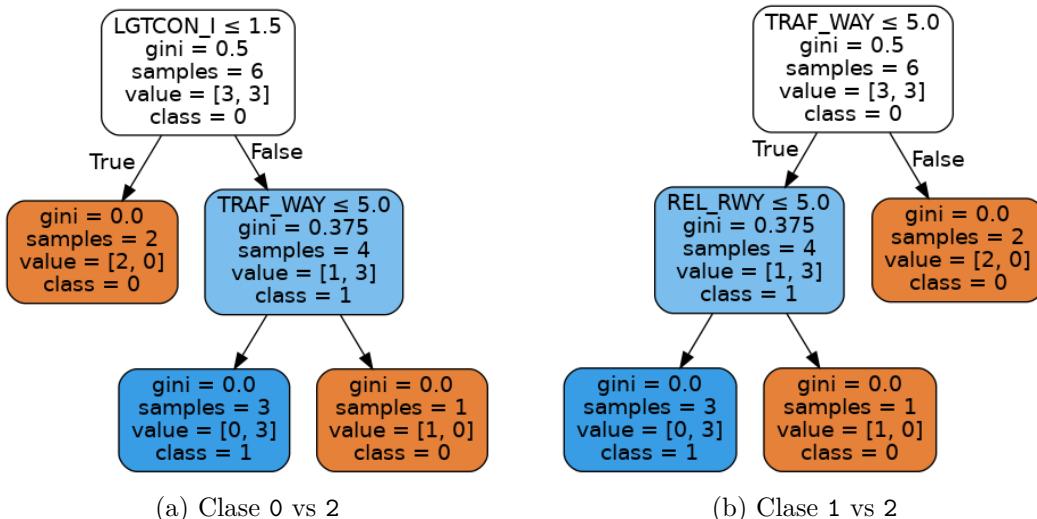


Imagen 2: Desbalanceo en los datos imputados

6. Resultados

En esta sección se van a exponer los resultados obtenidos a lo largo de este proceso de preprocesamiento en donde la clasificación se va a realizar mediante un árbol de decisión.

6.1. Datos sin imputar

Comenzamos exponiendo los resultados del conjunto de datos sin imputar. Después del procesamiento se ha dividido en 80% para entrenamiento y 20% para test. El dataset tiene ahora las siguientes características:

```
----- Información del conjunto de datos -----
Tamaño de X_train: (587, 15)
Tamaño de X_test: (147, 15)
Tamaño de y_train: (587,)
Tamaño de y_test: (147,)
```

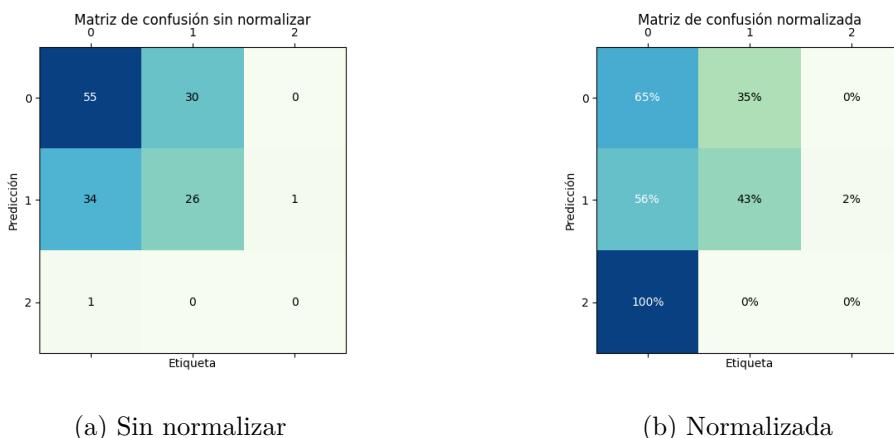
Imagen 3: Información de los datos sin imputar

Los resultados obtenidos son los siguientes:

```
----- Ejecutando el modelo sobre los datos sin imputar -----
Construyendo el árbol de decisión
Entrenando el árbol de decisión
Prediciendo etiquetas
Accuracy: 0.5306122448979592
Matriz de confusión:
[[49 36 0]
 [32 29 0]
 [ 0 1 0]]
Pintando el árbol
```

Imagen 4: Resultados de los datos sin imputar

Se ha simplificado mucho para que el árbol obtenido sea menos complejo y esto se traduce en un accuracy del 53,06 %. La matriz de confusión, y su normalizada, son:



(a) Sin normalizar

(b) Normalizada

Imagen 5: Matriz de confusión

Por último el árbol de decisión obtenido:

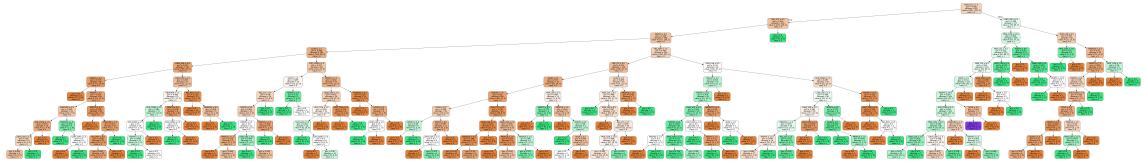


Imagen 6: Árbol de los datos sin imputar

6.2. Datos imputados

Continuamos exponiendo los resultados del conjunto de datos imputado. Después del procesamiento se ha dividido en 80 % para entrenamiento y 20 % para test. El dataset tiene ahora las siguientes características:

```
----- Información del conjunto de datos imputados -----
Tamaño de X_train: (616, 15)
Tamaño de X_test: (155, 15)
Tamaño de y_train: (616,)
Tamaño de y_test: (155,)
```

Imagen 7: Información de los datos imputados

Los resultados obtenidos para el conjunto imputado son los siguientes:

```
----- Ejecutando el modelo sobre los datos imputados -----
Construyendo el árbol de decisión
Entrenando el árbol de decisión
Prediciendo etiquetas
Accuracy: 0.47096774193548385
Matriz de confusión:
[[56 42 0]
 [39 17 0]
 [ 0  1 0]]
Pintando el árbol
```

Imagen 8: Resultados de los datos imputados

Se ha simplificado mucho para que el árbol obtenido sea menos complejo y esto se traduce en un accuracy del 47,09 %. Esto indica que la manera de imputar usada mejora a la que venía por defecto ya que el accuracy obtenido por el algoritmo propio es superior.

La matriz de confusión, y su normalizada, son:

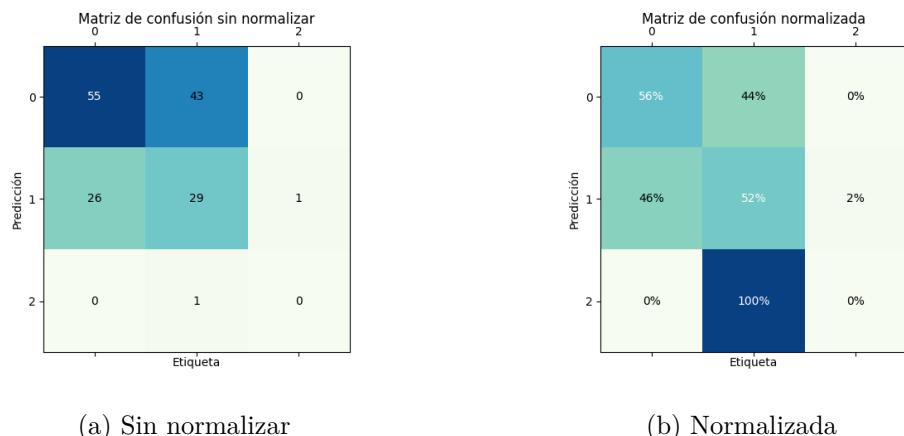


Imagen 9: Matriz de confusión

Por último el árbol de decisión obtenido:

