Clasificación Estelar

Desde siempre hemos mirado al cielo pensando en ¿quiénes somos? ¿por qué estamos aquí? ¿Qué son las estrellas?, en base a estas preguntas y otras más profundas y específicas, nace el estudio de los astros: la astronomía.

En la Astronomía, la clasificación estelar se basa en características espectrales observables. Algunas de estas características son: filtro ultravioleta, filtro infrarrojo, corrimiento al rojo, etc.

¿Pero es posible en base a estas características identificar estos objetos?

Con esta pregunta en nuestras cabezas, elegimos un data frame que es un conjunto de 100.000 observaciones del espacio, obtenidas de la SDSS (Sloan Digital Sky Survey), el cúal es proyecto de investigación del espacio mediante imágenes en el espectro visible y de corrimiento rojo, realizadas en el telescopio Apache Point de Nuevo México.

Data frame: Clasificación Estelar

Cada observación se describe en 17 columnas de características y 1 columna de clase que identifica a cada observación como estrella, galaxia o cuásar (galaxia recién nacida o bien la energía del agujero negro del centro de la galaxia recien nacida).

Ahora bien, después de haber planteado nuestras preguntas y seleccionado una base de datos acorde al tema, surge una nueva y mas que importante pregunta, ¿Sirve nuestro data frame para realizar la clasificación deseada?

Análisis General de los Datos

Comencemos describiendo cada columna del data frame:

N°	Nombre de Columna	Tipo de Variable	Descripción
1	obj_ID	Numérica Discreta	Identificador único de cada objeto
2	alpha	Numérica Continua	Ángulo de ascensión derecha
3	delta	Numérica Continua	Ángulo de declinación
4	u	Numérica Continua	Filtro ultravioleta
5	g	Numérica Continua	Filtro verde
6	r	Numérica Continua	Filtro rojo
7	i	Numérica Continua	Filtro infrarrojo cercano en el sistema fotométrico
8	z	Numérica Continua	Filtro infrarrojo en el sistema fotométrico
9	run_ID	Numérica Discreta	Número de ejecución utilizado para identificar el análisis específico
10	rereun_ID	Numérica Discreta	Número de repetición para especificar cómo se procesó la imagen
11	cam_col	Numérica Discreta	Columna de cámara para identificar la línea de exploración dentro de la ejecución
12	field_ID	Numérica Discreta	Número de campo para identificar cada campo
13	spec_obj_ID	Numérica Discreta	Identificador único utilizado para objetos espectroscópicos ópticos (esto significa que 2 observaciones diferentes con el mismo spec_obj_ID deben compartir la clase de salida)
14	class	Categórica Nominal	Clase del objeto (galaxia, estrella o cuásar)
15	redshift	Numérica Continua	Valor de corrimiento al rojo basado en el aumento de la longitud de onda
16	plate	Numérica Discreta	Identificador de placa del SDSS
17	MJD	Numérica Discreta	Fecha utilizada para indicar cuándo se tomó un determinado dato del SDSS
18	fiber_ID	Numérica Discreta	Identificador de fibra que apuntó la luz al plano focal en cada observación

Importamos las librerías con las que vamos a trabajar:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

Importamos nuestro data frame en la variable declarada como "df"

```
df =
pd.read_csv('https://raw.githubusercontent.com/MKSuser/Unsam1/refs/heads/main/star_classification.csv')
```

Printeamos los nombres de las columnas como vienen por default:

```
print(df.columns)
```

Renombramos los nombres de las variables, para hacerlas mas entendibles (información extraída del sitio):

```
df.rename(columns={'obj_ID': 'Id del objeto',
                   'alpha': 'Ang Asc Derecha',
                  'delta': 'Ang Declinación',
                   'u': 'Filtro ultravioleta',
                   'g':'Filtro verde',
                   'r': 'Filtro Rojo',
                   'i': 'Filtro Inf. cercano',
                   'z': 'Filtro Inf.',
                   'run_ID': 'Id de Analisis',
                   'rerun_ID': 'Id de repetición',
                   'cam_col': 'Columna de cámara',
                   'field_ID': 'Id de campo',
                   'spec_obj_ID': 'Id',
                   'class': 'Clase',
                   'redshift': 'Corrimiento al Rojo',
                   'plate': 'Id de placa SDSS',
                   'MJD': 'Fecha del dato',
                   'fiber_ID': 'Id de fibra',

                   },inplace=True)
```

Identificamos que tipo de dato es cada columna:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 18 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   Id del objeto         100000 non-null  float64
 1   Ang Asc Derecha       100000 non-null  float64
 2   Ang Declinación       100000 non-null  float64
 3   Filtro ultravioleta   100000 non-null  float64
 4   Filtro verde          100000 non-null  float64
 5   Filtro Rojo           100000 non-null  float64
 6   Filtro Inf. cercano   100000 non-null  float64
 7   Filtro Inf.           100000 non-null  float64
 8   Id de Analisis        100000 non-null  int64  
 9   Id de repetición      100000 non-null  int64  
 10  Columna de cámara     100000 non-null  int64  
 11  Id de campo           100000 non-null  int64  
 12  Id                    100000 non-null  float64
 13  Clase                 100000 non-null  object 
 14  Corrimiento al Rojo   100000 non-null  float64
 15  Id de placa SDSS      100000 non-null  int64  
 16  Fecha del dato        100000 non-null  int64  
 17  Id de fibra           100000 non-null  int64  
dtypes: float64(10), int64(7), object(1)
memory usage: 13.7+ MB
```

Como podemos observar, la columna 13 "Clase" es un objeto, así que posiblemente sea nuestra "salida".

Para entender mejor la composición de los datos, imprimimos 5 muestras aleatorias:

```
df.sample(5)
```

	Id del objeto	Ang Asc Derecha	Ang Declinación	Filtro ultravioleta	Filtro verde	Filtro Rojo	Filtro Inf. cercano	Filtro Inf.	Id de Analisis	Id de repetición	Columna de cámara	Id de campo	Id	Clase	Corrimiento al Rojo	Id de placa SDSS	Fecha del dato	Id de fibra
80661	1.237656e+18	256.805559	34.379167	24.05063	21.81778	21.56482	21.71070	21.50854	2335	301	4	82	5.621824e+18	STAR	-0.000721	4993	55738	750
29318	1.237661e+18	113.898527	19.019870	23.74801	22.98985	20.90062	19.97892	19.61908	3605	301	5	25	5.053184e+18	GALAXY	0.501353	4488	55571	527
48337	1.237679e+18	28.734305	-2.239941	22.56552	22.23733	20.59675	19.67936	19.22537	7780	301	4	94	4.895686e+18	GALAXY	0.559802	4348	55559	993
58943	1.237679e+18	333.929737	4.466526	23.65763	21.00473	19.18282	18.48564	18.05436	7765	301	4	148	4.861893e+18	GALAXY	0.315895	4318	55508	934
92155	1.237680e+18	332.131595	19.577808	20.75017	20.66658	20.25803	20.20986	20.07291	8102	301	4	54	8.527732e+18	QSO	1.353902	7574	56945	604

Definitivamente la columna "Clase" es nuestra salida para poder hacer nuestra clasificación estelar.

Antes de continuar, observemos en detalle nuestra columna "clase", contemos cuantas salidas posibles tiene y en que cantidades:

```
df["Clase"].value_counts()
```

	count
Clase	
GALAXY	59445
STAR	21594
QSO	18961

dtype: int64

Realizamos lo mismo, pero esta vez limpiando los posibles NaN:

```
df = df.dropna()

df["Clase"].value_counts()
```

	count
Clase	
GALAXY	59445
STAR	21594
QSO	18961

dtype: int64

Como podemos observar, no hay ningún NaN.

Ahora bien, ya pudimos determinar que las 3 salidas posibles son "GALAXY" (galaxia), "STAR" (estrella) y "QSO" (cuásar) y cuántas hay de cada una.

Sabiendo esto, convertiremos cada clase en un número (creamos un diccionario), esto nos ayudará a buscar correlatividades:

```
class_rep = {
      "GALAXY": 0,
      "STAR":   1,
      "QSO":    2,
}
```

Para hacer uso de una buena práctica, creamos una nueva variable "df_limpio" y guardamos el data frame dentro de ella, para poder modificaciones para un mejor análisis de datos. Luego reemplazamos los datos de la columna "clase" por los creados en el paso anterior y movemos la columna "clase" al final para un mejor ordenamiento:

```
df_limpio = df

df_limpio['Clase'] = df_limpio['Clase'].replace(class_rep)

df_limpio = df_limpio[[col for col in df.columns if col != 'Clase'] +
['Clase']]
```

```
<ipython-input-10-6ee8f88730d3>:3: FutureWarning: Downcasting behavior
in `replace` is deprecated and will be removed in a future version. To
retain the old behavior, explicitly call
`result.infer_objects(copy=False)`. To opt-in to the future behavior,
set `pd.set_option('future.no_silent_downcasting', True)`
  df_limpio['Clase'] = df_limpio['Clase'].replace(class_rep)
```

Chequeamos que la columna "clase" se haya modificado exitosamente, imprimiendo 5 filas aleatorias:

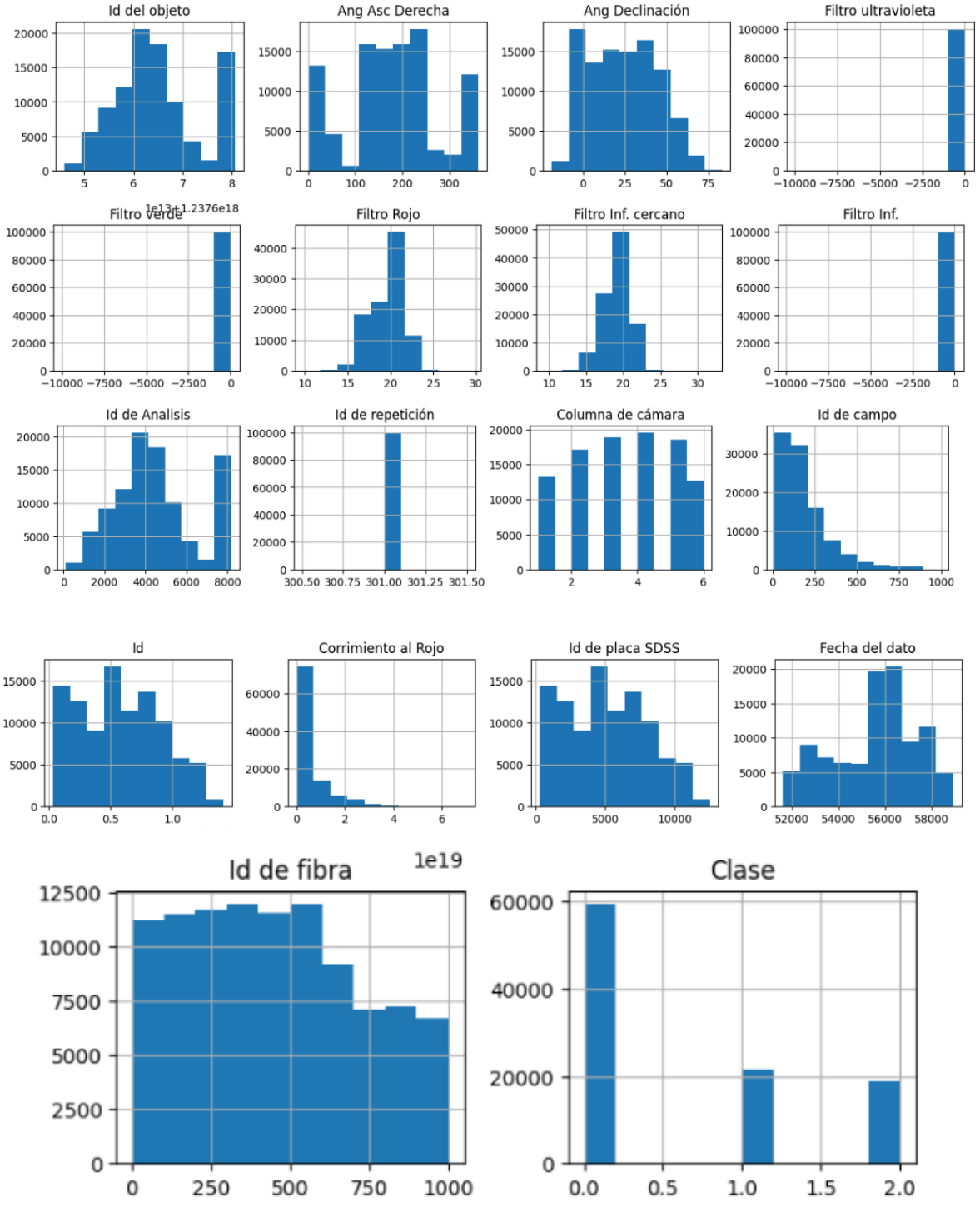
	Id del objeto	Ang Asc Derecha	Ang Declinación	Filtro ultravioleta	Filtro verde	Filtro Rojo	Filtro Inf. cercano	Filtro Inf.	Id de Analisis	Id de repetición	Columna de cámara	Id de campo	Id	Corrimiento al Rojo	Id de placa SDSS	Fecha del dato	Id de fibra	Clase
74465	1.237662e+18	234.788563	35.714106	24.51550	23.37323	21.23780	19.93472	19.49065	3926	301	5	56	5.600302e+18	0.647532	4974	56038	275	0
33620	1.237662e+18	170.837041	42.609084	21.02949	20.19381	20.11296	20.15406	20.04498	3840	301	6	106	5.276009e+18	-0.000440	4686	56013	152	1
14971	1.237665e+18	193.149631	33.999307	23.12813	21.70500	20.44870	19.25274	18.44051	4576	301	2	493	4.474431e+18	0.744436	3974	55320	382	0
88575	1.237661e+18	214.480194	46.565680	25.01578	20.43502	18.65641	18.05597	17.69474	3664	301	5	36	1.447917e+18	0.327951	1286	52725	35	0
23115	1.237664e+18	118.740651	53.484142	20.75950	18.86445	17.63461	17.13206	16.75931	4264	301	5	120	2.105541e+18	0.173043	1870	53383	393	0

Ahora vamos a analisar la distribución de datos, para eso imprimimos los histogramas de cada columna:

```
df_limpio.hist(figsize=(15, 15))
```

```
array([[<Axes: title={'center': 'Id del objeto'}>,
        <Axes: title={'center': 'Ang Asc Derecha'}>,
        <Axes: title={'center': 'Ang Declinación'}>,
        <Axes: title={'center': 'Filtro ultravioleta'}>],
       [<Axes: title={'center': 'Filtro verde'}>,
        <Axes: title={'center': 'Filtro Rojo'}>,
        <Axes: title={'center': 'Filtro Inf. cercano'}>,
        <Axes: title={'center': 'Filtro Inf.'}>],
       [<Axes: title={'center': 'Id de Analisis'}>,
        <Axes: title={'center': 'Id de repetición'}>,
        <Axes: title={'center': 'Columna de cámara'}>,
        <Axes: title={'center': 'Id de campo'}>],
       [<Axes: title={'center': 'Id'}>,
        <Axes: title={'center': 'Corrimiento al Rojo'}>,
        <Axes: title={'center': 'Id de placa SDSS'}>,
```

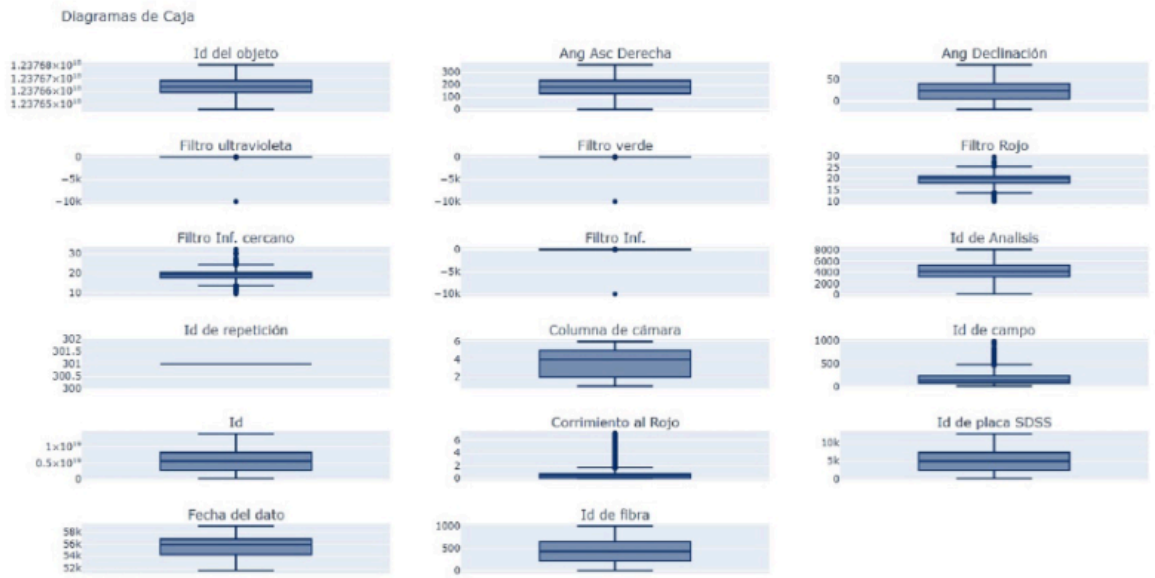
```
  <Axes: title={'center': 'Fecha del dato'}>],
 [<Axes: title={'center': 'Id de fibra'}>,
  <Axes: title={'center': 'Clase'}>, <Axes: >, <Axes: >]],
dtype=object)
```



Pasamos a verificar la existencia de datos atípicos en nuestro dataframe utilizando "ploty".

Ploty permite hacer una imagen con pequeños gráficos con la herramienta "subplots". Así logramos hacer visibles los valores que se encuentran por fuera de los intercuartiles -las cajas- teniendo en cuenta el "* 1.5". La idea es poder verificar la distribución de datos de otra manera distinta a la de los histogramas.

```
import plotly.express as px
import plotly.subplots as sp

# Número de columnas (menos el output de clases)
num_caracteristicas = df_limpio.shape[1] - 1

# Número de columnas en el graph
num_columnas = 3

# Número de filas necesarias para acomodar todas las columnas
num_filas = (num_caracteristicas // num_columnas) + 1

# Creamos la imagen con los graficos peque
fig = sp.make_subplots(rows=num_filas, cols=num_columnas,
subplot_titles=df_limpio.columns[:-1].tolist())

# Creamos diagramas de caja para cada columna
for i, columna in enumerate(df_limpio.columns[:-1]):
    fila = i // num_columnas + 1  # Calculamos el índice de fila
    columna_idx = i % num_columnas + 1  # Calculamos el índice de
columna
    boxplot = px.box(df_limpio, y=columna, title=f'Diagrama de caja de
{columna}', color_discrete_sequence=px.colors.sequential.Blues_r)

    # Añadimos el trazo del diagrama de caja al gráfico peque
    for trazo in boxplot.data:
        fig.add_trace(trazo, row=fila, col=columna_idx)

# Actualizamos el diseño de la imagen
fig.update_layout(title_text='Diagramas de Caja', height=800)

# Printeo
fig.show()
```



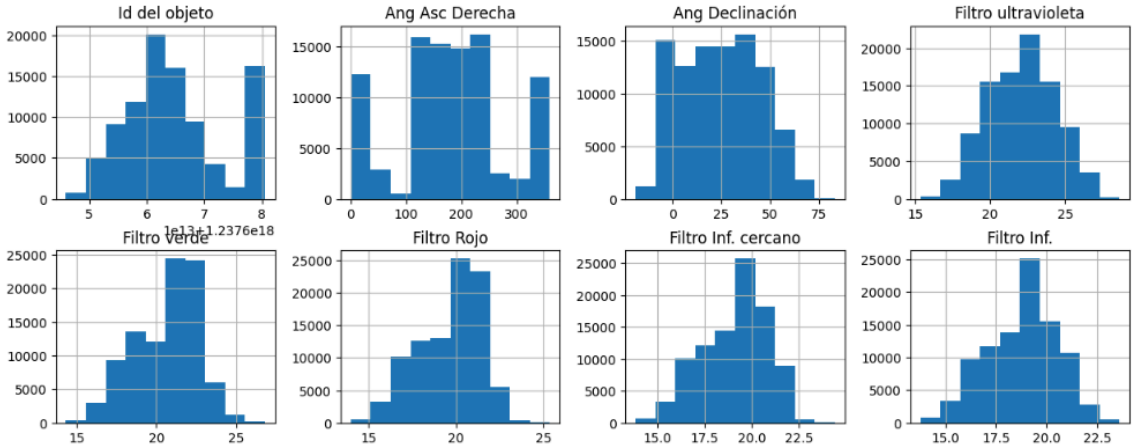
Como observamos una varianza de datos, optamos por trabajarlos por cuartiles, para alejar los valores atípicos.

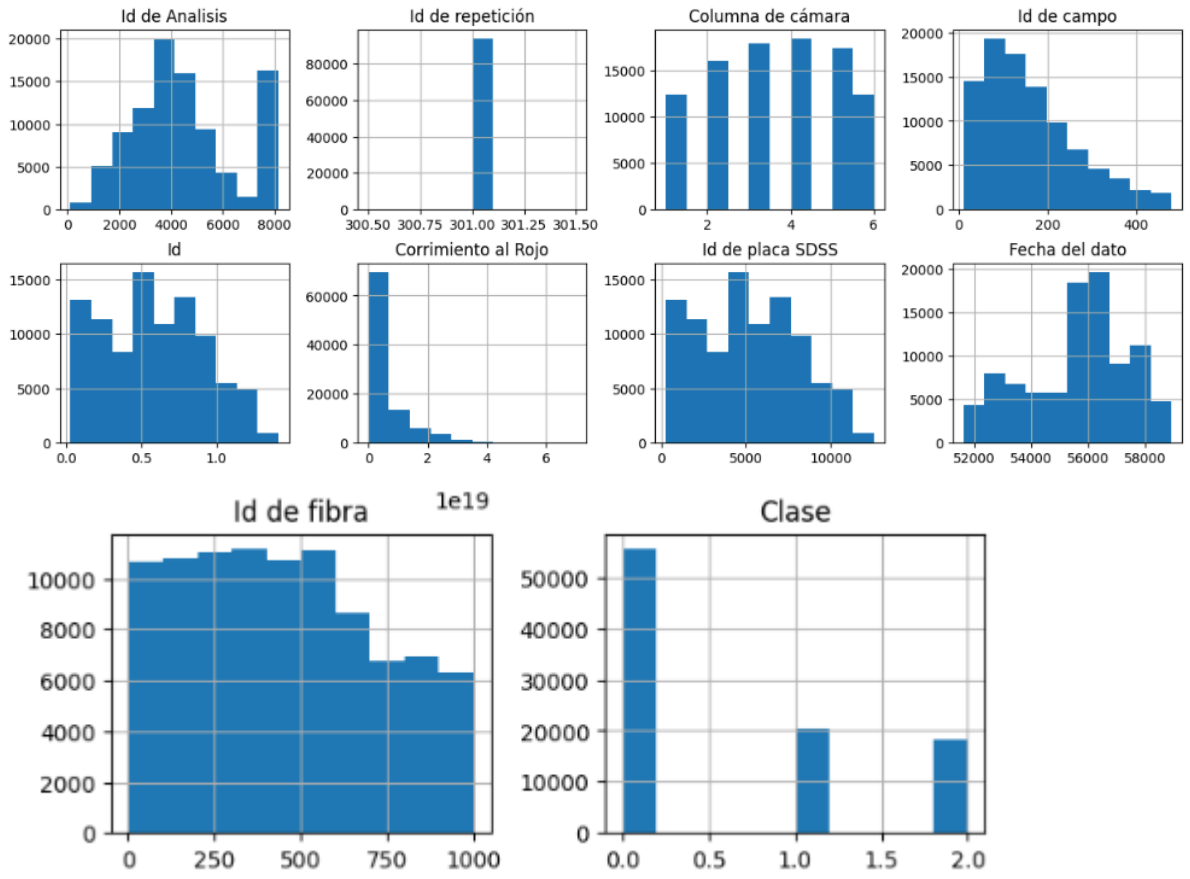
Para esto, primero sacamos las columnas "clase" y "Corrimiento al Rojo", luego separamos los cuartiles iniciales y finales, tomamos la diferencia (IQR) y aplicamos la ecuación del 1.5. Sacamos del "df_limpio" las filas con valores atípicos, volvemos a colocar las columnas que sacamos, limpiamos los NaN posibles e imprimimos los nuevos histogramas:

```
# Primero sacamos las columnas detalladas
df_limpio_salida = df_limpio['Clase']
df_limpio_red = df_limpio['Corrimiento al Rojo']

# Separamos los cuartiles teniendo encuenta antes del 25%, y luego del
75%
Q1 = df_limpio.quantile(0.25)
Q3 = df_limpio.quantile(0.75)

# Tomamos la diferencia para poder laburar
IQR = Q3 - Q1

# Teniendo en cuenta el IQR aplicamos la ecuación del 1.5
atipicos = ((df_limpio < (Q1 - 1.5 * IQR)) | (df_limpio > (Q3 + 1.5 *
IQR)))

# Sacamos del df_limpio las filas con valores atípicos
df_limpio = df_limpio.where(np.invert(atipicos))
```

```
# Volvemos a colocar las columnas que sacamos para no generar NaNs que
desequilibran la base
df_limpio['Clase'] = df_limpio_salida
df_limpio['Corrimiento al Rojo'] = df_limpio_red

# Limpiamos los NaN que se puedan haber generado
df_limpio = df_limpio.dropna()

# Nuevos histogramas
df_limpio.hist(figsize=(15, 15))
```

```
array([[<Axes: title={'center': 'Id del objeto'}>,
        <Axes: title={'center': 'Ang Asc Derecha'}>,
        <Axes: title={'center': 'Ang Declinación'}>,
        <Axes: title={'center': 'Filtro ultravioleta'}>],
       [<Axes: title={'center': 'Filtro verde'}>,
        <Axes: title={'center': 'Filtro Rojo'}>,
        <Axes: title={'center': 'Filtro Inf. cercano'}>,
        <Axes: title={'center': 'Filtro Inf.'}>],
       [<Axes: title={'center': 'Id de Analisis'}>,
        <Axes: title={'center': 'Id de repetición'}>,
        <Axes: title={'center': 'Columna de cámara'}>,
        <Axes: title={'center': 'Id de campo'}>],
       [<Axes: title={'center': 'Id'}>,
        <Axes: title={'center': 'Corrimiento al Rojo'}>,
        <Axes: title={'center': 'Id de placa SDSS'}>,
        <Axes: title={'center': 'Fecha del dato'}>],
       [<Axes: title={'center': 'Id de fibra'}>,
        <Axes: title={'center': 'Clase'}>, <Axes: >, <Axes: >]],
      dtype=object)
```





Como podemos observar en los histogramas, contemplamos como la distribución de datos se hizo mas armónica que antes de quitar los cuartiles, entendiendo que los valores atípicos generan un desvalance en la distribución general de los datos.

Ahora realizaremos una estadística rápida de los datos normalizados, excluyendo la columna "clase":

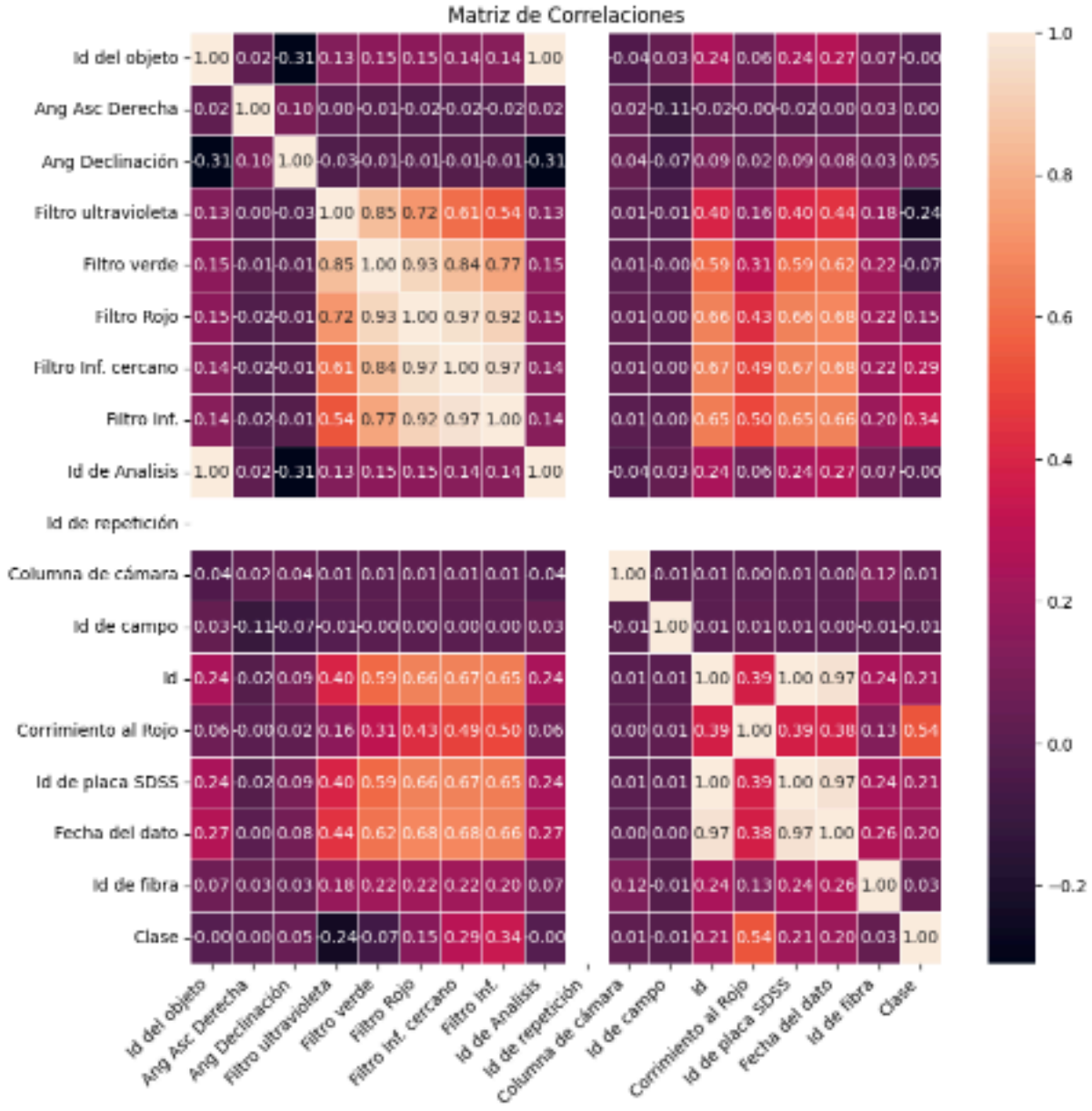
```
df_limpio_salida = df_limpio['Clase']

df_limpio_stats = df_limpio.describe().T
df_limpio_n = (df_limpio - df_limpio_stats['mean']) /
df_limpio_stats['std']  # Para normalizar: (valor - promedio) /
desv_estandar

df_limpio_n['Clase'] = df_limpio_salida

df_limpio_n.describe()
```

	Id del objeto	Ang Asc Derecha	Ang Declinación	Filtro ultravioleta	Filtro verde	Filtro Rojo	Filtro Inf. cercano	Filtro Inf.	Id de Analisis	Id de repetición	Columna de cámara	Id de campo	Id	Corrimiento al Rojo	Id de placa SDSS	Fecha del dato	Id de Fibra	Clase
ount	9.423000e+04	9.423000e+04	9.423000e+04	9.423000e+04	9.423000e+04	9.423000e+04	9.423000e+04	9.423000e+04	9.423000e+04	0.0	9.423000e+04	9.423000e+04	9.423000e+04	9.423000e+04	9.423000e+04	9.423000e+04	9.423000e+04	94230.000000
mean	1.537825e-11	2.967947e-16	3.559123e-17	-1.421953e-15	-9.240147e-16	1.056426e-16	-1.542563e-15	1.503579e-15	8.445377e-18	NaN	-9.772508e-17	-1.375390e-16	2.997355e-16	2.412965e-18	6.756302e-17	-5.741348e-16	-1.568427e-17	0.602706
ctd	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	NaN	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	0.790908
min	-2.220194e+00	-1.874779e+00	-2.225193e+00	-3.017696e+00	-3.146584e+00	-3.104853e+00	-3.030083e+00	-2.885170e+00	-2.220215e+00	NaN	-1.587325e+00	-1.431396e+00	-1.684375e+00	-8.078466e-01	-1.684356e+00	-2.252299e+00	-1.641947e+00	0.000000
25%	-6.585988e-01	-5.243745e-01	-9.182433e-01	-7.743911e-01	-8.246431e-01	-8.198730e-01	-7.754395e-01	-7.462537e-01	-6.586893e-01	NaN	-9.581692e-01	-7.823020e-01	-8.926849e-01	-7.167002e-01	-8.926750e-01	-6.781511e-01	-8.412472e-01	0.000000
50%	-1.614992e-01	5.843449e-03	-7.742845e-04	4.135116e-02	2.315774e-01	2.629905e-01	1.845515e-01	1.340043e-01	-1.614010e-01	NaN	3.001427e-01	-2.095718e-01	-5.672465e-02	-2.027057e-01	-5.674718e-02	1.611705e-01	-6.248453e-02	0.000000
75%	4.508342e-01	5.807933e-01	8.074450e-01	7.147557e-01	7.329122e-01	7.565672e-01	7.554135e-01	6.567189e-01	4.508026e-01	NaN	9.292986e-01	5.540685e-01	7.485283e-01	1.757785e-01	7.485481e-01	6.551527e-01	7.272466e-01	1.000000
max	1.874697e+00	1.867804e+00	2.959903e+00	2.918567e+00	3.059649e+00	3.117907e+00	3.047688e+00	2.763571e+00	1.874532e+00	NaN	1.558455e+00	3.035899e+00	2.495655e+00	8.747721e+00	2.495623e+00	1.840374e+00	2.010560e+00	2.000000

Estas transformaciones nos ayudan a estandarizar los datos para que sean comparables en escala y asegurar que el data frame esté listo para usarse.

Todo muy lindo, pero mejor grafiquemos las correlaciones:

```
fig, ax = plt.subplots(figsize=(10, 10))
sns.heatmap(df_limpio.corr(), annot=True, linewidths=0.5, fmt=
'.2f',ax=ax)
ax.set_xticklabels(df_limpio.columns.to_list(), rotation=45,
ha='right')
plt.title("Matriz de Correlaciones")
plt.show()
```



Si bien determinamos los coeficientes de correlación entre las variables, ¿para qué nos sirve esta info?

El coeficiente de correlación, también llamado correlación de Pearson, mide la fuerza de la relación entre dos variables como un valor entre –1 y 1.

*Un coeficiente de correlación más cercano a 0 indica que no hay correlación.

*Un coeficiente de correlación más cercano a 1 indica una fuerte correlación positiva, lo que significa que cuando una variable aumenta, la otra aumenta proporcionalmente.

*Si está más cerca de –1, entonces indica una fuerte correlación negativa, lo que significa que a medida que una variable aumenta, la otra disminuye proporcionalmente.

Ahora con una mejor visión de los datos, podemos observar que las siguientes columnas están correlacionadas con nuestra columna de salida "clase":

-Filtro Ultravioleta

-Filtro Rojo

-Filtro Infrarrojo Cercano

-Filtro Infrarrojo

-Id

-Corrimiento al Rojo

-Id de placa SDSS

-Fecha del dato

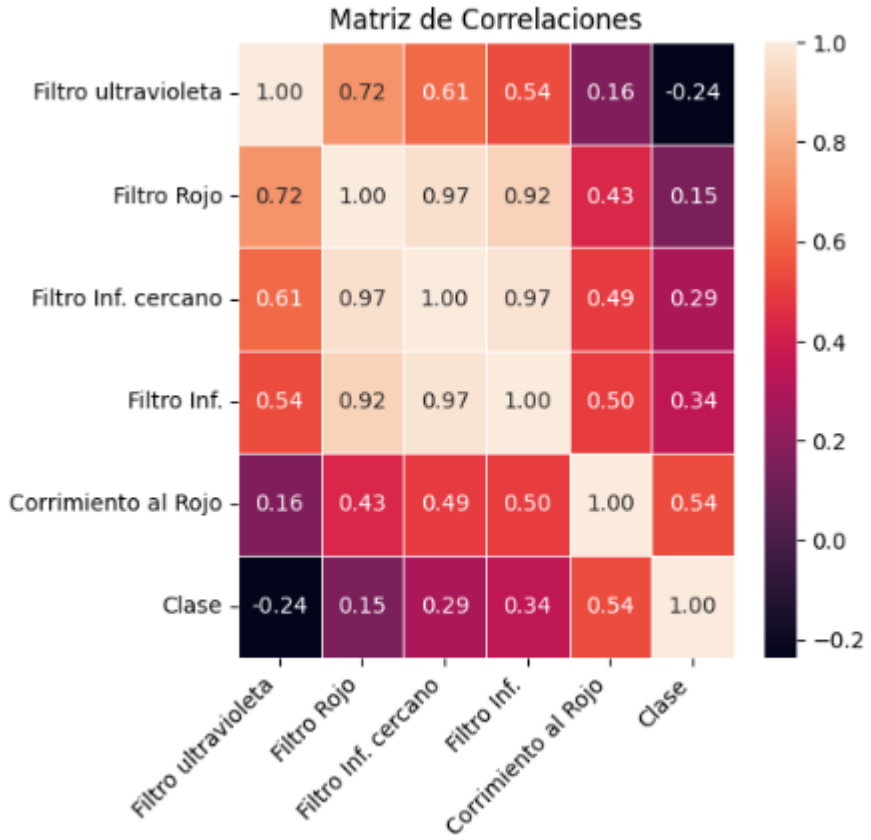
Al ver las columnas que se correlacionan con nuestra salida, podemos descartar "Id", "Id de placa SDSS" y "Fecha del dato", dado que no son datos observables en el espacio, sino mas bien, identificaciones posteriores.

Entonces manos a la obra, limpiemos nuestro data frame de los datos que no son correlativos:

```
df_limpio = df_limpio.drop ('Id del objeto', axis = 1)
df_limpio = df_limpio.drop ('Ang Asc Derecha', axis = 1)
df_limpio = df_limpio.drop ('Ang Declinación', axis = 1)
# df_limpio = df_limpio.drop ('Filtro ultravioleta', axis = 1)
df_limpio = df_limpio.drop ('Filtro verde', axis = 1)
# df_limpio = df_limpio.drop ('Filtro Rojo', axis = 1)
# df_limpio = df_limpio.drop ('Filtro Inf. cercano', axis = 1)
# df_limpio = df_limpio.drop ('Filtro Inf.', axis = 1)
df_limpio = df_limpio.drop ('Id de Analisis', axis = 1)
df_limpio = df_limpio.drop ('Id de repetición', axis = 1)
df_limpio = df_limpio.drop ('Columna de cámara', axis = 1)
df_limpio = df_limpio.drop ('Id de campo', axis = 1)
df_limpio = df_limpio.drop ('Id', axis = 1)
# df_limpio = df_limpio.drop ('Corrimiento al Rojo', axis = 1)
df_limpio = df_limpio.drop ('Id de placa SDSS', axis = 1)
df_limpio = df_limpio.drop ('Fecha del dato', axis = 1)
df_limpio = df_limpio.drop ('Id de fibra', axis = 1)
```

Volvamos a imprimir esa matriz de correlaciones y veamos que limpia y bonita quedó:

```
fig, ax = plt.subplots(figsize=(5, 5))
sns.heatmap(df_limpio.corr(), annot=True, linewidths=0.5, fmt=
'.2f',ax=ax)
ax.set_xticklabels(df_limpio.columns.to_list(), rotation=45,
ha='right')
plt.title("Matriz de Correlaciones")
plt.show()

# Código para descargar el data frame limpio
# from google.colab import files
# df_limpio.to_csv('df_limpio.csv', index=False)
# files.download("df_limpio.csv")
```



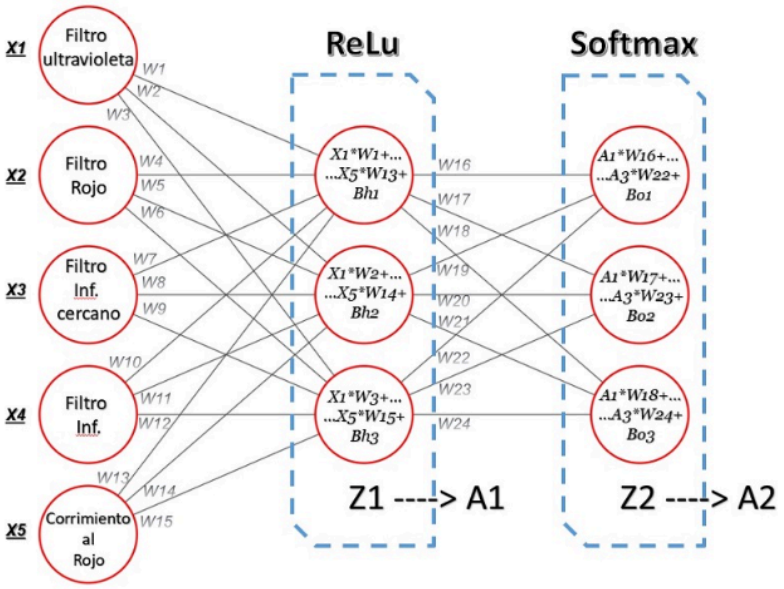
Después de este análisis tan exhaustivo, en vista de esta última matriz de correlaciones y usando estas últimas variables (Filtro Ultravioleta, Filtro Rojo, Filtro Infrarrojo Cercano, Filtro Infrarrojo y Corrimiento al Rojo) podemos concluir que nuestra base de datos es adecuada para entrenar una red neuronal, para determinar si un objeto observado en el espacio es una estrella, galaxia o cuásar.

RED NEURONAL

Nuestra Red Neuronal cuenta con 2 capas, una oculta y una de salida, cada capa cuenta con 3 neuronas.

Para la capa oculta elegimos la función de activación ReLu, dado que no tiene problemas de saturación y es más fácil de implementar, además de ser la más usada (según Google) para hacer redes neuronales.

Para la capa de salida, primero (por inexperiencia) intentamos usar la función Logistic, pero al tener varias salidas, investigamos y (gracias también al profe que nos terminó de explicar y confirmar) usamos la función Softmax, la cual recibe varias entradas, las cuales las trata como probabilidades, hace sus matemáticas y obtiene una probabilidad por cada entrada (entre todas suman 1) y la que sea más cercana a 1 es la seleccionada como salida. Gracias a esta función, podemos obtener los resultados esperados.



```
# Con estocastico

# Llamado de las librerias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tqdm.auto import tqdm

# Declaramos el dataFrame como all_data
df_limpio =
pd.read_csv('https://raw.githubusercontent.com/casquifer/Mate-III/refs/
heads/main/df_limpio.csv')

# Separamos los datos a tener en cuenta, por un lado los datos de
entrada, por el otro los de salida ("class")
all_inputs = df_limpio.iloc[:, 0:5].values # Entradas: columnas 0 a 7
all_outputs = df_limpio.iloc[:, -1].values # Salidas: última columna

# Declaramos la funcion a utilizar para normalizar los datos
def normalizador(X):
    # Calculamos la media y la desviación estándar
    promedio = np.mean(X, axis=0)  # Promedio de cada columna
    desvEst = np.std(X, axis=0)    # Desviación estándar de c/c

    datoNormalizado = (X - promedio) / desvEst
    return datoNormalizado

all_inputs = normalizador(all_inputs)

### Convertir las salidas a formato one-hot

# Declaramos la cantidad de respuestas posibles
# Tener en cuenta que las salidas son 3
respPosibles = 3

# Con nume.eye(respPosibles) generamos una matriz identidad 3x3 que va
a reemplazar a los valores (0,1,2) originales
# Al combinarla con all_outputs vinculamos cada fila de la MId con cada
valor
# La primer fila sería 0, la segunda 1, y la tercera 2
# Hacemos esto para poder trabajar las 3 posibilidades dentro de la
red.
```

```
y_matriz = np.eye(respPosibles)[all_outputs]

# Matriz --- Clase a la que apunta la fila

# 1, 0, 0 --- 0
# 0, 1, 0 --- 1
# 0, 0 ,1 --- 2

# Dividir en un conjunto de entrenamiento y uno de prueba
X_train, X_test, Y_train, Y_test = train_test_split(all_inputs,
y_matriz, test_size=1/3)

n = X_train.shape[0]  # número de registros de entrenamiento

print('Número de registros de entrenamiento: \n')
print(n, '\n')

### Funciones de activación

# Función ReLu
relu = lambda x: np.maximum(x, 0)


#******************Dado que vamos a tener 3 resultados posibles,
decidimos usar la función Softmax que aplica mejor a estos
casos**********************************

#Softmax: recibe una matriz de "x" valores y los convierte en
probabilidades, donde cada valor está entre 0 y 1.
# La suma de todas estas probabilidades es igual a 1, por eso la que
mas se acerca a 1 es a opción elegida como salida

#Expresión matemática de la Softmax: "e" elevado a la "Zi" / sumatoria
K j=1 "e" elevado a la "Zj"

#np.exp(x): calcula la función exponencial de cada elemento en x.
Devueve "e" a la "x"
#np.max(x, axis=0, keepdims=True): encuentra el valor máximo en x a lo
largo del eje especificado (en este caso es 0) y "keepdims" mantienen
las dimensiones originaes
#np.sum(exp_x, axis=0, keepdims=True): calcula la suma de los valores
en "exp" en el eje 0 y mantiene las dimensiones originales
```

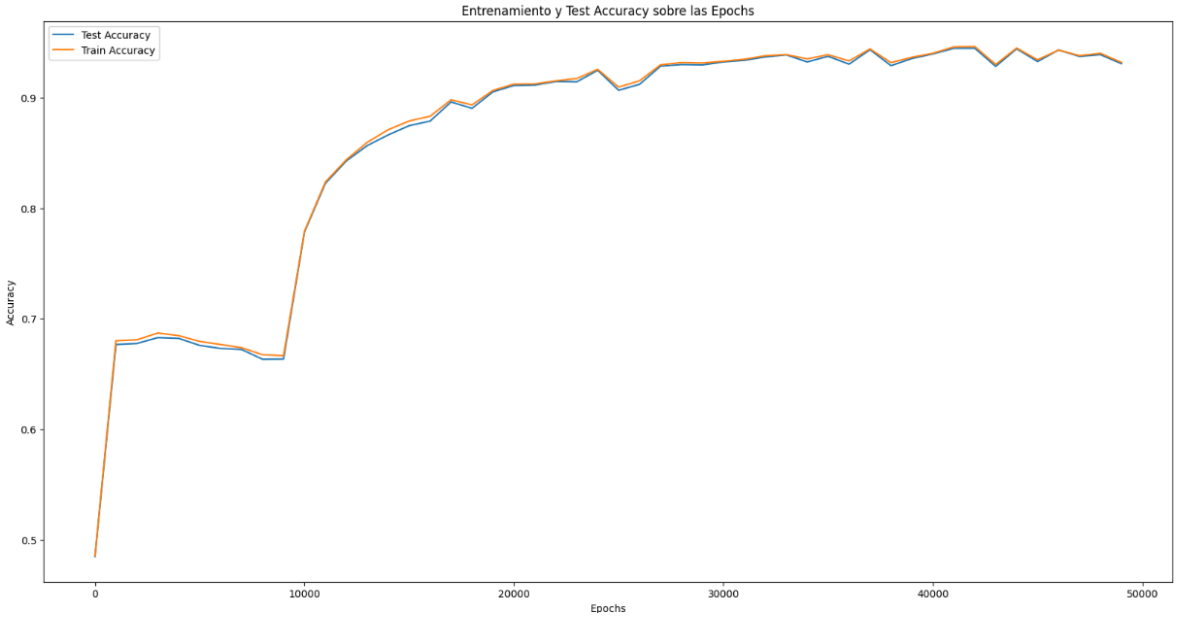
```
def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=0, keepdims=True))
    return exp_x / np.sum(exp_x, axis=0, keepdims=True)

### Construimos la red neuronal con pesos y sesgos iniciados
aleatoriamente
# Primero determinamos un seed para controlar los valores random

np.random.seed(5429)

w_hidden = (np.random.rand(3, 5) * 2) - 1 # Pesos de la capa oculta (3
neuronas, 5 entradas)
w_output = (np.random.rand(3, 3) * 2) - 1 # Pesos de la capa de salida
(3 clases, 3 neuronas ocultas)

b_hidden = (np.random.rand(3, 1) * 2) - 1 # Biases de la capa oculta (3
neuronas)
b_output = (np.random.rand(3, 1) * 2) - 1 # Biases de la capa de salida
(3 clases)

# Función del forward para recorrer la red de atrás para adelante
def forward_prop(X):
    #Z1 = w_hidden @ X.T + b_hidden
    Z1 = w_hidden @ X.T + b_hidden
    A1 = relu(Z1)
    Z2 = w_output @ A1 + b_output
    A2 = softmax(Z2)
    return Z1, A1, Z2, A2

# Función para deteminar el accuracy
def accuracy(X, Y):
    test_predictions = forward_prop(X)[3]
    predicted_classes = np.argmax(test_predictions, axis=0)  # Obtener
clase con mayor prob
    true_classes = np.argmax(Y, axis=1)
    accuracy = np.mean(predicted_classes == true_classes)
    print("Aciertos: ", (accuracy * 100).round(1), "\n")

# Printeamos el accuracy que generamos con el forward
print('Pre entrenamiento: \n')
print('Test\n')
accuracy(X_test, Y_test)
print('Train\n')
```

```
accuracy(X_train, Y_train)

# Creamos 2 listas para guardar los resultados, para luego imprimir el gráfico
conteoTest = []
conteoTrain = []

# Cálculo de precisión
def accuracyTest(X, Y):
    test_predictions = forward_prop(X)[3]  # solo nos interesa A2
    predicted_classes = np.argmax(test_predictions, axis=0)  # obtener clase con mayor probabilidad
    true_classes = np.argmax(Y, axis=1)  # etiquetas verdaderas
    accuracy = np.mean(predicted_classes == true_classes)  # porcentaje de aciertos
    # print("Porcentaje de aciertos: ", (accuracy * 100).round(2))
    conteoTest.append(accuracy)

def accuracyTrain(X, Y):
    test_predictions = forward_prop(X)[3]  # solo nos interesa A2
    predicted_classes = np.argmax(test_predictions, axis=0)  # obtener clase con mayor probabilidad
    true_classes = np.argmax(Y, axis=1)  # etiquetas verdaderas
    accuracy = np.mean(predicted_classes == true_classes)  # porcentaje de aciertos
    # print("Porcentaje de aciertos: ", (accuracy * 100).round(2))
    conteoTrain.append(accuracy)

### Devuelve pendientes para pesos y sesgos usando la regla de la cadena
# Derivada de ReLU
def d_relu(Z):
    return (Z > 0)

# Derivada de softmax
def d_softmax(muestra):
    s = muestra.reshape(-1, 1) # Convierte el vector de salida en una columna
    #reshape(-1,1): cambia la forma de "muestra" de un vector (n) a (n,1) y al pasarle "-1", numpy calcula automáticamente el tamaño apropiado para la primera dimensión
    return np.diagflat(s) - np.dot(s, s.T) # Matriz diagonal menos el producto externo
```

```
    #diagflat(s): crea una matriz diagonal, en la que los valores
diagonales son valores del vector "s"
    #dot(s, s.t): calcula el producto externo de "s" con "s" traspuesta
    #np.diagflat(s) - np.dot(s, s.T): la finalidad de esta resta, es
calcular la matriz Jocobiana de Softmax

#**********Al usar la Softmax como función de salida, es recomendable
usar el Cross Entropy para calcular el error**********************
#Cross Entropy: se usa para medir la diferencia entre la distribución
real (valores esperados) y la distribución predicha (las probabilidades
que la red cacula con Softmax)

#Expresión matemática de la Cross Entropy: - (sumatoria de Y)(log(S))
#Derivada de Cross Entropy: S - Y
        #((3,1),(3,1),(3,1),(3,1),(1,5),(3,1))
def backward_prop(Z1, A1, Z2, A2, X, Y):

    dC_dA2 = A2 - Y.T  # (3, 1)
    dA2_dZ2 = d_softmax(A2)  # (3, 3)
    dZ2_dA1 = w_output  # (3, 3)
    dZ2_dW2 = A1
    dZ2_dB2 = 1
    dA1_dZ1 = d_relu(Z1)  # (3, 1)
    dZ1_dW1 = X
    dZ1_dB1 = 1

    dC_dW2 = dA2_dZ2 @ dC_dA2  @ dZ2_dW2.T
    dC_dB2 = dA2_dZ2 @ dC_dA2
    dC_dA1 = dZ2_dA1 @ dA2_dZ2 @ dC_dA2
    dC_dW1 = dC_dA1 * dA1_dZ1 @ dZ1_dW1
    dC_dB1 = dC_dA1 * dA1_dZ1 * dZ1_dB1

    return dC_dW1, dC_dB1, dC_dW2, dC_dB2

# La tasa de aprendizaje
L = 0.01

# Ejecutar descenso de gradiente estocástico
epochs = 50000

for i in tqdm(range(epochs)):
    idx = np.random.randint(0, n)  # Elegir un solo índice aleatorio
    X_sample = X_train[idx:idx+1]  # Obtener el ejemplo
```

```
    Y_sample = Y_train[idx:idx+1]  # Obtener la etiqueta
correspondiente
    Z1, A1, Z2, A2 = forward_prop(X_sample)
    dW1, dB1, dW2, dB2 = backward_prop(Z1, A1, Z2, A2, X_sample,
Y_sample)
    # Actualizamos los pesos y biases
    w_hidden -= L * dW1
    b_hidden -= L * dB1
    w_output -= L * dW2
    b_output -= L * dB2

    # Cada 1000 épocas, almacenamos precisión en test y train
    if i % 1000 == 0:
        accuracyTrain(X_train, Y_train)  # Calcula precisión en
entrenamiento
        accuracyTest(X_test, Y_test)    # Calcula precisión en prueba


# Cálculo de precisión
print('\nPost entrenamiento: \n')
print('Test\n')
accuracy(X_test, Y_test)
print('Train\n')
accuracy(X_train, Y_train)

# Graficamos los resultados
plt.figure(figsize=(20, 10))
plt.plot(range(0, epochs, 1000), conteoTest, label="Test Accuracy")
plt.plot(range(0, epochs, 1000), conteoTrain, label="Train Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Entrenamiento y Test Accuracy sobre las Epochs")
plt.legend()
plt.show()
```

```
Aciertos:  48.5

Train

Aciertos:  48.6

100% ████████████████████████████████████████ 50000/50000 [00:08<00:00, 5892.02it/s]

Post entrenamiento:

Test

Aciertos:  94.5

Train

Aciertos:  94.7
```



Luego de realizar nuestra Red Neuronal, viendo el porcentaje de aciertos y el gráfico, nos preguntamos, ¿estamos cayendo en Overfitting?

Para poder responder esta pregunta, primero entendamos qué es Overfitting. El Overfitting (Sobreajuste) es el efecto de sobreentrenar la Red Neuronal, en otras palabras, hacer que nuestra Red Neuronal "memorice" los resultados y se sobreajuste a la hora de devolver resultados.

¿Cómo lo detectamos?

Una manera es viendo si el porcentaje de aciertos del entrenamiento es significativamente mayor al del conjunto de test y si esta diferencia se incrementa a medida que aumentamos las épocas. O también si observamos que el test se estabiliza y el entrenamiento aumenta, es otra señal de la presencia de Overfitting.

¿Qué maneras tenemos para prevenir o comprobar si estamos cayendo en Overfitting?

Hay varios métodos para prevenir o comprobar si estamos en presencia de Overfitting:

-Dropout: consiste en desactivar algunas neuronas aleatoriamente durante el entrenamiento, esto ayuda a no depender de neuronas específicas.

-Reducir la complejidad de la Red Neuronal: una forma es reducir la cantidad de neuronas en la capa oculta para evitar que la Red Neuronal aprende demasiado sobre datos específicos.

-Ajustar el número de épocas: reducir la cantidad de veces que itera nuestra red, nos ayuda a evitar que se siga entrenando una vez que dejó de mejorar el test.

-Aumentar la cantidad de datos: al tener más muestras, se reduce la posibilidad de generalizar y caer en el Overfitting.

Al observar nuestra Red Neuronal, decidimos aplicar 2 de estas técnicas para chequear si estamos en presencia de Overfitting: reducir la complejidad de la red y ajustar el número de épocas.

En este caso redujimos a 2 las neuronas de la capa oculta y a 25000 las épocas:

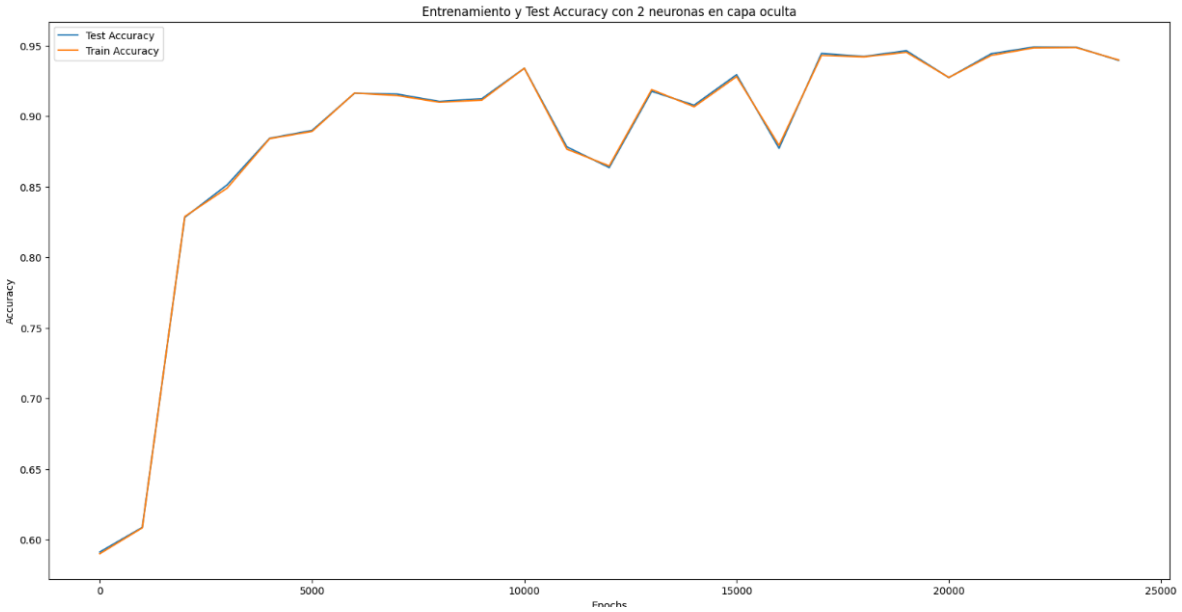
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tqdm.auto import tqdm

df_limpio =
pd.read_csv('https://raw.githubusercontent.com/casquifer/Mate-III/refs/
heads/main/df_limpio.csv')
all_inputs = df_limpio.iloc[:, 0:5].values
all_outputs = df_limpio.iloc[:, -1].values

def normalizador(X):
    promedio = np.mean(X, axis=0)
    desvEst = np.std(X, axis=0)
    return (X - promedio) / desvEst

all_inputs = normalizador(all_inputs)

respPosibles = 3
y_matriz = np.eye(respPosibles)[all_outputs]

X_train, X_test, Y_train, Y_test = train_test_split(all_inputs,
y_matriz, test_size=1/3)
n = X_train.shape[0]

relu = lambda x: np.maximum(x, 0)

def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=0, keepdims=True))
    return exp_x / np.sum(exp_x, axis=0, keepdims=True)


np.random.seed(5429)

# Iniciamos los pesos y sesgos con un número menor de neuronas en la
capa oculta (2)
w_hidden = (np.random.rand(2, 5) * 2) - 1  # 2 neuronas en capa oculta,
5 entradas
```

```
w_output = (np.random.rand(3, 2) * 2) - 1  # 3 clases, 2 neuronas en
capa oculta

b_hidden = (np.random.rand(2, 1) * 2) - 1  # Biases de la capa oculta
(2 neuronas)
b_output = (np.random.rand(3, 1) * 2) - 1  # Biases de la capa de
salida (3 clases)

def forward_prop(X):
    Z1 = w_hidden @ X.T + b_hidden
    A1 = relu(Z1)
    Z2 = w_output @ A1 + b_output
    A2 = softmax(Z2)
    return Z1, A1, Z2, A2

def accuracy(X, Y):
    test_predictions = forward_prop(X)[3]
    predicted_classes = np.argmax(test_predictions, axis=0)
    true_classes = np.argmax(Y, axis=1)
    accuracy = np.mean(predicted_classes == true_classes)
    print("Aciertos: ", (accuracy * 100).round(1), "\n")

conteoTest = []
conteoTrain = []

def accuracyTest(X, Y):
    test_predictions = forward_prop(X)[3]
    predicted_classes = np.argmax(test_predictions, axis=0)
    true_classes = np.argmax(Y, axis=1)
    accuracy = np.mean(predicted_classes == true_classes)
    conteoTest.append(accuracy)

def accuracyTrain(X, Y):
    test_predictions = forward_prop(X)[3]
    predicted_classes = np.argmax(test_predictions, axis=0)
    true_classes = np.argmax(Y, axis=1)
    accuracy = np.mean(predicted_classes == true_classes)
    conteoTrain.append(accuracy)

def d_relu(Z):
    return (Z > 0)

def backward_prop(Z1, A1, Z2, A2, X, Y):
```

```
    dC_dA2 = A2 - Y.T
    dZ2_dA1 = w_output
    dZ2_dW2 = A1
    dA1_dZ1 = d_relu(Z1)

    dC_dW2 = dC_dA2 @ dZ2_dW2.T
    dC_dA1 = dZ2_dA1.T @ dC_dA2
    dC_dW1 = (dC_dA1 * dA1_dZ1) @ X
    dC_dB2 = dC_dA2
    dC_dB1 = dC_dA1 * dA1_dZ1

    return dC_dW1, dC_dB1, dC_dW2, dC_dB2

L = 0.01
epochs = 25000

for i in tqdm(range(epochs)):
    idx = np.random.randint(0, n)
    X_sample = X_train[idx:idx+1]
    Y_sample = Y_train[idx:idx+1]
    Z1, A1, Z2, A2 = forward_prop(X_sample)
    dW1, dB1, dW2, dB2 = backward_prop(Z1, A1, Z2, A2, X_sample,
Y_sample)

    w_hidden -= L * dW1
    b_hidden -= L * dB1
    w_output -= L * dW2
    b_output -= L * dB2

    if i % 1000 == 0:
        accuracyTrain(X_train, Y_train)
        accuracyTest(X_test, Y_test)

print('\nPost entrenamiento: \n')
print('Test\n')
accuracy(X_test, Y_test)
print('Train\n')
accuracy(X_train, Y_train)

plt.figure(figsize=(20, 10))
plt.plot(range(0, epochs, 1000), conteoTest, label="Test Accuracy")
plt.plot(range(0, epochs, 1000), conteoTrain, label="Train Accuracy")
plt.xlabel("Epochs")
```

```
plt.ylabel("Accuracy")
plt.title("Entrenamiento y Test Accuracy con 2 neuronas en capa oculta")
plt.legend()
plt.show()
```

```
100% 25000/25000 [00:03<00:00, 7156.36it/s]

Post entrenamiento:

Test

Aciertos:  94.3

Train

Aciertos:  94.2
```



Como podemos observar, los resultados de aciertos y los gráficos son bastantes similares, ambos muestran un rápido aprendizaje trabajando tanto con 3 o 2 neuronas en la capa oculta y con 50000 y 25000 épocas.

Si reducimos las épocas a menos de 50000 en la red de 3 neuronas, el porcentaje de aciertos comienza a descender, por lo que entendemos que no está memorizando los resultados por cantidad de épocas.

Dado que el segundo modelo con menos neuronas y menos épocas, obtuvo resultados similares podemos deducir que el primer modelo no

presenta una tendencia al Overfitting. Finalmente podemos entender que también podría funcionar nuestra red con menos recursos, dado que los

Red hecha con Scikit-Learn:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

df_limpio =
pd.read_csv('https://raw.githubusercontent.com/casquifer/Mate-III/refs/
heads/main/df_limpio.csv')

# Extraer variables de entrada (todas las filas, todas las columnas
menos de "class")
X = (df_limpio.values[:, :-1])

# Extraer columna de salida (todas las filas, columna "class")
Y = df_limpio.values[:, -1]

scaler = StandardScaler()
X = scaler.fit_transform(X)

# Separar los datos de entrenamiento y prueba
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=1/3)

#red
nn = MLPClassifier(solver='sgd',
                   hidden_layer_sizes=(3, ),
                   activation='relu',
                   max_iter=50_000,
                   learning_rate_init=.01)

#entrenamiento
nn.fit(X_train, Y_train)

print("Porcentaje de aciertos con train: ", (nn.score(X_train,
Y_train)*100))
```

```
print("Porcentaje de aciertos con test: ", (nn.score(X_test,
Y_test)*100))
```

```
Porcentaje de aciertos con train:  95.96466093600765
Porcentaje de aciertos con test:  96.03311047437121
```

Comparación con Scikit-Learn

Como podemos observar, los resultados de ambas redes son bastante similares, al igual que el tiempo de ejecución. Si bien scikit le saca un puntito de ventaja, de buenas primeras no estamos en condiciones de saber si está cayendo en un overfitting. Tampoco podemos controlar que funciones usa internamente. Por otro lado vemos la simpleza con la que obtenemos resultados, casi que podríamos decir que una persona que no sabe cómo funciona una red neuronal por dentro, puede crearse una, siempre y cuando tenga las herramientas adecuadas.

Conclusión Final

En este hermoso viaje de crear una Red Neuronal desde cero, hemos aprendido a estudiar un data frame, a saber cómo observar y analizar a fondo. Primero analizando sus datos columna a columna, determinando que variables no son útiles y cuales nos llevarían a complicaciones, como por ejemplo los datos atípicos. Pasando por la normalización de los datos hasta analizar los histogramas. Con estas detecciones y análisis podríamos saber si aplicamos o no un recorte de cuartiles a nuestro data frame, para poder emprolijar lo más posible. Finalmente analizamos las correlaciones entre los datos y la columna de salida, para poder dar esa última limpieza al data frame con el fin dejarlo listo para poder crear una Red Neuronal.

Como ya comparamos antes, podemos poner en una balanza los pros y contras de ambas formas de crear una Red Neuronal, pero a nuestro criterio, siempre es mejor tener el control total de lo que uno está haciendo. Es por eso que no nos conformamos con la comodidad de una librería (que vaya uno a saber qué cálculos y uso del CPU haga) y elegimos hacer las cosas a mano, entendiendo y controlando todo lo que pasa dentro de nuestra Red Neuronal.

En resumen, hemos aprendido mucho sobre el análisis y limpieza de data frames y sobre la construcción y análisis de Redes Neuronales.

Autores

CASCO, Rodrigo

GÓMEZ CIRANA, Jonathan