MLP

August 12, 2025

Técnicas de Inteligencia Artificial

Redes neuronales y Deep Learning

1 Importación de librerias necesarias

[3]:
```
#Se importarán las siguientes librerías:

#import pathlib

#import matplotlib.pyplot as plt
#import pandas as pd
#import seaborn as sns

#from tensorflow import keras
#from tensorflow.keras import layers
#import tensorflow as tf
#from tensorflow.keras.callbacks import Callback

#from sklearn.model_selection import train_test_split
#from sklearn.preprocessing import StandardScaler
#from sklearn.metrics import classification_report
#from tensorflow.keras.models import Sequential
#from tensorflow.keras.layers import Dense

#from sklearn.ensemble import RandomForestClassifier
#from sklearn.linear_model import LogisticRegression
#from sklearn.metrics import accuracy_score, classification_report

#import kagglehub
#import pandas as pd
#import os
```

[ ]:
```

```

1.1 Cargar el Dataset

Con al menos 1000 instancias, una variable/atributo de la salida, y que dependa de, al menos, 6 variables/atributos de entrada.

[19]:
```
# Paso 1: Descargar el dataset
path = kagglehub.dataset_download("yasserh/wine-quality-dataset")

# Paso 2: Verificar el contenido del directorio descargado
print(f"Dataset descargado en: {path}")
print("Contenido del directorio:")
print(os.listdir(path))  # Lista los archivos en el directorio

# Paso 3: Ajustar el nombre correcto del archivo CSV
csv_file = os.path.join(path, "WineQT.csv")  # Nombre correcto del archivo CSV
print(f"Cargando archivo CSV: {csv_file}")

# Paso 4: Cargar el dataset
dataset = pd.read_csv(csv_file)

# Paso 5: Mostrar las primeras 5 filas del dataset
print("Primeras 5 filas del dataset:")
print(dataset.head(5))
```

```
Dataset descargado en: C:\Users\pedur\.cache\kagglehub\datasets\yasserh\wine-
quality-dataset\versions\1
Contenido del directorio:
['WineQT.csv']
Cargando archivo CSV: C:\Users\pedur\.cache\kagglehub\datasets\yasserh\wine-
quality-dataset\versions\1\WineQT.csv
Primeras 5 filas del dataset:
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality  Id
0      9.4        5   0
1      9.8        5   1
2      9.8        5   2
```

```
3      9.8        6   3
4      9.4        5   4
```

1.2 Descripción de la fuente del Dataset

Este conjunto de datos está relacionado con las variantes tintas del vino portugués "Vinho Verde". Describe la cantidad de diferentes compuestos químicos presentes en el vino y su efecto en la calidad del mismo. Las clases están ordenadas, pero no están equilibradas (por ejemplo, hay muchos más vinos de calidad normal que excelentes o deficientes). El dataset puede encontrarse en ' https://www.kaggle.com/api/v1/datasets/download/yasserh/wine-quality-dataset'

1.3 Problema a resolver.

El problema consiste en predecir la calidad del vino en base al resto de variables de entrada. Por lo tanto, la variable respuesta será 'quality' y el resto de variables serán las variables de entrada. La utilidad de obtener un modelo que prediga la calidad reside en la capacidad, tanto del consumidor casual como en la compra al pormayor de vino, permitiendo comprar vino de mejor calidad sin tener que degustarlo, ahorrando así costes.

1.4 Caracterización del Dataset

[4]:
```
dataset.dropna()
```

[4]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
…               …                 …            …                …        …
1138            6.3             0.510         0.13             2.3      0.076
1139            6.8             0.620         0.08             1.9      0.068
1140            6.2             0.600         0.08             2.0      0.090
1141            5.9             0.550         0.10             2.2      0.062
1142            5.9             0.645         0.12             2.0      0.075

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
…                     …                     …       …   …          …
1138                 29.0                  40.0  0.99574  3.42       0.75
1139                 28.0                  38.0  0.99651  3.42       0.82
1140                 32.0                  44.0  0.99490  3.45       0.58
1141                 39.0                  51.0  0.99512  3.52       0.76
1142                 32.0                  44.0  0.99547  3.57       0.71
```

```
      alcohol  quality    Id
0         9.4        5     0
1         9.8        5     1
2         9.8        5     2
3         9.8        6     3
4         9.4        5     4
…         …        …    …
1138     11.0        6  1592
1139      9.5        6  1593
1140     10.5        5  1594
1141     11.2        6  1595
1142     10.2        5  1597

[1143 rows x 13 columns]
```

[5]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64
 12  Id                    1143 non-null   int64
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
```

[6]:
```
dataset.shape
```

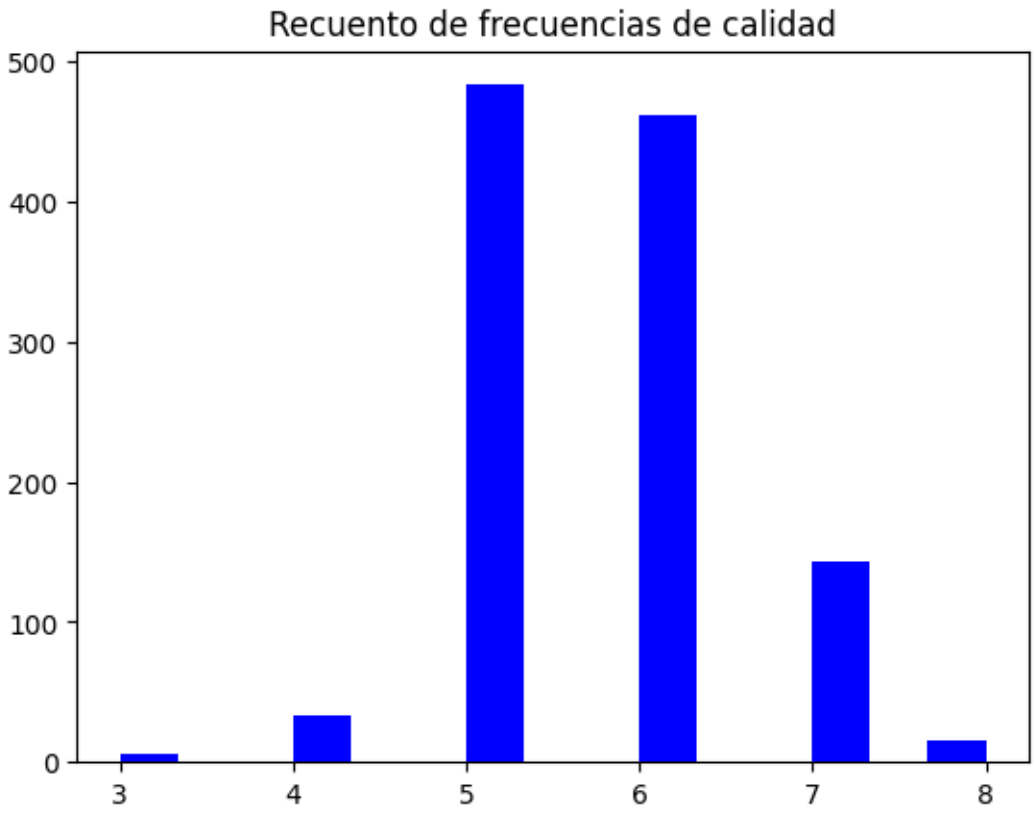
[6]:
```
(1143, 13)
```

[7]:
```
dataset['quality'].unique()
```

[7]:
```
array([5, 6, 7, 4, 8, 3])
```

[8]:
```
dataset.quality.hist(bins=15, grid=False, color="blue").set_title("Recuento de␣
↪frecuencias de calidad")
```

[8]:
```
Text(0.5, 1.0, 'Recuento de frecuencias de calidad')
```



[9]:
```
dataset['quality'].value_counts()
```

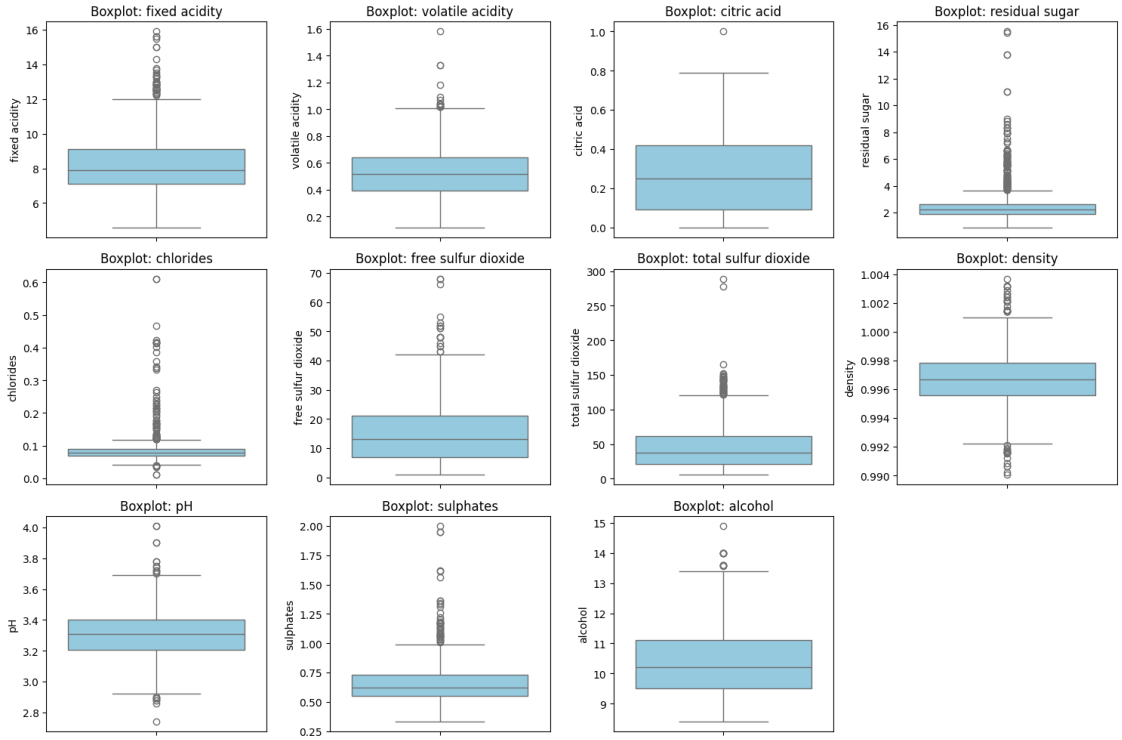
[9]:
```
quality
5    483
6    462
7    143
4     33
8     16
3      6
Name: count, dtype: int64
```

[ ]:

[ ]:

Vamos a realizar un diagrama de caja para las variables de entrada (salvo Id, pues la eliminaremos en el procesamiento de los datos por no aportar información)

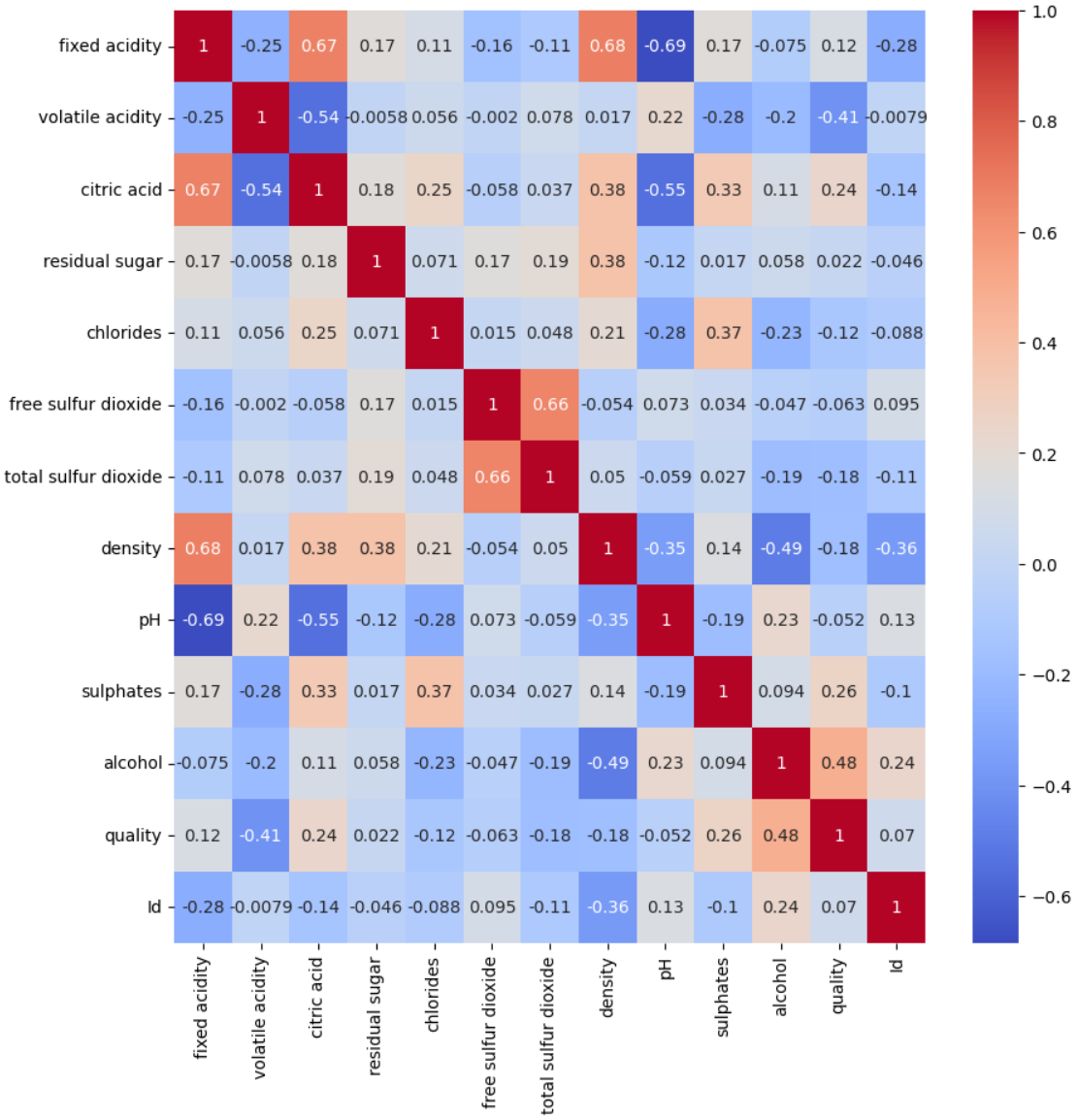
[10]:
```
numerical_cols = dataset.select_dtypes(include=['float64']).columns
plt.figure(figsize=(15, 10))
columnas = 4  # Número de columnas en la cuadrícula
filas = -(-len(numerical_cols) // columnas)  # Calcular filas necesarias␣
 ↪(redondeo hacia arriba)

for i, col in enumerate(numerical_cols, 1):
    plt.subplot(filas, columnas, i)
    sns.boxplot(dataset[col], color='skyblue')
    plt.title(f"Boxplot: {col}")


plt.tight_layout()  # Ajustar el espacio entre los subgráficos
plt.show()
```



Los diagramas de caja sugieren que las variables de entrada no siguen distribución normal, lo cual sugiere que posiblemente requiera que normalicemos los datos para trabajar con ellos.

[ ]:
```
```

[11]:
```
plt.figure(figsize=(10,10))
corr = dataset.corr()
sns.heatmap(corr ,annot=True , cmap= 'coolwarm');
```



[12]:
```
#Código que responde a la descripción anterior (incorpore las lineas de code necesarias. Describa cadas sentencia de código)
```

En un par de párrafos haga un resumen de los principales hallazagos encontrados:

El conjunto de datos contiene un total de 1143 instancias. Hay 12 variables de entrada (todas son variables cuantitativas continuas, salvo Id) Estos son:

- fixed acidity: Nivel de acidez fija en el vino.
- volatile acidity: Nivel de acidez volátil, asociado a aromas no deseados.
- citric acid: Cantidad de ácido cítrico, que contribuye a la frescura.
- residual sugar: Azúcares residuales que permanecen tras la fermentación.
- chlorides: Cantidad de cloruros, relacionado con la salinidad.

- free sulfur dioxide: Dióxido de azufre libre, que actúa como conservante.
- total sulfur dioxide : Dióxido de azufre total.
- density: Densidad del vino, relacionada con su contenido de azúcar y alcohol.
- pH: Medida de acidez.
- sulphates: Sulfatos que influyen en la estabilidad microbiana.
- alcohol : Porcentaje de alcohol por volumen.
- Id : Identificador único de cada muestra. Entero

La variable objetivo, quality, tiene 6 clases diferentes, que representan la calidad del vino según una escala discreta. Los valores y su significado son:

- 3: Muy baja calidad.
- 4: Baja calidad.
- 5: Calidad media.
- 6: Calidad ligeramente superior a la media.
- 7: Buena calidad.
- 8: Muy buena calidad. Esta variable es de tipo entero y discreta.

El número de instancias pertenecientes a cada clase en la variable objetivo: - Clase 3: 6 instancias. - Clase 4: 33 instancias. - Clase 5: 483 instancias. - Clase 6: 462 instancias. - Clase 7: 143 instancias. - Clase 8: 16 instancias.

Estadísticas de la variable objetivo: La distribución de la calidad muestra que la mayoría de los vinos tienen una calidad entre 5 y 6, lo cual indica que predominan las calidades medias. Las calidades extremas (3 y 8) son las menos representadas. Esto indica normalidad en la distribución de los datos

Se puede observar que volatile acidity tiene correlación de nivel medio (proporcionalidad inversa), al igual que alcohol (proporcionalidad directa en este caso) Así mismo, las demás variables de entrada tienen relación débil.

1.5 Preprocesamiento del dataset. Transformaciones previas necesarias para la modelación

[14]:
```
data=dataset
```

[15]:
```
#Eliminamos la variable ID porque no nos aporta ninguna información
data=data.drop('Id', axis=1)
```

[ ]:
```

```

[16]:
```
def categorize(value):
    if value == 3 :
        return 'Tres'
    elif value == 4 :
        return 'Cuatro'
    elif value == 5 :
        return 'Cinco'
    elif value == 6 :
        return 'Seis'
```

```
    elif value == 7 :
        return 'Siete'
    else:
        return 'Ocho'

# Aplicar la función
data['quality'] = dataset['quality'].apply(categorize)
```

[17]:
```
data['Tres'] = (data['quality'] == 'Tres') * 1.0
data['Cuatro'] = (data['quality'] == 'Cuatro') * 1.0
data['Cinco'] = (data['quality'] == 'Cinco') * 1.0
data['Seis'] = (data['quality'] == 'Seis') * 1.0
data['Siete'] = (data['quality'] == 'Siete') * 1.0
data['Ocho'] = (data['quality'] == 'Ocho') * 1.0
```

[ ]:
```

```

1.6 División del dataset en datos de entrenamiento y datos de test

[19]:
```
# Seleccionar las últimas 6 columnas
y = data.iloc[:, -6:].copy()  # Crea un nuevo DataFrame con las últimas 6␣
 ↪columnas

# Eliminar las últimas 6 columnas del DataFrame original y quality
X = data.drop('quality', axis=1)
X = X.drop('Tres', axis=1)
X = X.drop('Cuatro', axis=1)
X = X.drop('Cinco', axis=1)
X = X.drop('Seis', axis=1)
X = X.drop('Siete', axis=1)
X = X.drop('Ocho', axis=1)

#print("DataFrame sin las últimas 6 columnas (X):")
#print(X)

#print("\nNuevo DataFrame con las últimas 6 columnas (Y):")
#print(y)
```

[20]:
```
# Normalicemos las variables de entrada
scaler = StandardScaler()
X_norm= scaler.fit_transform(X)
```

[21]:
```
# Dividir en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size=0.2,␣
 ↪random_state=0)
```

[ ]:
```

```

[ ]:

1.7 Propuesta de arquitectura de red neuronal

La red neuronal propuesta tiene una estructura secuencial y está diseñada para abordar problemas de clasificación multiclase. Está compuesta por una capa de entrada, dos capas ocultas intermedias y una capa de salida. A continuación, se detalla la arquitectura: - Capa de entrada Tiene 64 neuronas. Emplea como funnción de activación ReLU (Rectified Linear Unit), configurada para procesar las características de entrada, con una dimensión definida por X_train.shape[1].

- Primera capa intermedia: emplea 64 neuronas. la función de activación es ReLU. Está diseñada para aprender relaciones no lineales en los datos.
- Segunda capa intermedia: compuesta 32 neuronas. La función de activación es ReLU.
- Capa de salida: el número de neuronas es igual al número de clases del problema (y.shape[1]). En este caso 6. La función de activación es Softmax que genera una distribución de probabilidad sobre las clases posibles, adecuada para problemas de clasificación multiclase.

Configuración del Modelo

Se utiliza el optimizador Adam, configurado con una tasa de aprendizaje inicial de 0.001.

La función de pérdida seleccionada es Categorical Crossentropy (Entropía Cruzada Categórica), adecuada para evaluar discrepancias entre las etiquetas reales y las predicciones del modelo en problemas de clasificación multiclase.

Se utiliza la métrica de exactitud (accuracy) para evaluar el desempeño del modelo durante el entrenamiento y la validación.

[22]:
```
# Código de la estructuración de la red
```

[23]:
```
# Definir la arquitectura del modelo
def build_model():
    model = keras.Sequential([
        layers.Dense(64, activation='relu', input_shape=[X_train.shape[1]]),  #␣
 ↪Capa de entrada
        layers.Dense(64, activation='relu'),  # Primera capa intermedia
        layers.Dense(32, activation='relu'),  # Segunda capa intermedia
        layers.Dense(y.shape[1], activation='softmax')  # Capa de salida para␣
 ↪clasificación
    ])

    # Configuración del optimizador
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)  # Optimizador␣
 ↪Adam

    # Compilación del modelo
    model.compile(loss='categorical_crossentropy',  # Pérdida: Entropía cruzada␣
 ↪categórica
                  optimizer=optimizer,            # Optimizador Adam
```

```
                  metrics=['accuracy'])           # Métricas: Exactitud

    return model
```

[24]:
```
Clas_RNA=build_model()
```

```
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[25]:
```
#Código de la inspección del modelo de red
```

[26]:
```
Clas_RNA.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	768
dense_1 (Dense)	(None, 64)	4,160
dense_2 (Dense)	(None, 32)	2,080
dense_3 (Dense)	(None, 6)	198

Total params: 7,206 (28.15 KB)

Trainable params: 7,206 (28.15 KB)

Non-trainable params: 0 (0.00 B)

1.8 Ajuste de modelo de Clasificación RNA

[27]:
```
# Ejemplo de predicción con un batch pequeño (10 muestras)
example_batch = X_train[:10]
example_result = Clas_RNA.predict(example_batch)
print("Ejemplo de predicciones con 10 muestras:", example_result)

# Clase para imprimir puntos durante el entrenamiento (Callback personalizado)
class PrintDot(Callback):
```

```
    def on_epoch_end(self, epoch, logs=None):
        if epoch % 10 == 0:  # Cambié a 10 para hacerlo más visible en el␣
↪ejemplo
            print('')
        print('.', end='')
```

```
1/1             0s 113ms/step
Ejemplo de predicciones con 10 muestras: [[0.19506513 0.11970367 0.13603957
0.21876366 0.15083636 0.17959157]
 [0.16709827 0.17000076 0.14048964 0.14809987 0.17185554 0.20245594]
 [0.20553136 0.1382069  0.10741565 0.19214205 0.18754548 0.16915856]
 [0.1720589  0.16020687 0.15950605 0.15455788 0.17043737 0.18323292]
 [0.17618395 0.15063113 0.14156222 0.16289337 0.16823871 0.20049062]
 [0.1881073  0.14938232 0.14287594 0.19979486 0.1389159  0.18092366]
 [0.16095376 0.1626304  0.12324324 0.13827862 0.17974785 0.23514616]
 [0.1882744  0.13723993 0.07957418 0.2024655  0.17741135 0.2150346 ]
 [0.20263192 0.14817968 0.11348763 0.1865686  0.16791898 0.18121313]
 [0.41700843 0.05756908 0.05401776 0.2348554  0.11896951 0.11757988]]
```

[28]:
```
#Código de ajuste y entrenamiento
```

[29]:
```
# Número de epoch
EPOCHS = 20

# Entrenar el modelo con validación y callback para imprimir puntos
history = Clas_RNA.fit(
    X_train, y_train,
    epochs=EPOCHS, validation_split=0.2, verbose=1,
    callbacks=[PrintDot()]
)
```

```
Epoch 1/20
 1/23             21s 995ms/step - accuracy:
0.2500 - loss: 1.8164
23/23            1s 12ms/step -
accuracy: 0.3041 - loss: 1.7459 - val_accuracy: 0.5683 - val_loss: 1.4078
Epoch 2/20
23/23            0s 5ms/step -
accuracy: 0.5465 - loss: 1.3331 - val_accuracy: 0.6011 - val_loss: 1.1107
Epoch 3/20
23/23            0s 5ms/step -
accuracy: 0.5802 - loss: 1.1387 - val_accuracy: 0.5956 - val_loss: 1.0201
Epoch 4/20
23/23            0s 5ms/step -
accuracy: 0.5711 - loss: 1.0532 - val_accuracy: 0.6175 - val_loss: 0.9903
Epoch 5/20
23/23            0s 5ms/step -
accuracy: 0.5926 - loss: 0.9700 - val_accuracy: 0.6230 - val_loss: 0.9838
```

```
Epoch 6/20
23/23             0s 5ms/step -
accuracy: 0.6263 - loss: 0.9512 - val_accuracy: 0.6230 - val_loss: 0.9613
Epoch 7/20
23/23             0s 5ms/step -
accuracy: 0.6072 - loss: 0.9596 - val_accuracy: 0.6339 - val_loss: 0.9629
Epoch 8/20
23/23             0s 5ms/step -
accuracy: 0.6331 - loss: 0.9123 - val_accuracy: 0.6612 - val_loss: 0.9462
Epoch 9/20
23/23             0s 5ms/step -
accuracy: 0.6824 - loss: 0.8436 - val_accuracy: 0.6557 - val_loss: 0.9478
Epoch 10/20
23/23             0s 5ms/step -
accuracy: 0.6694 - loss: 0.8751 - val_accuracy: 0.6393 - val_loss: 0.9413
Epoch 11/20
 1/23             0s 31ms/step - accuracy:
0.6250 - loss: 0.8229
23/23             0s 5ms/step -
accuracy: 0.6340 - loss: 0.8902 - val_accuracy: 0.6503 - val_loss: 0.9365
Epoch 12/20
23/23             0s 5ms/step -
accuracy: 0.6884 - loss: 0.7979 - val_accuracy: 0.6503 - val_loss: 0.9385
Epoch 13/20
23/23             0s 5ms/step -
accuracy: 0.6648 - loss: 0.7964 - val_accuracy: 0.6503 - val_loss: 0.9395
Epoch 14/20
23/23             0s 5ms/step -
accuracy: 0.6325 - loss: 0.8644 - val_accuracy: 0.6339 - val_loss: 0.9448
Epoch 15/20
23/23             0s 4ms/step -
accuracy: 0.6645 - loss: 0.8347 - val_accuracy: 0.6612 - val_loss: 0.9296
Epoch 16/20
23/23             0s 5ms/step -
accuracy: 0.6918 - loss: 0.7795 - val_accuracy: 0.6557 - val_loss: 0.9365
Epoch 17/20
23/23             0s 5ms/step -
accuracy: 0.6378 - loss: 0.8591 - val_accuracy: 0.6667 - val_loss: 0.9392
Epoch 18/20
23/23             0s 5ms/step -
accuracy: 0.6945 - loss: 0.7731 - val_accuracy: 0.6503 - val_loss: 0.9420
Epoch 19/20
23/23             0s 5ms/step -
accuracy: 0.7080 - loss: 0.7282 - val_accuracy: 0.6230 - val_loss: 0.9521
Epoch 20/20
23/23             0s 5ms/step -
accuracy: 0.7040 - loss: 0.7553 - val_accuracy: 0.6448 - val_loss: 0.9406
```

1.9 Evaluación de modelo RNA

[31]:
```
#Código de evaluación de la red propuesta (evaluación conjunto de test)
```

[32]:
```
# Evaluar el modelo en el conjunto de prueba
loss, accuracy = Clas_RNA.evaluate(X_test, y_test)
print(f"Pérdida: {loss}, Exactitud: {accuracy}")
```

```
8/8             0s 6ms/step -
accuracy: 0.6160 - loss: 0.9524
Pérdida: 0.9818258285522461, Exactitud: 0.6157205104827881
```

[54]:
```
# Predecir las clases en el conjunto de prueba
predicciones = Clas_RNA.predict(X_test)
predicciones_clases = predicciones.argmax(axis=1)
print("Predicciones (clases):", predicciones_clases[:10])
```

```
8/8             0s 5ms/step
Predicciones (clases): [2 2 2 3 2 3 3 3 3 3]
```

[34]:
```
# Convertir las predicciones en clases (índices de las clases más probables)
predicciones_clases = predicciones.argmax(axis=1)
```

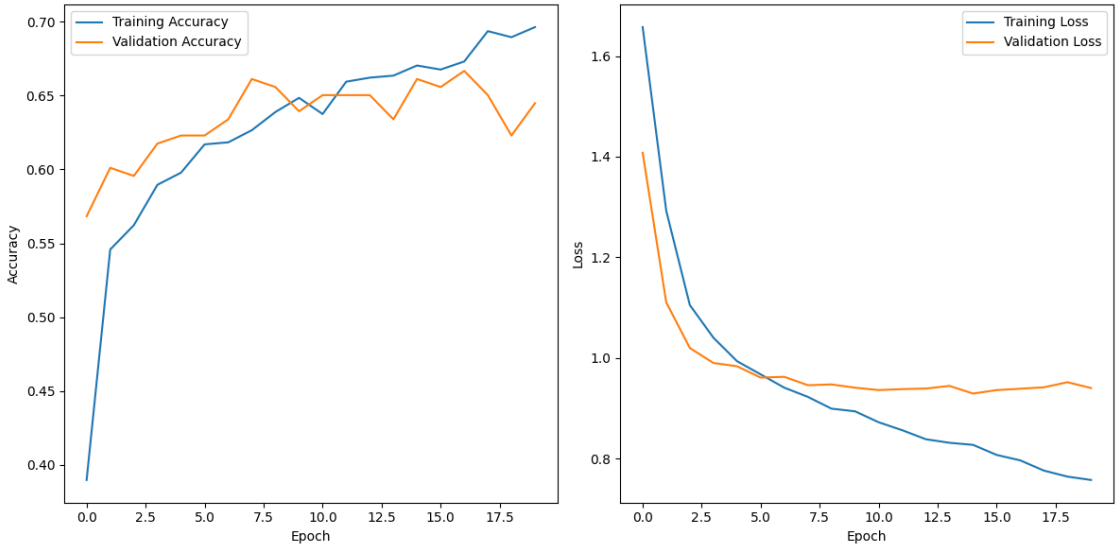
[35]:
```
etiquetas = y.columns.tolist()  # Las etiquetas corresponden a los nombres de␣
↪las columnas de 'y'

# Mapear las predicciones de clase a etiquetas
predicciones_etiquetas = [etiquetas[i] for i in predicciones_clases]

# Imprimir las predicciones en términos de las etiquetas
print("Primeras 10 predicciones (etiquetas):", predicciones_etiquetas[:10])
```

```
Primeras 10 predicciones (etiquetas): ['Cinco', 'Cinco', 'Cinco', 'Seis',
'Cinco', 'Seis', 'Seis', 'Seis', 'Seis', 'Seis']
```

[ ]:
```

```

[36]:
```
# Función para graficar el progreso del entrenamiento
def plot_history(history):
    # Graficar la precisión y la pérdida
    hist = pd.DataFrame(history.history)
    hist['epoch'] = history.epoch

    # Visualización de la precisión de entrenamiento y validación
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.plot(hist['epoch'], hist['accuracy'], label='Training Accuracy')
```

```
    plt.plot(hist['epoch'], hist['val_accuracy'], label='Validation Accuracy')
    plt.legend()

    # Visualización de la pérdida de entrenamiento y validación
    plt.subplot(1, 2, 2)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.plot(hist['epoch'], hist['loss'], label='Training Loss')
    plt.plot(hist['epoch'], hist['val_loss'], label='Validation Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()

# Visualizar el progreso del entrenamiento
plot_history(history)

# Mostrar estadísticas de evaluación en los conjuntos de entrenamiento y␣
 ↪validación
train_accuracy = history.history['accuracy'][-1]
val_accuracy = history.history['val_accuracy'][-1]
train_loss = history.history['loss'][-1]
val_loss = history.history['val_loss'][-1]

print(f"Última precisión en el conjunto de entrenamiento: {train_accuracy:.4f}")
print(f"Última precisión en el conjunto de validación: {val_accuracy:.4f}")
print(f"Última pérdida en el conjunto de entrenamiento: {train_loss:.4f}")
print(f"Última pérdida en el conjunto de validación: {val_loss:.4f}")
```



```
Última precisión en el conjunto de entrenamiento: 0.6963
Última precisión en el conjunto de validación: 0.6448
Última pérdida en el conjunto de entrenamiento: 0.7580
Última pérdida en el conjunto de validación: 0.9406
```

[ ]:

[37]:
```python
# Evaluar el modelo en el conjunto de prueba
test_loss, test_accuracy = Clas_RNA.evaluate(X_test, y_test, verbose=0)

# Mostrar estadísticas de evaluación del conjunto de prueba
print(f"\nEstadísticas de evaluación para el conjunto de prueba:")
print(f"Pérdida en el conjunto de prueba: {test_loss:.4f}")
print(f"Exactitud en el conjunto de prueba: {test_accuracy:.4f}")

# Obtener las predicciones del conjunto de prueba
predicciones = Clas_RNA.predict(X_test)

# Convertir las predicciones a clases
predicciones_clases = predicciones.argmax(axis=1)

# Convertir las etiquetas verdaderas (y_test) a clases numéricas
y_true_clases = y_test.to_numpy().argmax(axis=1)  # Convertir a índices␣
 ↪numéricos

# Evaluar con métricas adicionales como clasificación
print("\nReporte de clasificación (precisión, recall, F1-score):")
# Imprimir el reporte de clasificación
print(classification_report(y_true_clases, predicciones_clases))
```

```
Estadísticas de evaluación para el conjunto de prueba:
Pérdida en el conjunto de prueba: 0.9818
Exactitud en el conjunto de prueba: 0.6157
8/8             0s 4ms/step

Reporte de clasificación (precisión, recall, F1-score):
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.00      0.00      0.00         7
           2       0.70      0.76      0.73       100
           3       0.56      0.60      0.58        92
           4       0.48      0.37      0.42        27
           5       0.00      0.00      0.00         2

    accuracy                           0.62       229
   macro avg       0.29      0.29      0.29       229
```

```
weighted avg       0.58      0.62      0.60       229


C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

[ ]:

```
```

1.10 Ajuste de modelos de clasificación alternativos

Vamos a comparar el modelo con Random Forest y un modelo de regresión logística

[38]:
```
# Convertir etiquetas one-hot a índices de clase usando Numpy
y_train_class = y_train.to_numpy().argmax(axis=1)  # Convierte a Numpy y aplica
 ↪argmax
y_test_class = y_test.to_numpy().argmax(axis=1)    # Convierte a Numpy y aplica
 ↪argmax
```

[39]:
```
# Inicialización de los modelos
rf_model = RandomForestClassifier(n_estimators=100, max_depth=10,
 ↪random_state=42)
logreg_model = LogisticRegression(C=1.0, solver='lbfgs', max_iter=1000,
 ↪random_state=42)
```

[40]:
```
#Código de ajuste del modelo 1
```

[41]:
```
rf_model.fit(X_train, y_train_class)  # Usar y_train_class, que tiene las
 ↪clases en formato índice
```

[41]:
```
RandomForestClassifier(max_depth=10, random_state=42)
```

[42]:
```
#Código de ajuste del modelo 2
```

[43]:
```
logreg_model.fit(X_train, y_train_class)  # Usar y_train_class también
```

[43]:
```
LogisticRegression(max_iter=1000, random_state=42)
```

[44]:
```
#Código para mostrar la evaluación de los modelos
```

[45]:
```
# Predicciones de prueba
rf_pred = rf_model.predict(X_test)
logreg_pred = logreg_model.predict(X_test)

# Resultados
print("Random Forest - Clasificación Report:\n",␣
 ↪classification_report(y_test_class, rf_pred))
print("Regresión Logística - Clasificación Report:\n",␣
 ↪classification_report(y_test_class, logreg_pred))
```

```
Random Forest - Clasificación Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.00      0.00      0.00         7
           2       0.73      0.80      0.77       100
           3       0.62      0.68      0.65        92
           4       0.63      0.44      0.52        27
           5       0.00      0.00      0.00         2

    accuracy                           0.68       229
   macro avg       0.33      0.32      0.32       229
weighted avg       0.65      0.68      0.66       229

Regresión Logística - Clasificación Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.00      0.00      0.00         7
           2       0.74      0.84      0.79       100
           3       0.59      0.62      0.61        92
           4       0.37      0.26      0.30        27
           5       0.00      0.00      0.00         2

    accuracy                           0.65       229
   macro avg       0.28      0.29      0.28       229
weighted avg       0.61      0.65      0.62       229


C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
```

```
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Construya un o dos párrafos con los principales hallazgos. Incluye una explicación de los parámetros que consideres relevantes en cada ejecución.

Se puede apreciar una precisión en torno al 68% en el caso del RF y del 65% en el caso de la regresión logística. Teniendo en cuenta que son datos reales, la precisión es aceptable. También hay que remarcar que clasifica mal las clases 'Tres' y 'Ocho', debido a que hay pocas instancias para entrenar esas clases en concreto

1.11 Comparación del desempeño de modelos

[46]:
```
#Código para mostrar la comparación de métricas de desempeño de las dos␣
↪propuestas en tabla
```

[52]:
```
# Métricas de los modelos
metrics = {
    'Model': ['Clas_RNA', 'Random Forest', 'Regresión Logística'],
    'Precisión': [val_accuracy, accuracy_score(y_test_class, rf_pred),␣
↪accuracy_score(y_test_class, logreg_pred)],
    'F1-Score': [
        classification_report(y_test_class, predicciones_clases,␣
↪output_dict=True)['weighted avg']['f1-score'],
        classification_report(y_test_class, logreg_pred,␣
↪output_dict=True)['weighted avg']['f1-score'],
```

```python
        classification_report(y_test_class, rf_pred,␣
↪output_dict=True)['weighted avg']['f1-score']
    ],
    'Pérdida': [val_loss, None, None]  # Asumiendo que no tenemos la pérdida␣
↪para los modelos de sklearn
}

# Crear DataFrame para visualizar métricas
import pandas as pd
df_metrics = pd.DataFrame(metrics)

# Mostrar tabla de comparación
print(df_metrics)
```

```
                 Model  Precisión  F1-Score   Pérdida
0             Clas_RNA   0.644809  0.598085  0.940584
1        Random Forest   0.676856  0.623921       NaN
2  Regresión Logística   0.646288  0.658097       NaN
```

```
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
```

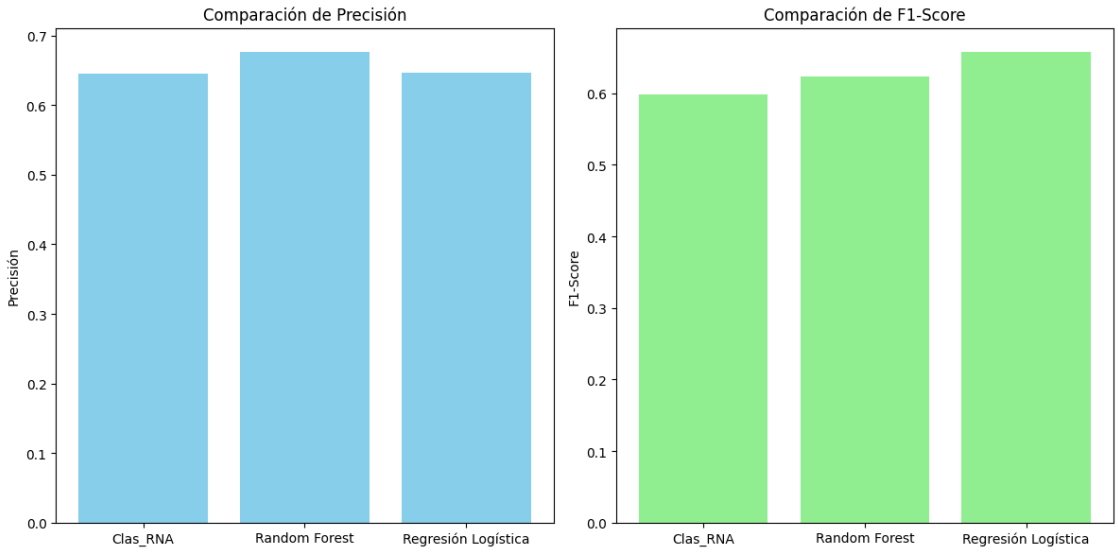
```
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Pedro\anaconda3\envs\tia_act2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

[48]:

```
#Código para mostrar la comparación de métricas de desempeño de las dos␣
↪propuestas en gráfica
```

[53]:

```
# Graficar resultados
plt.figure(figsize=(12, 6))

# Graficar precisión
plt.subplot(1, 2, 1)
plt.bar(df_metrics['Model'], df_metrics['Precisión'], color='skyblue')
plt.ylabel('Precisión')
plt.title('Comparación de Precisión')

# Graficar F1-Score
plt.subplot(1, 2, 2)
plt.bar(df_metrics['Model'], df_metrics['F1-Score'], color='lightgreen')
plt.ylabel('F1-Score')
plt.title('Comparación de F1-Score')

plt.tight_layout()
plt.show()
```



Random Forest como la Regresión Logística lograron una precisión de 0.676856 y 0.646288 y un F1-Score de 0.623921 y 0.658097 , respectivamente, números parecidos a los obtenidos por RNA. Los tres modelos parecen tener un desempeño parecido.

1.12 Discusión de los resultados obtenidos y argumentos sobre cómo se podrían mejorar de dichos resultados

- Resultados comparados:

Los resultados obtenidos de los modelos de clasificación muestran un desempeño aceptable en las tres técnicas evaluadas. El modelo Clas_RNA alcanzó una precisión de 0.644809 y un F1-Score de 0.0.598085, con una pérdida de 0.940584, lo que indica balance aceptable entre precisión y capacidad para generalizar sin sobreajustarse a los datos de entrenamiento.

Por otro lado, tanto el Random Forest como la Regresión Logística lograron una precisión de 0.676856 y 0.646288 y un F1-Score de 0.623921 y 0.658097 , respectivamente. En términos generales, los tres modelos parecen tener un desempeño aceptable.

- Conclusiones:

Hemos obtenido que todos los modelos son aceptables a la hora de hacer predicciones de los datos. Alcanzan un desempeño parecido, pero en términos de precisión Random Forest alcanza una mayor precisión, mientras que la regresión logística tiene mejor F1-score, por lo que ambos modelos son mejores que el de RNA

- Mejoras posibles:

Podrían mejorarse los resultados si aumentamos el número de capas o el número de epoch, así como el número de instancias de entrenamiento (para disponer de más instancias cuya clase en la variable objetivo sean 3 y 8)

[ ]:

[ ]: