ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

CLASIFICACIÓN DE EMOCIONES UTILIZANDO DATASET DEAP

PROYECTO SEGUNDO PARCIAL DE LA MATERIA FUNDAMENTOS DE INTELIGENCIA ARTIFICIAL

NOMBRES Y APELLIDOS DE LOS ESTUDIANTES

Jhoana Aucancela
Jimy Calvo
Alberto Heredia
Jhon Torres

Ing. Mayra Alvarez

DMQ, Marzo 2023

ÍNDICE DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

1 DESCRIPCIÓN DE LO DESARROLLADO

En el presente proyecto se pretende utilizar la información proveniente de señales de un electroencefalograma (EEG) procesar dicha información proveniente de señales cerebrales registradas, con el fin de organizar e identificar diferentes emociones. Para lo cual se utilizará un conjunto de datos llamado DEAP, que consta de diferentes archivos de formato .dat. Centrándonos principal mente en analizar la clasificación de emociones presente en 5 de estos archivos utilizando diferentes técnicas para el procesamiento de señales y algoritmos de aprendizaje automático para identificar patrones en los datos y así poder clasificar las emociones con precisión.

1.1 Objetivo general

Extracción, análisis de un conjunto de datos de los archivos .dat asignados y la implementación de Algoritmos Supervisados

1.2 Objetivos específicos

1. Extraer los datos contenidos en los archivos .dat asignados, como parte del proceso de obtención de la información requerida.
2. Analizar los datos extraídos de los archivos asignados, con el propósito de obtener información relevante y significativa acerca de los mismos.
3. Implementar algoritmos supervisados.

1.3 Marco Teórico

La inteligencia artificial (IA) se puede definir como la capacidad de una máquina o un sistema informático para realizar tareas que normalmente requerirían inteligencia humana para realizarlas. Es una rama de la informática que se enfoca en el desarrollo de algoritmos y sistemas que puedan aprender y razonar como lo hace un ser humano [1].

Para el desarrollo del proyecto será necesario comprender los siguientes términos y métodos definidos a continuación:

DEAP dataset

Base de datos de registro de señales EGG de los participantes y las evaluaciones subjetivas de los sujetos [2].

Calificaciones de 120 extractos de un minuto de videos, para la identificación de las emociones y calificados por 14-16 participantes

Valencia

Identificador de emociones del ser humano, valor que puede identificar la alegría si el valor es alto, pero al ser bajo el valor identifica la ira o el miedo [3].

Arousal

Identificador de emociones del ser humano, valor que puede identificar la exitación intensa si el valor es alto, pero al ser bajo el valor identifica el sueño profundo [3].

Pickle

s un módulo en la librería estándar de Python que proporciona herramientas para serializar y deserializar objetos de Python, que se encuentra en los archivos [4].

Numpy

Biblioteca que permite realizar operaciones rápidas, manipulaciones, ordenamientos de matrices, entre muchas más funcionalidades como obtener la media. [5]

Pandas

Biblioteca que permite la manipulación de DataFrame, lectura y escritura de datos de archivos con formatos CSV, archivos de texto, Microsoft Excel, bases de datos SQL, y el rápido formato HDF5. Entre otras prestaciones [3].

Sklearn.preprocessing

Es un módulo en la librería de Python scikit-learn que proporciona herramientas para preprocesar y transformar datos antes de aplicar modelos de aprendizaje automático

OneHotEncoder

Para crear variables ficticias (dummy variables) a partir de variables categóricas [6].

Sklearn

Biblioteca que contine varios recursos de índole estadístico matemático, el cual es empleado para elabora modelos predictivos y análisis exploratorio de a datos, regresión lineal, regresión logística, categorización y agrupación de datos [6].

GaussianNB

Es un paquete de implantación para manejo del algoritmo Gaussian Naive Bayes para clasificar [7].

sklearn.svm

Estimador de aprendizaje, el cual contine herramienta para construir y evaluar modelos de clasificación y regresión basados SVM. Esta emplea criterio para seleccionar los mejores parámetros para el modelo [8].

sklearn.neighbors

Conjunto de herramientas y dependenicas empleados para predecir valores desconocidos de un conjunto de datos en base los valores de los vecinos más cercanos. Esta se emplear para tareas de clasificación y regresión [9].

KNeighborsClassifier

Dependencia que contiene algoritmos que se puede entrenar mediante la colocación de

etiquetas en los datos para luego usar estos datos para predecir la etiqueta del nuevo dato [10].

Keras

Es una biblioteca el cual contiene herramientas de trabajo de optimización, entrenamiento y evaluación modelos de redes neuronales. Esta emplea algoritmos de aprendizaje profundo, algoritmos que permite la abstracción de datos. Keras permite configurar procesos de aprendizaje como también apilar capas [11].

Sequential

Proporciona una forma sencilla de construir una red neuronal secuencial en Keras utilizando el método ‘add() [11].

Dense

Es una clase en la librería de Python keras.layers.core que se utiliza para crear capas densas en Keras [11].

Clasification report

Es una función en Python que se utiliza para generar un informe de clasificación que muestra diversas métricas de evaluación del rendimiento de un modelo de clasificación [6]. La precisión mide la proporción de verdaderos positivos (instancias correctamente clasificadas como positivas) respecto a todas las instancias que el modelo ha clasificado como positivas (verdaderos positivos y falsos positivos).

Tain test Split.

es una función en Python que se utiliza para dividir un conjunto de datos en conjuntos de entrenamiento y prueba para el entrenamiento y evaluación de modelos de aprendizaje automático [6]. La función train_test_split toma como entrada un conjunto de datos X y una variable objetivo y, y devuelve cuatro conjuntos de datos: X_train, X_test, y_train, y_test.

2 DESARROLLO

Para empezar con el desarrollo del presente proyecto, se implementarán las librerías mostradas en la **Fig. 1**, las mismas previamente descritas en el capítulo previo.

```
In [1]: import pickle
        import numpy as np
        import sklearn
        import pandas as pd
        from sklearn.preprocessing import OneHotEncoder
        import statistics
```

Fig. 1: *Importación de librerías.*

En la **Fig. 2** se muestra la forma en que se realiza la carga de datos de los archivos.dat implementando la librería pickle.

```
In [2]: def read_file(filename):
            x = pickle._Unpickler(open(filename, 'rb'))
            x.encoding = 'latin1'
            p = x.load()
            return p
```

Fig. 2: *Función para leer los archivos.*

Para la lectura de los archivos específicos se almacenará la numeración de estos en un array como se indica en la **Fig. 3**, adicionalmente en la misma figura se muestra la lectura de los archivos asignados.

```
In [3]: files = []
        for n in range(1, 5):
            s = ''
            if n < 10:
                s += '0'
            s += str(n)
            files. append(s)
        print(files)

        ['01', '02', '03', '04']

In [4]: labels = []
        data = []

        for i in files:
            filename = "s" + i + ".dat"
            trial = read_file(filename)
            labels.append(trial ['labels'])
            data.append(trial['data'])
```

Fig. 3: *Creación del array de archivos.*

La matriz contenida en los archivos .dat presenta una estructura de características o datos de tamaño 1600x40x8064 (videos, canales, características) y sus respectivas

etiquetas de tamaño 160x4 (video, etiquetas), de las cuales únicamente se utilizarán dos (valencia y arousal). En consecuencia, se requiere la adecuada gestión y almacenamiento de dicha matriz de datos como se muestra en la **Fig. 4**.

```
In [5]: # Re-shape arrays into desired shapes
        labels = np.array(labels)
        labels = labels.flatten()
        labels = labels.reshape(160, 4)

In [6]: data= np.array(data)
        data = data.flatten()
        data = data.reshape(160, 40, 8064)
        data.shape

Out[6]: (160, 40, 8064)
```

Fig. 4: *Labels y Data*

Con el objetivo de mejorar la comprensión de los datos extraídos de los archivos, se procederá a estructurarlos en un DataFrame que incluya las columnas correspondientes a los valores de Valencia y Arousal como se indica en la **Fig. 5**, en la misma figura se presenta la visualización del DataFrame generado, acompañado del resultado obtenido mediante la aplicación del método describe(). Dicho método arroja una tabla que proporciona información detallada sobre las estadísticas descriptivas de las columnas numéricas del DataFrame, tales como la media, la desviación estándar, los valores mínimo y máximo, y los cuartiles (25%, 50%, y 75%), para cada una de dichas columnas.

```
In [8]: # Extracción de Valencia and Arousal
        df_label_range = pd.DataFrame({'Valencia': labels[:,0], 'Arousal': labels[:,1]})
        print(df_label_range)
        print(df_label_range.describe())
```

```
     Valencia  Arousal
0        7.71     7.60
1        8.10     7.31
2        8.58     7.54
3        4.94     6.01
4        6.96     3.92
..        ...      ...
155      2.72     2.76
156      1.00     1.00
157      1.79     1.85
158      1.92     2.96
159      1.86     2.91

[160 rows x 2 columns]
         Valencia     Arousal
count  160.000000  160.000000
mean     5.157062    4.774438
std      2.429886    2.347240
min      1.000000    1.000000
25%      3.082500    2.942500
50%      5.000000    4.465000
75%      7.155000    7.010000
max      9.000000    9.000000
```

Fig. 5: *Dataframe de los datos extraídos.*

One hot encoder

El código fragmentado presentado en la **Fig. 6** consta de dos funciones que aplican una transformación en los datos, específicamente marcando con el valor '1' aquellos valores cuya valencia y arousal superen la media correspondiente.

```
In [9]: # Function, marca 1 a los valores mayores a la media de valencia
        def valencia_encoder(trial):
            media = np.mean(labels[:,0])
            if (labels[trial,0] >= media):
                return 1
            else:
                return 0
        # Function, marca 1 a los valores mayores a la media de arousal
        def arousal_encoder(trial):
            media = np.mean(labels[:,1])
            if (labels[trial,1] >= media):
                return 1
            else:
                return 0
```

Fig. 6: *Implementación de One hot encoder.*

En el código fragmentado expuesto en la **Fig. 7**, se realiza un proceso de normalización de las etiquetas y, posteriormente, se procede a ajustar la data normalizada en un DataFrame, en el cual se establecen dos columnas para los valores

de Valencia positiva y Arousal alta, respectivamente.

```
In [10]: # Normalización de las etiquetas
         labels_encoded = []
         for i in range (len(labels)):
           labels_encoded.append([valencia_encoder(i), arousal_encoder(i)])
         labels_encoded = np.reshape(labels_encoded, (160, 2))
         df_labels = pd.DataFrame(data=labels_encoded, columns=["Valencia Positiva", "Arousal Alta"])
         print(df_labels)

              Valencia Positiva  Arousal Alta
         0                    1             1
         1                    1             1
         2                    1             1
         3                    0             1
         4                    1             0
         ..                 ...           ...
         155                  0             0
         156                  0             0
         157                  0             0
         158                  0             0
         159                  0             0

         [160 rows x 2 columns]
```

Fig. 7: *Normalización de las etiquetas.*

32 canales de señales EEG

Después de generar las etiquetas correspondientes, se debe aplicar una selección de características a la matriz de datos, en la que solo se tomarán en cuenta 32 de los 160 canales disponibles. Como resultado, se obtendrá una matriz de datos con una dimensión de 160x32x8064 como se indica en la **Fig. 8**, donde la primera dimensión corresponde a la cantidad de canales seleccionados, la segunda dimensión representa la cantidad de características o atributos, y la tercera dimensión corresponde a la cantidad de muestras en el conjunto de datos.

32 canales de señales EEG

```
In [12]: eeg_channels = np.array(["Fp1", "AF3", "F3", "F7", "FC5", "FC1", "C3", "T7", "CP5", "CP1", "P3",

In [13]: eeg_data = []
         for i in range (len(data)):
           for j in range (len(eeg_channels)):
             eeg_data.append(data[i,j])
         eeg_data = np.reshape(eeg_data, (len(data), len(eeg_channels), len(data[0,0])))
         print(eeg_data.shape)

         (160, 32, 8064)
```

Fig. 8: *32 Canales de señales EEG*

Extracción de características

Una vez se tenga la matriz de datos y la matriz de etiquetas, se debe realizar un proceso de extracción de características. Para ello, se pueden utilizar medidas estadísticas como la media, la varianza, la mediana, la curtosis, la skewness, entre otras como se indica en la **Fig. 9**. Estas medidas permiten obtener información relevante de los datos y mejorar el rendimiento del modelo de clasificación. El objetivo de este proceso es reducir la

dimensionalidad de la matriz de datos, seleccionando solo las características más importantes y relevantes para la tarea de clasificación. Esto se logra a través de la aplicación de técnicas de selección de características y reducción de dimensionalidad, con el fin de mejorar la eficiencia y precisión del modelo de clasificación.

```
In [14]: print(eeg_data.size)

         41287680

In [15]: mean_data = np.mean(eeg_data, axis= 2)
         mean_data.shape

Out[15]: (160, 32)

In [16]: med_data = np.median(eeg_data, axis= 2)
         med_data.shape

Out[16]: (160, 32)
```

Fig. 9: *Medidas estadísticas.*

El código presentado en la **Fig. 10** incluye una función que modifica los valores almacenados en el array mean_data. Este array es el resultado de calcular la media aritmética de los datos en cuestión. En este caso, el array mean_data contiene la media de los datos y se utiliza como entrada para la función que realiza la modificación correspondiente.

```
In [17]: def cambiarValores(arr):
             for i in range(arr.shape[0]):
                 for j in range(arr.shape[1]):
                     if arr[i][j] > 0:
                         arr[i][j] = True
                     else:
                        arr[i][j] = False
             return arr
         cambiarValores(mean_data)

Out[17]: array([[0., 0., 0., ..., 1., 1., 0.],
                [0., 0., 0., ..., 1., 1., 1.],
                [1., 1., 1., ..., 0., 0., 0.],
                ...,
                [0., 0., 1., ..., 1., 1., 0.],
                [0., 0., 0., ..., 1., 1., 1.],
                [1., 1., 0., ..., 0., 0., 1.]])
```

Fig. 10: *Cambiar valores*

En la **Fig. 11** se muestra un fragmento de código que se encarga de estructurar los datos contenidos en el array mean_data en un DataFrame.

```
In [18]: meanDF = pd.DataFrame(mean_data)
         meanDF
```

Out[18]:

	0	1	2	3	4	5	6	7	8	9	...	22	23	24	25	26	27	28	29	30	31
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	...	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	...	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	...	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
155	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	...	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0
156	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	...	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0
157	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	...	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0
158	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0
159	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	...	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0

160 rows × 32 columns

Fig. 11: *DataFrame de la media aritmética de data*

Por otro lado, en la **Fig. 12** se presenta un fragmento de código que se utiliza para renombrar las columnas del DataFrame, lo que facilita el análisis posterior de los datos.

```
In [19]: meanDF = meanDF.rename(columns={0: 'Fp1', 1:"AF3", 2:"F3", 3:"F7", 4:"FC5", 5:"FC1", 6:"C3", 7:"T7",
         meanDF
```

Out[19]:

	Fp1	AF3	F3	F7	FC5	FC1	C3	T7	CP5	CP1	...	FC2	Cz	C4	T8	26	CP2	P4	P8	PO4	O2
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	...	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	...	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	...	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
155	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	...	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0
156	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	...	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0
157	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	...	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0
158	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0
159	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	...	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0

160 rows × 32 columns

Fig. 12: *Renombrar el nombre de las columnas del DataFrame meanDF.*

En el fragmento de código presentado en la **Fig. 13** se utiliza la función merge de la librería Pandas para combinar dos DataFrames en uno solo. El primer DataFrame corresponde a las etiquetas de clasificación, mientras que el segundo DataFrame corresponde al DataFrame de la media aritmética que se creó anteriormente. La unión de estos dos DataFrames se realiza utilizando una o varias columnas como llaves de unión.

Al combinar los dos DataFrames, se crea un nuevo DataFrame que contiene toda la información de ambos DataFrames. En este caso, el DataFrame resultante tendrá una columna

correspondiente a las etiquetas de clasificación y otra columna correspondiente a los valores de la media aritmética. La unión de estos dos DataFrames es un paso importante para el posterior análisis y modelado de los datos.

```
In [20]: data_EGG = pd.merge(labels_df,meanDF, left_index=True, right_index=True)
         data_EGG
```

Out[20]:

	Valencia Positiva	Arousal Alta	Fp1	AF3	F3	F7	FC5	FC1	C3	T7	...	FC2	Cz	C4	T8	26	CP2	P4	P8	PO4	O2
0	1	1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0
1	1	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0
2	1	1	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	...	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0	1	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
4	1	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
155	0	0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	...	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0
156	0	0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	...	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0
157	0	0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0
158	0	0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0
159	0	0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	...	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0

160 rows × 34 columns

Fig. 13: *Combinar DataFrames.*

División de datos

En la **Fig. 14** y **Fig. 15** se muestra el proceso de división de los datos en dos conjuntos, 'X' y 'y', que se utilizarán en la implementación de algoritmos supervisados de aprendizaje automático. El conjunto 'X' corresponde a los datos de entrada, que se utilizan para hacer predicciones, mientras que el conjunto 'y' corresponde a las etiquetas de clasificación, que se utilizan para evaluar la precisión de las predicciones.

```
In [21]: # division de la data
         X = data_EGG.iloc[:, 1:]
         X
```

Out[21]:

	Arousal Alta	Fp1	AF3	F3	F7	FC5	FC1	C3	T7	CP5	...	FC2	Cz	C4	T8	26	CP2	P4	P8	PO4	O2
0	1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0
1	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0
2	1	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	...	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
4	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	...	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
155	0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	...	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0
156	0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	...	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0
157	0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	...	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	0.0
158	0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0
159	0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	...	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0

160 rows × 33 columns

Fig. 14: *División de la data para X.*

```
In [23]: y = data_EGG['Valencia Positiva']
         y

Out[23]: 0      1
         1      1
         2      1
         3      0
         4      1
               ..
         155    0
         156    0
         157    0
         158    0
         159    0
         Name: Valencia Positiva, Length: 160, dtype: int32
```

Fig. 15: *División de la data para y.*

Algoritmos Supervisados

Para llevar a cabo la implementación de los algoritmos supervisados, se hará uso de las librerías que se muestran en la **Fig. 16**. Estas librerías fueron previamente descritas en el capítulo anterior y son ampliamente utilizadas en el análisis y modelado de datos en Python.

```
In [24]: # importar librerias
         from sklearn.model_selection import train_test_split
         from sklearn.naive_bayes import GaussianNB
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.linear_model import LogisticRegression
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.svm import SVC
         from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score
```

Fig. 16: *Librerías - Algoritmos supervisados.*

La **Fig. 17** muestra un fragmento de código que establece la división de la data en conjuntos de entrenamiento y prueba. Este proceso es una parte esencial del aprendizaje automático, ya que permite evaluar el rendimiento del modelo en datos no vistos durante el entrenamiento.

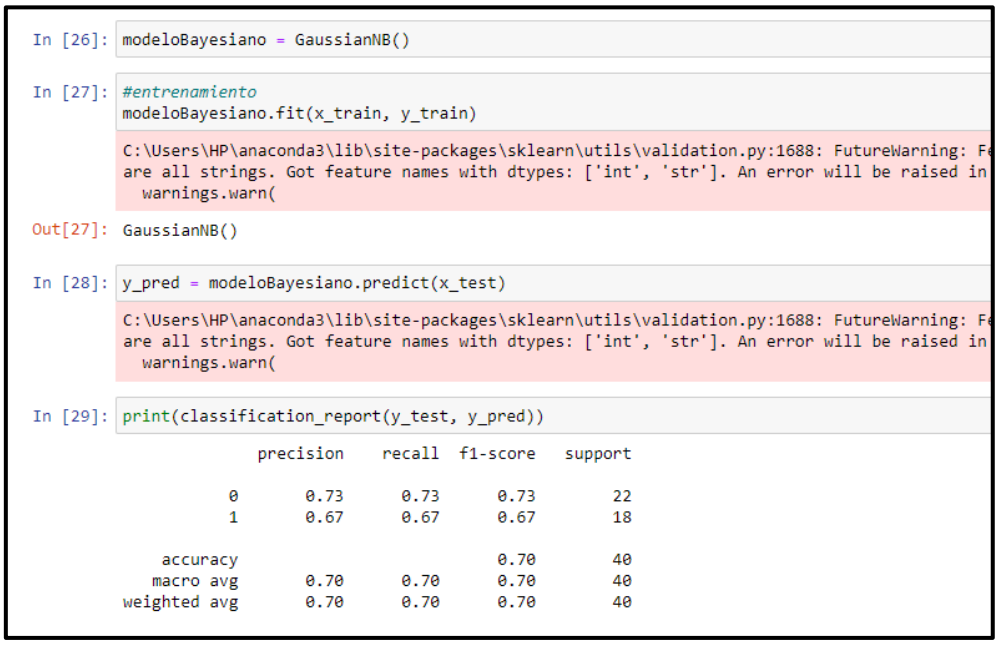
```
In [25]: # Asignamos en las variables x_train, y_train, los datos de entrada (x) y (y) respectivamente
         # x_test y y_test, designará el 30% de los datos (mejora muy poco con 25%)
         x_train, x_test, y_train, y_test = train_test_split(X,y, test_size=0.25, random_state=70)
```

Fig. 17: *División de la data de entrenamiento y prueba.*

En este caso, la función 'train_test_split' de la librería Scikit-learn se utiliza para dividir la data en dos conjuntos, uno para entrenamiento y otro para prueba. La función toma como entrada la data de entrada 'X' y las etiquetas 'y', y devuelve cuatro conjuntos de datos: 'X_train' y 'y_train', que corresponden a los datos de entrenamiento y las etiquetas correspondientes, y 'X_test' y 'y_test', que corresponden a los datos de prueba y las etiquetas correspondientes.

Bayesiano

La **Fig. 18** muestra un fragmento de código que implementa el algoritmo de clasificación bayesiana utilizando la librería Scikit-learn de Python. El algoritmo de clasificación bayesiana se basa en el teorema de Bayes y se utiliza para clasificar observaciones en diferentes categorías. Este algoritmo asume que las características de las observaciones son independientes entre sí y que su distribución es conocida.



```
In [26]: modeloBayesiano = GaussianNB()

In [27]: #entrenamiento
         modeloBayesiano.fit(x_train, y_train)

         C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\validation.py:1688: FutureWarning: Fe
         are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in
           warnings.warn(

Out[27]: GaussianNB()

In [28]: y_pred = modeloBayesiano.predict(x_test)

         C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\validation.py:1688: FutureWarning: Fe
         are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in
           warnings.warn(

In [29]: print(classification_report(y_test, y_pred))

                       precision    recall  f1-score   support

                    0       0.73      0.73      0.73        22
                    1       0.67      0.67      0.67        18

             accuracy                           0.70        40
            macro avg       0.70      0.70      0.70        40
         weighted avg       0.70      0.70      0.70        40
```

Fig. 18: *Algoritmo Bayesiano.*

Al implementar este algoritmo obtenemos como resultados:

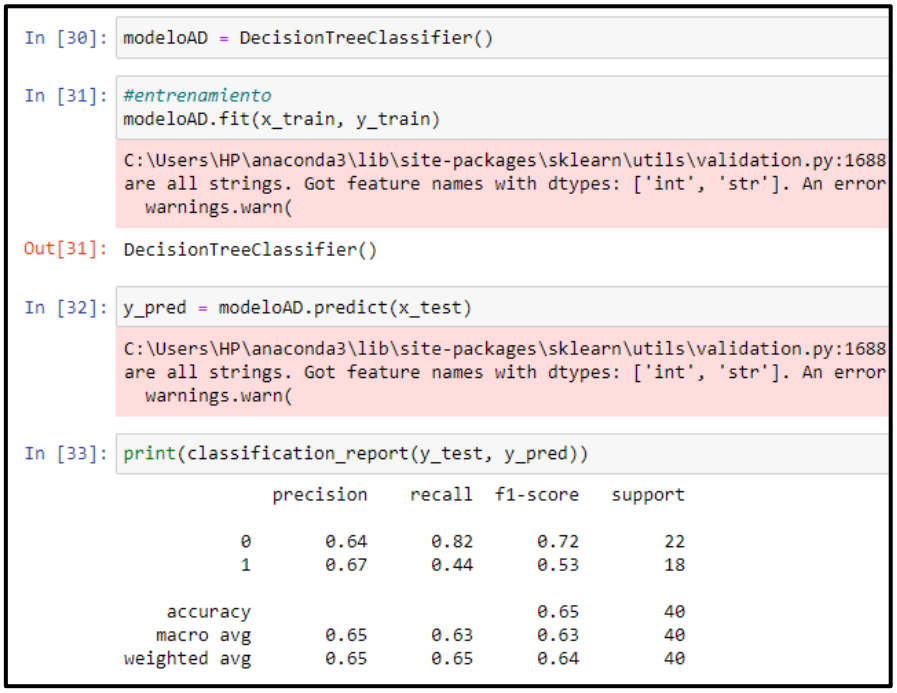
- La precisión (precisión) indica la proporción de observaciones clasificadas como positivas que realmente son positivas. En este caso, la precisión para la clase 0 es del 73%, lo que significa que el 73% de las observaciones clasificadas como clase 0 realmente son de esa clase. La precisión para la clase 1 es del 67%, lo que significa que el 67% de las observaciones clasificadas como clase 1 realmente son de esa clase.
- El recall para la clase 0 es del 73%, lo que significa que el 73% de las observaciones de la clase 0 fueron identificadas correctamente por el modelo. El recall para la clase 1 es del 67%, lo que significa que el 67% de las observaciones de la clase 1 fueron identificadas correctamente por el modelo.

En general, estos resultados indican que el modelo tiene una precisión y recall moderados para ambas clases, y que el f1-score también es moderado. Además, la precisión macro y ponderada son iguales, lo que sugiere que las clases están balanceadas en el conjunto de datos.

Árbol de Decisión

El fragmento de código presentado en la **Fig. 19** es una implementación del algoritmo Árbol de

Decisión en Python. El algoritmo funciona dividiendo repetidamente el conjunto de datos en subconjuntos más pequeños, basados en los valores de las características, hasta que se logra la máxima homogeneidad dentro de cada subconjunto en términos de la variable objetivo.



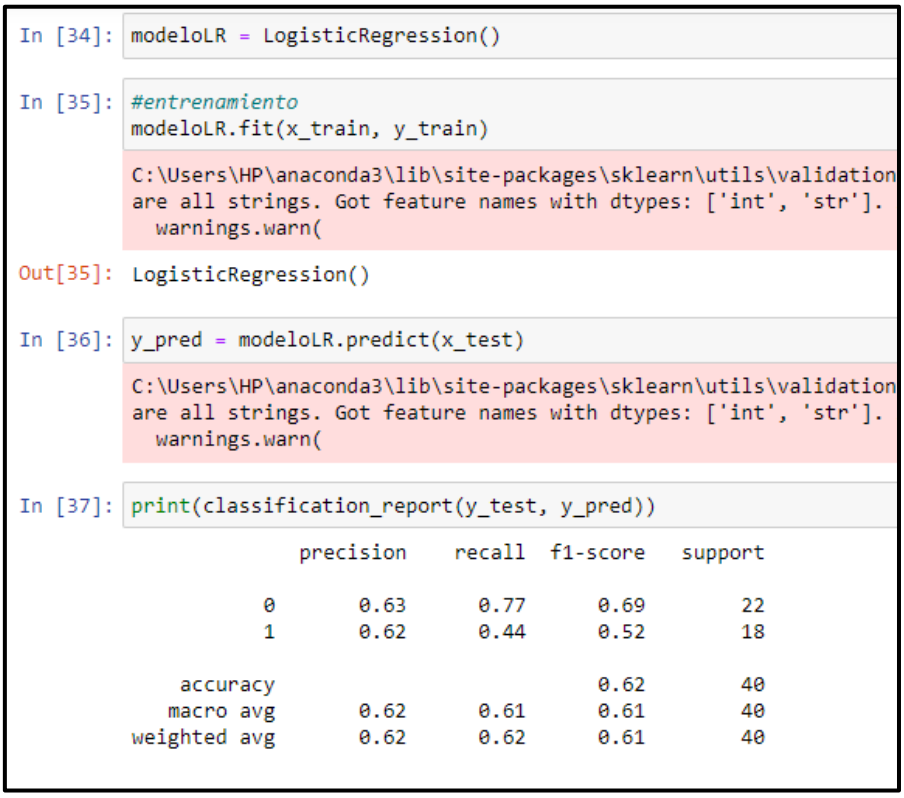
```
In [30]: modeloAD = DecisionTreeClassifier()

In [31]: #entrenamiento
         modeloAD.fit(x_train, y_train)

         C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\validation.py:1688
         are all strings. Got feature names with dtypes: ['int', 'str']. An error
           warnings.warn(

Out[31]: DecisionTreeClassifier()

In [32]: y_pred = modeloAD.predict(x_test)

         C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\validation.py:1688
         are all strings. Got feature names with dtypes: ['int', 'str']. An error
           warnings.warn(

In [33]: print(classification_report(y_test, y_pred))

                       precision    recall  f1-score   support

                    0       0.64      0.82      0.72        22
                    1       0.67      0.44      0.53        18

             accuracy                           0.65        40
            macro avg       0.65      0.63      0.63        40
         weighted avg       0.65      0.65      0.64        40
```

Fig. 19: *Algoritmo árbol de decisión.*

El informe de clasificación muestra los resultados de la evaluación del modelo de Árbol de Decisión en el conjunto de prueba. La precisión promedio para la detección de las dos clases (0 y 1) fue del 65%, lo que significa que el modelo clasificó correctamente el 65% de las instancias en el conjunto de prueba. La precisión de la clase 0 (0.64) indica que el modelo pudo identificar correctamente el 64% de los casos en los que la etiqueta real era 0, mientras que la precisión de la clase 1 (0.67) indica que el modelo pudo identificar correctamente el 67% de los casos en los que la etiqueta real era 1. El valor de recall de la clase 0 (0.82) indica que el modelo identificó correctamente el 82% de los casos reales de la clase 0, mientras que el recall de la clase 1 (0.44) indica que el modelo identificó correctamente el 44% de los casos reales de la clase 1. El F1-score promedio fue de 0.63, lo que indica un equilibrio entre precisión y recall.

Regresión logística

En el código presentado en la **Fig. 20** se evidencia la implementación del algoritmo de Regresión Logística.



```
In [34]: modeloLR = LogisticRegression()

In [35]: #entrenamiento
         modeloLR.fit(x_train, y_train)

         C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\validation
         are all strings. Got feature names with dtypes: ['int', 'str'].
           warnings.warn(

Out[35]: LogisticRegression()

In [36]: y_pred = modeloLR.predict(x_test)

         C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\validation
         are all strings. Got feature names with dtypes: ['int', 'str'].
           warnings.warn(

In [37]: print(classification_report(y_test, y_pred))

                       precision    recall  f1-score   support

                    0       0.63      0.77      0.69        22
                    1       0.62      0.44      0.52        18

             accuracy                           0.62        40
            macro avg       0.62      0.61      0.61        40
         weighted avg       0.62      0.62      0.61        40
```

Fig. 20: *Algoritmo Regresión logística*

La tabla muestra la precisión, recall y f1-score obtenidos por el algoritmo Regresión logística en la clasificación de los datos de prueba. La precisión para la clase 0 es del 63%, lo que significa que de todas las muestras clasificadas como clase 0, el 63% eran realmente de esa clase. Para la clase 1, la precisión es del 62%, lo que indica que de todas las muestras clasificadas como clase 1, el 62% eran realmente de esa clase. El recall para la clase 0 es del 77%, lo que significa que de todas las muestras que eran realmente de la clase 0, el 77% fueron correctamente identificadas. Para la clase 1, el recall es del 44%, lo que indica que solo el 44% de las muestras que eran realmente de la clase 1 fueron correctamente identificadas. El f1-score es una medida de la precisión general del modelo, y en este caso es del 69% para la clase 0 y del 52% para la clase 1. La exactitud (accuracy) general del modelo es del 62%.

KNN

En el fragmento de código mostrado en la **Fig. 21** se implementa el algoritmo K-Vecinos más Cercanos (KNN, por sus siglas en inglés) utilizando la librería scikit-learn de Python. Este algoritmo es un método de clasificación supervisada que busca predecir la clase de un nuevo punto de datos basado en las clases de los K puntos de entrenamiento más cercanos a él en el espacio de características.

```
In [38]: # definir valor de K
         n_vecinos = 18 #compara con 18 vecinos

         modeloKNN = KNeighborsClassifier(n_vecinos)

In [39]: #entrenamiento
         modeloKNN.fit(x_train, y_train)

         C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\validation.py:1688: Fut
         are all strings. Got feature names with dtypes: ['int', 'str']. An error will
           warnings.warn(

Out[39]: KNeighborsClassifier(n_neighbors=18)

In [40]: y_pred = modeloKNN.predict(x_test)

         C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\validation.py:1688: Fut
         are all strings. Got feature names with dtypes: ['int', 'str']. An error will
           warnings.warn(

In [41]: print(classification_report(y_test, y_pred))

                       precision    recall  f1-score   support

                    0       0.66      0.86      0.75        22
                    1       0.73      0.44      0.55        18

             accuracy                           0.68        40
            macro avg       0.69      0.65      0.65        40
         weighted avg       0.69      0.68      0.66        40
```

Fig. 21: *Algoritmo KNN*

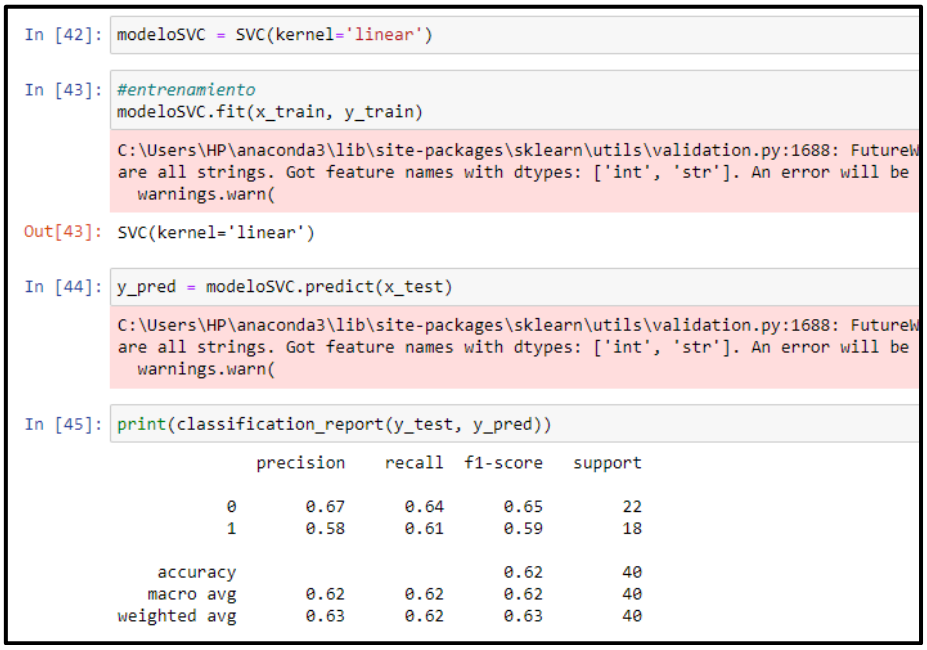
En este código, se establece el número de vecinos K en 18, se entrena el modelo utilizando los datos de entrenamiento y se realiza la predicción de las clases para los datos de prueba.

En este código, se establece el número de vecinos K en 18, se entrena el modelo utilizando los datos de entrenamiento y se realiza la predicción de las clases para los datos de prueba.

La tabla muestra la precisión, recall y f1-score obtenidos por el algoritmo KNN en la clasificación de los datos de prueba. La precisión para la clase 0 es del 66%, lo que significa que de todas las muestras clasificadas como clase 0, el 66% eran realmente de esa clase. Para la clase 1, la precisión es del 73%, lo que indica que de todas las muestras clasificadas como clase 1, el 73% eran realmente de esa clase. El recall para la clase 0 es del 86%, lo que significa que de todas las muestras que eran realmente de la clase 0, el 86% fueron correctamente identificadas. Para la clase 1, el recall es del 44%, lo que indica que solo el 44% de las muestras que eran realmente de la clase 1 fueron correctamente identificadas. El f1-score es una medida de la precisión general del modelo, y en este caso es del 75% para la clase 0 y del 55% para la clase 1. La exactitud (accuracy) general del modelo es del 68%.

SVC

En el código presentado en la **Fig. 22** se demuestra la implementación del algoritmo de Máquinas de Soporte Vectorial (SVC, por sus siglas en inglés).



```
In [42]: modeloSVC = SVC(kernel='linear')

In [43]: #entrenamiento
         modeloSVC.fit(x_train, y_train)

         C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\validation.py:1688: FutureW
         are all strings. Got feature names with dtypes: ['int', 'str']. An error will be
           warnings.warn(

Out[43]: SVC(kernel='linear')

In [44]: y_pred = modeloSVC.predict(x_test)

         C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\validation.py:1688: FutureW
         are all strings. Got feature names with dtypes: ['int', 'str']. An error will be
           warnings.warn(

In [45]: print(classification_report(y_test, y_pred))

                       precision    recall  f1-score   support

                    0       0.67      0.64      0.65        22
                    1       0.58      0.61      0.59        18

             accuracy                           0.62        40
            macro avg       0.62      0.62      0.62        40
         weighted avg       0.63      0.62      0.63        40
```

Fig. 22: *Algoritmo SVC*

El informe de métricas de evaluación muestra que el modelo SVC logra una precisión del 67% para la clase 0 y del 58% para la clase 1, lo que significa que el modelo logra identificar correctamente el 67% y el 58% de las instancias positivas (clase 0 y 1, respectivamente). Por otro lado, el recall del modelo es del 64% para la clase 0 y del 61% para la clase 1, lo que indica que el modelo logra recuperar el 64% y el 61% de las instancias positivas (clase 0 y 1, respectivamente). La medida f1-score es una media armónica de precision y recall, por lo que esta métrica es adecuada para evaluar el equilibrio entre ambas medidas. El f1-score del modelo es del 65% para la clase 0 y del 59% para la clase 1.

La precisión global del modelo, medida a través del accuracy, es del 62%, lo que indica que el modelo logra clasificar correctamente el 62% de las instancias en el conjunto de datos utilizado.

Redes Neuronales

En el fragmento de código que se presenta en la **Fig. 23** se puede observar una implementación del algoritmo de Redes Neuronales, una técnica de aprendizaje automático que se basa en la simulación de la estructura y el funcionamiento de las redes neuronales biológicas.

```
In [62]: # importar librerias
         from keras.models import Sequential
         from keras.layers.core import Dense

In [85]: modeloRN = Sequential()

In [86]: """
         1. Agregamos las capas Dense con “model.add()”.
         2. En input_dim=33 estamos definiendo la capa de entrada con 33 neuronas (entradas) y
            20 número de neuronas.
         Como función de activación “relu”.
         3. Agregamos una capa con 1 neurona de salida y función de activación relu.
         """
         modeloRN.add(Dense(80, input_dim=33, activation='relu'))
         modeloRN.add(Dense(1, activation='relu'))

In [87]: # parámetros  de la red
         modeloRN.compile(loss='mean_squared_error',
                          optimizer='adam',
                          metrics=['binary_accuracy'])
```

Fig. 23: *Algoritmo Redes neuronales.*

Al evaluar el algoritmo se obtiene como resultados:

- El valor de "loss" es 0.0510, lo que indica que el modelo tiene un bajo error en la predicción de los datos de prueba.
- El valor de "binary_accuracy" es 0.9417, lo que indica que el modelo ha logrado una alta precisión en la clasificación de los datos de prueba en una tarea binaria.

En general, se podría concluir que el modelo de Redes Neuronales ha tenido un buen rendimiento en la tarea de clasificación binaria con los datos de prueba proporcionados, lo que sugiere que podría tener una buena capacidad para generalizar y clasificar correctamente nuevos datos en una tarea similar.

```
In [88]: # entrenamiento del modelo (datos de entrada y salida)
         modeloRN.fit(x_train, y_train, epochs=100)

         Epoch 1/100
         4/4 [==============================] - 0s 3ms/step - loss: 0.3736 - binary_accuracy: 0.4583
         Epoch 2/100
         4/4 [==============================] - 0s 3ms/step - loss: 0.3133 - binary_accuracy: 0.4833
         Epoch 3/100
         4/4 [==============================] - 0s 3ms/step - loss: 0.2680 - binary_accuracy: 0.5667
         Epoch 4/100
         4/4 [==============================] - 0s 3ms/step - loss: 0.2578 - binary_accuracy: 0.5833
         Epoch 5/100
         4/4 [==============================] - 0s 2ms/step - loss: 0.2500 - binary_accuracy: 0.5917
         Epoch 6/100
         4/4 [==============================] - 0s 3ms/step - loss: 0.2369 - binary_accuracy: 0.6417
         Epoch 7/100
         4/4 [==============================] - 0s 3ms/step - loss: 0.2271 - binary_accuracy: 0.7000
         Epoch 8/100
         4/4 [==============================] - 0s 3ms/step - loss: 0.2179 - binary_accuracy: 0.7167
         Epoch 9/100
         4/4 [==============================] - 0s 3ms/step - loss: 0.2075 - binary_accuracy: 0.6750
         Epoch 10/100

In [89]: # evaluación del modelo
         scores = modeloRN.evaluate(x_train, y_train)

         4/4 [==============================] - 0s 3ms/step - loss: 0.0510 - binary_accuracy: 0.9417
```

Fig. 24: *Evaluación del algoritmo redes neuronales.*

3 CONCLUSIONES

- La carga de archivos en este proyecto es fundamental para poder acceder a la información contenida en la base de datos DEAP y utilizarla para entrenar el modelo de aprendizaje automático ya que contiene las señales EEG y etiquetas necesarias para clasificar las emociones. Para cargar correctamente estos archivos, se deben aplicar técnicas adecuadas de lectura y procesamiento de datos en Python, lo que permitirá preparar los datos para el entrenamiento del modelo.

- La carga y procesamiento adecuado de los archivos es una parte crítica de este proyecto, ya que es la base para lograr el objetivo final de desarrollar un modelo preciso y efectivo para clasificar emociones.

- La clasificación de emociones mediante el electroencefalograma es un proceso que implica varias tareas, desde la obtención de las señales EEG hasta la validación del modelo clasificador, por tal motivo a han usado diferentes algoritmos de aprendizaje automático que pueden ser utilizados para clasificar las emociones.

- El uso de algoritmos de aprendizaje supervisado en la clasificación de emociones es importancia en campos como la salud mental y la selección adecuada de algoritmos, técnicas de extracción de características es importante para la clasificación.

- La evaluación del modelo SVC muestra que tiene un desempeño aceptable en la clasificación de emociones. Sin embargo, aún hay margen de mejora para aumentar la precisión y el recall del modelo en ambas clases.

4 RECOMENDACIONES

- Es importante asegurarse de tener un buen entendimiento de los formatos de archivos y estructuras de datos utilizados en la base de datos DEAP y en los diferentes archivos .dat.

- Es recomendable revisar la documentación disponible para entender la información contenida en los archivos y la forma en que se deben procesar. Además, es importante aplicar técnicas adecuadas de procesamiento de señales y algoritmos de aprendizaje automático para lograr una clasificación precisa de las emociones.

- Es importante destacar que el preprocesamiento de datos es una tarea crítica en el proceso de clasificación de emociones, ya que puede afectar significativamente el rendimiento del modelo. Por lo tanto, se recomienda prestar especial atención a esta tarea y realizar un análisis exhaustivo de los datos antes de comenzar con la extracción de características y el entrenamiento del modelo.

- Se recomienda investigar y utilizar las técnicas más avanzadas en la materia, lo que permitirá lograr un mejor rendimiento en el modelo final, asegurándose que el modelo resultante sea coherente.

- Es importante enfocarse en la evaluación y exploración de diferentes modelos. Además, es necesario implementar las nuevas metodologías y enfoques en la clasificación de emociones para mejorar la precisión de los modelos.

5 REFERENCIAS BIBLIOGRÁFICAS

[1] Google Cloud, «Google Cloud,» [En línea]. Available: https://cloud.google.com/learn/what-is-artificial-intelligence?hl=es-419#:~:text=La%20inteligencia%20artificial%20(IA)%20es,hacer%20recomendaciones%20y%20mucho%20m%C3%A1s.. [Último acceso: 05 03 2023].

[2] EECS, «EECS,» [En línea]. Available: https://www.eecs.qmul.ac.uk/mmv/datasets/deap/readme.html. [Último acceso: 04 03 2023].

[3] J. J. C. Moratilla, «Riunet,» Análisis de emociones de sujetos a partir de señales de EGG, [En línea]. Available: bit.ly/3KSGWBX. [Último acceso: 04 03 2023].

[4] Python, «Python,» 04 03 2023. [En línea]. Available: https://docs.python.org/3/library/pickle.html. [Último acceso: 04 03 2023].

[5] N. Developers, «NumPy,» [En línea]. Available: https://numpy.org/doc/stable/user/whatisnumpy.html. [Último acceso: 04 03 2023].

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas y A. Passos, «Scikit-learn,» 2012. [En línea].

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas y A. Passos, «scikit-learn.org,» [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas y A. Passos, «sklearn.naive_bayes.GaussianNB,» [En línea]. Available: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.svm.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas y A. Passos, «KNeighborsClassifier,» [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, J. Dubourg, A. Passos, D. Cournapeau, M. Brucher y É. Duchesnay, «scikit-learn.org -KNeighborsClassifier,» [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[11] K. Developers, «Keras,» 14 11 2022. [En línea]. Available: https://keras.io/getting_started/. [Último acceso: 04 03 2023].

6 ANEXOS

ANEXO I. Archivo .ipynb utilizado

Repositorio