Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA EN TECNOLOGÍA Y SERVICIOS DE TELECOMUNICACIÓN

ÁREA DE INGENIERÍA TELEMÁTICA

Detección del trastorno del espectro autista en niños mediante técnicas de aprendizaje automático

ANEXOS

D. Ordoñez Prendes Ángel

TUTOR: D. Corcoba Magaña Victor

FECHA: Julio de 2024

Índice de Contenido

A) Código del primer estudio

Código correspondiente al estudio “Técnicas de 'Deep Learning' sobre métodos analíticos”

```
# -*- coding: utf-8 -*-
"""TFG_et2_Comparativa

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/14R2YZ-hd28j94bkOj4Xhi2aggmJulbJ0

# Instalación e Importación de Librerías
"""

import tensorflow as tf
import tensorflow_decision_forests as tfdf

import pandas as pd
import numpy as np
import csv

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import PrecisionRecallDisplay

import matplotlib.pyplot as plt
import seaborn

from sklearn.model_selection import KFold
import time
import psutil
import os
import tensorflow_decision_forests as tfdf

print(tf.__version__)
print(tfdf.__version__)
```

```
27      """# Funciones de creación y entrenamiento de modelos

28      Modelos Secuenciales (En función del optimizador)
29      """

30      def sequential_model():
31          model = tf.keras.models.Sequential()
32          model.add(tf.keras.layers.Dense(units=15,
                activation='sigmoid',input_shape=(15,)))
33          model.add(tf.keras.layers.Dense(units=10, activation='sigmoid'))
34          model.add(tf.keras.layers.Dropout(0.1))
35          model.add(tf.keras.layers.Dense(units=10, activation='relu'))
36          model.add(tf.keras.layers.Dense(units=2, activation='softmax'))

37          return model

38      def sequential_model_fit_by_optimizer(optimizer_type):

39          #Creación del Modelo

40          model = sequential_model()

41          #Compilación y Entrenamiento

42          model.compile(optimizer=optimizer_type,
                 loss='sparse_categorical_crossentropy',
                metrics=['sparse_categorical_accuracy'])
43              start_time = time.time()

44          model_fit = model.fit(dataset_train_data, dataset_train_label,
                epochs=30, validation_data=(dataset_validation_data ,
                dataset_validation_label),verbose=0)

45          training_time = time.time() - start_time

46          #Evaluación
47          Test_loss, Test_accuracy = model.evaluate(dataset_test_data,
                dataset_test_label)
48              dataset_test_predictions =
                np.argmax(model.predict(dataset_test_data), axis=-1)

49          return model_fit , dataset_test_predictions , Test_loss,
                Test_accuracy , training_time
```

```
50          """Funciones Secuenciales con Validación Cruzada"""

51          def sequential_model_fit_by_optimizer_kfold(optimizer_type):
52              kfolder = KFold(n_splits=10, shuffle=True)
53              folder_number =  1

54              acc_per_fold = []
55              loss_per_fold = []

56              data = np.concatenate((dataset_train_data, dataset_test_data),
                axis=0)
57              label = np.concatenate((dataset_train_label, dataset_test_label),
                axis=0)

58              #Iteración por folder

59              for train, test in kfolder.split(data,label):

60                  model = sequential_model()

61                  model.compile(optimizer=optimizer_type,
                    loss='sparse_categorical_crossentropy',
                    metrics=['sparse_categorical_accuracy'])

62                  print('------------------------------------------------------------------------')
63                  print(f'-> Training for folder number {folder_number}')
64                  model_fit = model.fit(data[train], label[train],
                            epochs=30,verbose=0)

65                  values = model.evaluate(data[test], label[test])
66                  print(f'--> Score for folder {folder_number}:
                    {model.metrics_names[0]}
                    of {values[0]}; {model.metrics_names[1]} of {values[1]*100}%')

67                  acc_per_fold.append(values[1] * 100)
68                  loss_per_fold.append(values[0])

69                  folder_number = folder_number + 1

70          # Resumen por folder

71              print('------------------------------------------------------------------------')
72              print('-> Score per fold')
```

```
73              for i in range(0, len(acc_per_fold)):
74                    print('------------------------------------------------------------------------')
75                    print(f'--> Fold {i+1} - Loss: {loss_per_fold[i]} - Accuracy:
                      {acc_per_fold[i]}%')

76              print('------------------------------------------------------------------------')
77              print('-> Average scores for all folds:')
78              print(f'--> Accuracy: {np.mean(acc_per_fold)} (+-
                  {np.std(acc_per_fold)})')

79              print(f'--> Loss: {np.mean(loss_per_fold)}')
80              print('------------------------------------------------------------------------\n')
81            return np.mean(acc_per_fold) , np.mean(loss_per_fold)

82        """Modelos de Árboles de Decisión"""

83        def Decision_Tree_model_fit(model_type):

84            #Creación del Modelo

85            if model_type=="GB":
86                model = tfdf.keras.GradientBoostedTreesModel()
87            else:
88                model = tfdf.keras.RandomForestModel()

89            #Compilación y Entrenamiento
90            model.compile(metrics=['accuracy'])
91            start_time = time.time()

92            #import gc
93            model.fit(TF_train, validation_data=(TF_validation))

94            training_time = time.time() - start_time

95            #Evaluación
96            Test_loss, Test_accuracy = model.evaluate(TF_test)
97            dataset_test_predictions_RF = model.predict(TF_test)
98
99            dataset_test_predictions_RF =
                  np.round(dataset_test_predictions_RF)

100           return model , dataset_test_predictions_RF , Test_loss,
                Test_accuracy , training_time
```

```
"""Modelos de Árboles de Decisión con Validación Cruzada"""

def Decision_Tree_model_fit_kfold(model_type):
    kfolder = KFold(n_splits=10, shuffle=True)
    folder_number =  1

    acc_per_fold = []
    loss_per_fold = []

    data = np.concatenate((dataset_train_data, dataset_test_data),
       axis=0)
    label = np.concatenate((dataset_train_label, dataset_test_label),
       axis=0)

    #Iteración por folder

    for train, test in kfolder.split(data,label):
        if model_type=="GB":
            model = tfdf.keras.GradientBoostedTreesModel()
        else:
            model = tfdf.keras.RandomForestModel()
        model.compile(metrics=['accuracy'])

        print('------------------------------------------------------------------------')
        print(f'-> Training for folder number {folder_number}')
        model_fit = model.fit(data[train], label[train],verbose=0)

        values = model.evaluate(data[test], label[test])
        print(f'--> Score for folder {folder_number}:
       {model.metrics_names[0]}
       of {values[0]}; {model.metrics_names[1]} of {values[1]*100}%')

        acc_per_fold.append(values[1] * 100)
        loss_per_fold.append(values[0])

        folder_number = folder_number + 1

     #Resumen por folder

    print('------------------------------------------------------------------------')
    print('-> Score per fold')

    for i in range(0, len(acc_per_fold)):
        print('------------------------------------------------------------------------')
        print(f'--> Fold {i+1} - Loss: {loss_per_fold[i]} - Accuracy:
       {acc_per_fold[i]}%')
        print('------------------------------------------------------------------------')
```

```
131                 print('-> Average scores for all folds:')
132                 print(f'--> Accuracy: {np.mean(acc_per_fold)} (+-
                        {np.std(acc_per_fold)})')
133                 print(f'--> Loss: {np.mean(loss_per_fold)}')
134                 print('------------------------------------------------------------------------\n')

135             return np.mean(acc_per_fold) , np.mean(loss_per_fold)
136         """# Funciones de Creación de Gráficos"""

137         def generate_conf_matrix(test_label,test_predictions,title):
138
            conf_matrx=metrics.confusion_matrix(test_label,test_predictions)
139             labels=np.unique(dataset_test_label)
140             plt.figure(figsize=(6,4))
141             seaborn.heatmap(conf_matrx,annot
                    =True,fmt='d',cmap='Blues',xticklabels=
                    labels,yticklabels=labels)
142             plt.xlabel('Predictions')
143             plt.ylabel('True Values')
144             plt.title(title)
145             plt.show()

146         def generate_accuracy_loss_graphs(model,title):
147             plt.figure(figsize=(12, 4))
148             plt.subplot(1, 2, 1)

149             plt.plot(model.history['sparse_categorical_accuracy'],
                        label='Accuracy (Training)')
150             plt.plot(model.history['val_sparse_categorical_accuracy'],
                        label='Accuracy (Validation)')
151             plt.xlabel('Epoch')
152             plt.ylabel('Precisión')
153             plt.title(title+' Accuracy Evolution')
154             plt.legend()
155             plt.grid(True)

156             plt.subplot(1, 2, 2)

157             plt.plot(model.history['loss'], label='Loss (Training)')
158             plt.plot(model.history['val_loss'], label='Loss (Validation)')
159             plt.xlabel('Epoch')
160             plt.ylabel('Loss')
161             plt.title(title+' Loss Evolution')
162             plt.legend()

163             plt.tight_layout()
164             plt.grid(True)
```

```
165             plt.show()
166         def generate_accuracy_graphs(model,title):
167             plt.figure(figsize=(12, 4))
168             plt.plot(model.history['sparse_categorical_accuracy'],
                label='Accuracy (Training)')

169             plt.plot(model.history['val_sparse_categorical_accuracy'],
                  label='Accuracy (Validation)')
170             plt.xlabel('Epoch')
171             plt.ylabel('Precisión')
172             plt.title(title+' Accuracy Evolution')
173             plt.legend()

174             plt.tight_layout()
175             plt.grid(True)
176             plt.show()

177         def generate_loss_graphs(model,title):
178             plt.figure(figsize=(12, 4))

179             plt.plot(model.history['loss'], label='Loss (Training)')
180             plt.plot(model.history['val_loss'], label='Loss (Validation)')
181             plt.xlabel('Epoch')
182             plt.ylabel('Loss')
183             plt.title(title+'Loss Evolution')
184             plt.legend()

185             plt.tight_layout()
186             plt.grid(True)
187             plt.show()

188         def Decision_Tree_generate_accuracy_loss_graphs(model):
189             logs = model.make_inspector().training_logs()
190             plt.figure(figsize=(12, 6))
191             plt.subplot(1, 2, 1)
192             plt.plot([log.num_trees for log in logs], [log.evaluation.accuracy for
                    log in logs])
193             plt.xlabel("Number of trees")
194             plt.ylabel("Accuracy (out-of-bag)")
195             plt.grid(True)
196             plt.subplot(1, 2, 2)
197             plt.plot([log.num_trees for log in logs], [log.evaluation.loss
                for log in logs])
198             plt.xlabel("Number of trees")
199             plt.ylabel("Logloss (out-of-bag)")
```

```
200             plt.grid(True)
201             plt.show()

202         """---

203         # Carga y Preprocesado del Dataset

204         Leer el DataSet desde un archivo .csv

205         IMPORTANTE: Cambios en los valores del DataSet en las variables
            formato string

            Autism: Yes - 1 // No - 0
            Gender: f - 1 // m - 0
            Ethnicity:

206         'Middle eastern' - 1 'White European' - 2 'Hispanic' - 3 'Black' - 4
            'Asian'
            - 5 'South asian' - 6 'Native Indian' - 7 'Others' - 8 'Latino' - 9 'Mixed'
            -
              10 'Pacifica' - 11

            Family Member with ASD:

207         'Family Member' - 0 'Health Care Professional' - 1 'Self' - 2

208         Se elimina la columna de QChat-10-Score pq causa over-feating en
              Random Forest

209         ---

210         **Pre Procesado de DataSet**
211         """

212         #Montar en Google Drive (SOLO para google colab)

213         #from google.colab import drive
214         #drive.mount('/content/drive')
215         #dataset = pd.read_csv("drive/MyDrive/TFG/Toddler Autism dataset
            July 2018_Text.csv").drop(columns=["Case_No","Qchat-10-
             Score","Who completed the test"])

216         dataset = pd.read_csv("Toddler Autism dataset July
             2018_Text.csv").drop(columns=["Case_No","Qchat-10-
            Score","Who  completed the test"]) #DS Ordiginal
217         dataset.head()
```

```
218     for counter, element in dataset.iterrows():

219         if element["Class/ASD_Traits"] == "Yes":
220             dataset.iloc[counter,15] = 1
221         elif element["Class/ASD_Traits"] == "No":
222             dataset.iloc[counter,15] = 0

223         if element["Sex"] == "f":
224             dataset.iloc[counter,11] = 1
225         elif element["Sex"] == "m":
226             dataset.iloc[counter,11] = 0

227         if element["Family_mem_with_ASD"] == "yes":
228             dataset.iloc[counter,14] = 1
229         elif element["Family_mem_with_ASD"] == "no":
230             dataset.iloc[counter,14] = 0

231         if element["Jaundice"] == "yes":
232             dataset.iloc[counter,13] = 1
233         elif element["Jaundice"] == "no":
234             dataset.iloc[counter,13] = 0

235         if element["Ethnicity"] == "White European":
236             dataset.iloc[counter,12] = 1
237         elif element["Ethnicity"] == "Hispanic":
238             dataset.iloc[counter,12] = 2
239         elif element["Ethnicity"] == "black":
240             dataset.iloc[counter,12] = 3
241         elif element["Ethnicity"] == "asian":
242             dataset.iloc[counter,12] = 4
243         elif element["Ethnicity"] == "south asian":
244             dataset.iloc[counter,12] = 5
245         elif element["Ethnicity"] == "Native Indian":
246             dataset.iloc[counter,12] = 6
247         elif element["Ethnicity"] == "Others":
248             dataset.iloc[counter,12] = 7
249         elif element["Ethnicity"] == "Latino":
250             dataset.iloc[counter,12] = 8
251         elif element["Ethnicity"] == "mixed":
252             dataset.iloc[counter,12] = 9
253         elif element["Ethnicity"] == "Pacifica":
254             dataset.iloc[counter,12] = 10
255         elif element["Ethnicity"] == "middle eastern":
256             dataset.iloc[counter,12] = 11
```

```
257         print(dataset["Class/ASD_Traits"].unique())
258         print(dataset["Sex"].unique())
259         print(dataset["Family_mem_with_ASD"].unique())
260         print(dataset["Jaundice"].unique())
261         print(dataset["Ethnicity"].unique() )

262     """Partición de datos para Trainng (80%) Test & Validation (20%)
            [80% Test & 20% Validation]"""

263     dataset_train , dataset_test_validation =
        train_test_split(dataset.astype(np.float32) , test_size=0.2,
        random_state=42)
264     dataset_test , dataset_validation =
        train_test_split(dataset_test_validation , test_size=0.2,
        random_state=42)

265     """Separación de Datos y Labels"""

266     dataset_train_data , dataset_train_label = dataset_train.iloc[:, :-1] ,
        dataset_train.iloc[:, -1]
267     dataset_test_data , dataset_test_label = dataset_test.iloc[:, :-1] ,
        dataset_test.iloc[:, -1]
268     dataset_validation_data , dataset_validation_label =
        dataset_validation.iloc[:, :-1] , dataset_validation.iloc[:, -1]

269     """DS Original"""

270     dataset_text_train , dataset_text_test_validation =
        train_test_split(dataset , test_size=0.2, random_state=42)
271     dataset_text_test , dataset_text_validation =
        train_test_split(dataset_text_test_validation , test_size=0.2,
        random_state=42)

272     """Conversión DS en formato TF"""

273     TF_train =
            tfdf.keras.pd_dataframe_to_tf_dataset(dataset_text_train,
            label="Class/ASD_Traits")
274     TF_test =
            tfdf.keras.pd_dataframe_to_tf_dataset(dataset_text_test,
            label="Class/ASD_Traits")
275     TF_validation =
          tfdf.keras.pd_dataframe_to_tf_dataset(dataset_text_validation,
          label="Class/ASD_Traits")
```

```
276         """---
277         # Implementación de Modelos sin Validación Cruzada

278         Modelos basados en Decision Tree
279         """

280         model_fit_RF, dataset_test_predictions_RF , test_loss_RF ,
             test_accuracy_RF , training_time_RF
             =  Decision_Tree_model_fit("RF") #Random Forest

281         model_fit_GB, dataset_test_predictions_GB , test_loss_GB ,
             test_accuracy_GB , training_time_GB
            = Decision_Tree_model_fit("GB") #Gradient Boosted

282         """Modelos Secuenciales"""

283         model_fit_Nadam , dataset_test_predictions_Nadam ,
             test_loss_Nadam ,test_accuracy_Nadam , training_time_Nadam
             = sequential_model_fit_by_optimizer('Nadam')

284         model_fit_Adam , dataset_test_predictions_Adam , test_loss_Adam ,
            test_accuracy_Adam , training_time_Adam
            = sequential_model_fit_by_optimizer('Adam')

285         model_fit_RMSprop , dataset_test_predictions_RMSprop ,
            test_loss_RMSprop,test_accuracy_RMSprop ,
             training_time_RMSprop
             =   sequential_model_fit_by_optimizer('RMSprop')

286         model_fit_Adamax , dataset_test_predictions_Adamax ,
             test_loss_Adamax, test_accuracy_Adamax , training_time_Adamax
          = sequential_model_fit_by_optimizer('Adamax')

287         model_fit_Adagrad , dataset_test_predictions_Adagrad ,
            test_loss_Adagrad , test_accuracy_Adagrad , training_time_Adagrad
             = sequential_model_fit_by_optimizer('Adagrad')

288         model_fit_Adadelta , dataset_test_predictions_Adadelta ,
            test_loss_Adadelta ,test_accuracy_Adadelta ,training_time_Adadelta
            = sequential_model_fit_by_optimizer('Adadelta')
```

```
"""---

# Comparación de Modelos sin Validación Cruzada
"""

#Decision Tree models
print("Test accuracy with Random Forest: ",test_accuracy_RF)
print("Test accuracy with Grandient Boosted: ",test_accuracy_GB)

#Sequential Models
print("Test accuracy with Adam: ",test_accuracy_Adam)
print("Test accuracy with Nadam: ",test_accuracy_Nadam)
print("Test accuracy with RMSprop: ",test_accuracy_RMSprop)
print("Test accuracy with Adamax: ",test_accuracy_Adamax)
print("Test accuracy with Adagrad: ",test_accuracy_Adagrad)
print("Test accuracy with Adadelta: ",test_accuracy_Adadelta)

#Sequential Models
print("Loss with Adam: ",test_loss_Adam)
print("Loss with Nadam: ",test_loss_Nadam)
print("Loss with RMSprop: ",test_loss_RMSprop)
print("Loss with Adamax: ",test_loss_Adamax)
print("Loss with Adagrad: ",test_loss_Adagrad)
print("Loss with Adadelta: ",test_loss_Adadelta)

#Decision Tree models
print("Test accuracy with Random Forest: ",training_time_RF)
print("Test accuracy with Grandient Boosted: ",training_time_GB)

#Sequential Models
print("Test accuracy with Adam: ",training_time_Adam)
print("Test accuracy with Nadam: ",training_time_Nadam)
print("Test accuracy with RMSprop: ",training_time_RMSprop)
print("Test accuracy with Adamax: ",training_time_Adamax)
print("Test accuracy with Adagrad: ",training_time_Adagrad)
print("Test accuracy with Adadelta: ",training_time_Adadelta)
```

```
"""# Implementación de Modelos con Validación Cruzada

Decision Trees con Validación Cruzada (K=10)
"""

print('\n---------------------------------------------------------------------')
print('RANDOM FOREST')
print('---------------------------------------------------------------------')
acc_RF , loss_RF = Decision_Tree_model_fit_kfold("RF")
print('\n---------------------------------------------------------------------')
print('GRADIENT BOOSTED')
print('---------------------------------------------------------------------')
acc_GB , loss_GB = Decision_Tree_model_fit_kfold("GB")

"""Modelos Secuenciales con Validación Cruzada (k=10)"""

print('\n---------------------------------------------------------------------')
print('ADAM')
print('---------------------------------------------------------------------')
acc_Adam , loss_Adam =
sequential_model_fit_by_optimizer_kfold('Adam')
print('\n---------------------------------------------------------------------')
print('NADAM')
print('---------------------------------------------------------------------')
acc_Nadam , loss_Nadam =
sequential_model_fit_by_optimizer_kfold('Nadam')
print('\n---------------------------------------------------------------------')
print('RMSPROP')
print('---------------------------------------------------------------------')
acc_RMSProp , loss_RMSProp =
sequential_model_fit_by_optimizer_kfold('RMSProp')
print('\n---------------------------------------------------------------------')
print('ADAMAX')
print('---------------------------------------------------------------------')
acc_Adamax , loss_Adamax =
sequential_model_fit_by_optimizer_kfold('Adamax')
print('\n---------------------------------------------------------------------')
print('ADAGRAD')
print('---------------------------------------------------------------------')
acc_Adagrad , loss_Adagrad =
sequential_model_fit_by_optimizer_kfold('Adagrad')
print('\n---------------------------------------------------------------------')
print('ADADELTA')
print('---------------------------------------------------------------------')
acc_Adadelta , loss_Adadelta =
sequential_model_fit_by_optimizer_kfold('Adadelta')
```

```
"""# Comparación de Modelos con Validación Cruzada"""

#Decision Tree
print("Test accuracy with Random Forest: ",acc_RF)
print("Test accuracy with Gradient Bosted: ",acc_GB)

#Sequential Models
print("Test accuracy with Adam: ",acc_Adam)
print("Test accuracy with Nadam: ",acc_Nadam)
print("Test accuracy with RMSProp: ",acc_RMSProp)
print("Test accuracy with Adamax: ",acc_Adamax)
print("Test accuracy with Adagrad: ",acc_Adagrad)
print("Test accuracy with Adadelta: ",acc_Adadelta)

#Decision Tree
print("Test loss with Random Forest: ",loss_RF)
print("Test loss with Gradient Bosted: ",loss_GB)

#Sequential Models
print("Test loss with Adam: ",loss_Adam)
print("Test loss with Nadam: ",loss_Nadam)
print("Test loss with RMSProp: ",loss_RMSProp)
print("Test loss with Adamax: ",loss_Adamax)
print("Test loss with Adagrad: ",loss_Adagrad)
print("Test loss with Adadelta: ",loss_Adadelta)

"""---


**Matrices de Confusión**

# Matrices de Confusión

Matrices de confusión de modelos de Decision Tree
"""

generate_conf_matrix(dataset_test_label,dataset_test_predictions_RF,'Random Forest')
generate_conf_matrix(dataset_test_label,dataset_test_predictions_GB,'Gradient Boosted')
```

```
"""
---
Matrices de confusión de modelos secuenciales"""

generate_conf_matrix(dataset_test_label,dataset_test_predictions_Adam,'Adam')
generate_conf_matrix(dataset_test_label,dataset_test_predictions_Nadam,'Nadam')
generate_conf_matrix(dataset_test_label,dataset_test_predictions_RMSprop,'RMSprop')
generate_conf_matrix(dataset_test_label,dataset_test_predictions_Adamax,'Adamax')
generate_conf_matrix(dataset_test_label,dataset_test_predictions_Adagrad,'Adagrad')
generate_conf_matrix(dataset_test_label,dataset_test_predictions_Adadelta,'Adadelta')

"""---

# Gráficas de Precisión y Pérdidas

Modelos Decision Tree
"""

Decision_Tree_generate_accuracy_loss_graphs(model_fit_RF)
Decision_Tree_generate_accuracy_loss_graphs(model_fit_GB)

"""
---
Modelos Secuenciales"""

generate_accuracy_loss_graphs(model_fit_RMSprop,'RMSprop')
generate_accuracy_loss_graphs(model_fit_Nadam,'Nadam')
generate_accuracy_loss_graphs(model_fit_Adam,'Adam')
generate_accuracy_loss_graphs(model_fit_Adamax,'Adamax')
generate_accuracy_loss_graphs(model_fit_Adagrad,'Adagrad')
generate_accuracy_loss_graphs(model_fit_Adadelta,'Adadelta')
```

B) Código del segundo estudio

Código correspondiente al estudio “Técnicas de 'Deep Learning' sobre datos ‘Eye Tracking’”

```
# -*- coding: utf-8 -*-
"""TFG_Et3_v05_29_2.ipynb

# Importación de Librerías
"""

import pandas as pd
import numpy as np
import os
import glob
import csv
import math
import time
import psutil

import tensorflow as tf
from tensorflow.keras.layers import LSTM, Dense
import matplotlib.pyplot as plt

#conda env config vars set SKLEARN_ALLOW_DEPRECATED_SKLEARN_PACKAGE_INSTALL=True
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split

"""# Asignación de Datos y Metadatos
Este conjunto de código se ejecuta una sola vez para generar un único dataset con los valores de entrenamiento y test reales


---
Unión del conjunto de datasets en uno único
"""

dataset_union = [i for i in glob.glob('DS/*.{}'.format('csv'))]
dataset = pd.concat([pd.read_csv(f) for f in dataset_union ])
dataset.head()
```

```
28          """Lectura de los archivos correspondientes al dataset y al
            metadata"""

29          dataset = pd.read_csv("DataSet.csv")
30          metadata = pd.read_csv("Metadata_Participants.csv")

31          """Preprocesado Inicial

32          ---

33          Detección de valores del DataSet sin identificar (Valores
            pertenecientes a sujetos no identificados)

34          Detecta el conjunto de datos perteneciente a un mismo participante
            (Utilizando la variable ID) y asigna el valor de etiqueta
            correspondiente

35          Si no detecta el ID del participante asigna una etiqueta "NODATA"
            para su posterior eliminación
36          """

37          dataset_with_labels = dataset
38          counter=1
39          previous_row=0
40          number_values = len(dataset_with_labels)

41          for row in dataset_with_labels["Participant"]:
42              try:
43                  if row != previous_row:
44                      dataset_with_labels.loc[dataset_with_labels["Participant"] ==
                     row , "LABEL"] =
                     metadata["Class"][metadata["ParticipantID"] ==
                     int(row)].values[0]

45                      previous_row = row

46                      print("--> Row ", counter, " of ", number_values, " --- ", row,
                            ",",

47                  metadata["ParticipantID"][metadata["ParticipantID"] ==
                 int(row)].values[0], " , ",

48                  metadata["Class"][metadata["ParticipantID"] ==
                    int(row)].values[0])

49              except ValueError as error:
```

```
50              if row != previous_row:
51                  dataset_with_labels.loc[dataset_with_labels["Participant"]
            == row , "LABEL"] = "NODATA"
52                  previous_row = row
53                  print("--> Row ", counter, " of ", number_values, " --- ", row,
           ", NOT VALID")
54              finally:

55                  counter+=1

56                  #Guardar el Archivo .csv
57
        dataset_with_labels.to_csv("DataSet_With_Labels_NoData.csv"
      , index=False)

58      """Elimina las variables etiquetadas como NODATA"""

59      dataset_with_labels_and_nodata =
     pd.read_csv("DataSet_With_Labels_NoData.csv")

60      with open("DataSet_With_Labels.csv", 'w', newline='') as
        DataSet_With_Labels_csv:
61          dataset_with_labels = csv.writer(DataSet_With_Labels_csv)
62          dataset_with_labels.writerow
            (dataset_with_labels_and_nodata[1:2])

63          counter = 0

64          for row in dataset_with_labels_and_nodata["LABEL"]:
65              if not row == "NODATA":
66                  dataset_with_labels.writerow
            (dataset_with_labels_and_nodata.iloc[counter].values)
67                  print("--> Adding Row number ",counter, " --- ",row)

68              else:
69                  print("--> NODATA Detected")
70                  counter+=1

71      """# Pre Procesado de Datos

72      Cargar el DataSet con los metadatos (Etiquetas)
73      """

74      dataset_with_labels =
       pd.read_csv("DataSet_With_Labels.csv").replace('-',pd.NA)

75      """Borrado de las columnas despreciables para el estudio"""
```

```
76          columns_to_drop = [
77          "Unnamed: 0","RecordingTime [ms]",
                    "Time of     Day [h:m:s:ms]","Trial",
            "Stimulus","Export Start Trial Time [ms]",
            "Export End Trial Time [ms]",
            "Color","Tracking Ratio [%]","Category Group",
            "Category Right","Category Left","Index Right","Index Left",
            "AOI Name Right","AOI Name Left","Annotation Name",
            "Annotation Description","Annotation Tags",
           "Mouse Position X [px]","Mouse Position Y [px]",
           "Scroll Direction X","Scroll Direction Y","Content","Port Status",
           "AOI Group Right","AOI Scope Right","AOI Order Right",
           "AOI Group Left","AOI Scope Left","AOI Order Binocular",
           "groupe d'enfants"
78          ]

79          dataset_with_labels =
            dataset_with_labels.drop(columns=columns_to_drop)

80          """Detección y aproximación de los datos nulos"""

81          def value_is_valid(value_to_inspect):
82              try:
83                  return float(value_to_inspect)
84              except ValueError:
85                  return np.nan

86              index_data_pending = []
87              index_temp_data = []
88              fixed_columns = 2
89              length_dataset = len(dataset_with_labels)

90              for index in range(length_dataset):
91                  print("-> Indice de Fila ",index,
                    " Valores Temporales Almacenados:
                    ",len(index_temp_data)," Valores Pendientes:
                    ",len(index_data_pending))
92                  nan_counter = 0
93                  nan_column = []
94                  last_data_of_participant = False

95                  if dataset_with_labels["Participant"].iloc[index] !=
                            dataset_with_labels["Participant"].iloc[index+1]:
96                              last_data_of_participant = True
97                      if dataset_with_labels["Participant"].iloc[index] !=
                                    dataset_with_labels["Participant"].iloc[index-1]:
```

```
98                  if (len(index_data_pending) > 0) or
                        (len(index_temp_data) > 0):

99                      print("--> Queda información sin recuperar en
                        subconjunto de datos")
100                     last_data_of_participant = False
101                     index_data_pending = []
102                     index_temp_data = []

103             for column in range(len(dataset_with_labels.columns)):
104                 if (pd.isnull(dataset_with_labels.iloc[index,column])):
105                     nan_counter += 1
106                     nan_column.append(column)

107                 if (nan_counter == len(dataset_with_labels.columns) –
                        fixed_columns):
108                     dataset_with_labels =
                        dataset_with_labels.drop(index)
109                     length_dataset -= 1
110                     print("--> Fila Eliminada")
111                 else:
112                     for column in nan_column:

113                         value = dataset_with_labels.iloc[index,column]
114                         last_value = dataset_with_labels.iloc[index-1,column]
115                         next_value =
                            dataset_with_labels.iloc[index+1,column]

116                         if (pd.isnull(value)):
117                             value = np.nan

118                             if (index != 0 and not pd.isna(last_value)):
119                                 last_value = value_is_valid(last_value)
120                             else:
121                                 last_value = np.nan


122                             if (index != length_dataset and
                                not pd.isna(next_value)):
123                                     next_value =
                                        value_is_valid(next_value)
124                                 else:
125                                     next_value = np.nan
```

```
126                     if (not math.isnan(last_value) and
                         not math.isnan(next_value)):

127                         if (last_value == next_value):
128                             dataset_with_labels.iloc[index,column] =
                                last_value
129                         # Valor NaN entre dos valores actualizado
130                         else:
131                             dataset_with_labels.iloc[index,column] =
132                             (next_value + last_value)/2
133                         # Valor NaN entre dos valores aproximado

134                     elif (not math.isnan(last_value) and
                          math.isnan(next_value)):

135                         index_temp_data.append([index-1,column])
136                         # Valor Temporal Almacenado

137                     elif (last_data_of_participant == False) and
                          (math.isnan(last_value) and
                           math.isnan(next_value)):

138                         index_data_pending.append([index,column])
139                     # Valor de Columna Pendiente de actualización
140
141                     elif (last_data_of_participant == True) or
                    (math.isnan(last_value) and not
                     math.isnan(next_value)):

142                         index_row_last_data = -1

143                         for element_temp_data in
                        reversed(index_temp_data):

144                             if element_temp_data[1] == column:
145                                 index_row_last_data =
                                element_temp_data[0]
146                             break

147                         if index_row_last_data == -1:
148                             temp_data = np.nan
149                         else:
150                             temp_data = dataset_with_labels.iloc
                        [index_row_last_data,column]
```

```
151                     if (math.isnan(temp_data) or temp_data ==
                            next_value):
152                         dataset_with_labels.iloc[index,column] =
                        next_value

153                         index_row_data_pending = []
154                         index_data_pending_updated = []

155                         for index, data_pending_column in
                            index_data_pending:

156                             if data_pending_column == column:
157                                 index_row_data_pending.append(index)
158                             else:
159                                 index_data_pending_updated.append(
                                (index, data_pending_column))

160                         index_data_pending =
                            index_data_pending_updated

161                         dataset_with_labels.iloc
                            [index_row_data_pending,column] =
                            next_value

162                 #Valores Anteriores actualizados a valor existente

163                     else:
164                         dataset_with_labels.iloc[index,column] =
                        next_value
165                         index_row_data_pending = []

166                         for index, data_pending_column in
                            index_data_pending:

167                             if data_pending_column[1] == column:
168                                 index_row_data_pending.append(index)

169                         len_index_row_data_pending =
                        len(index_row_data_pending)


170                             delta = (next_value –
                            temp_data)/len_index_row_data_pending
171                             increment_counter = 1
```

```
172                                 for row in index_row_data_pending:
173                                     if (temp_data < next_value):
174                                         dataset_with_labels.iloc[row,column] =
                                    temp_data + increment_counter*delta
175                                     else:
176                                         dataset_with_labels.iloc[row,column] =
                                    temp_data - increment_counter*delta

177                                     increment_counter += 1

178                                 index_data_pending = [element_index_data
                                for element_index_data in
                                    index_data_pending
                                    if element_index_data !=
                                    [index_row_data_pending, column]]

179         # Valores Anteriores aproximados a valor existente
180                             index += 1  #Re-actualización de index

181         dataset_with_labels.to_csv("DataSet_With_Labels_Less_NaN.csv",
             index=False)

182         """Detección de los valores atípicos por medio del rango
            intercuartil"""

183         dataset_with_labels =
            pd.read_csv("DataSet_With_Labels_Less_NaN.csv")

184         columns =
            dataset_with_labels.drop(columns=["Participant","LABEL"]).columns

185         for column in columns:
186             mean_column  = dataset_with_labels[column].mean()
187             dataset_with_labels[column] =
            dataset_with_labels[column].fillna(mean_column)

188             Q1 = dataset_with_labels[column].quantile(0.25)
189             Q3 = dataset_with_labels[column].quantile(0.75)
190             IQR = Q3 - Q1

191             upper_limit = Q3 + 1.5 * IQR
192             lower_limit = Q1 - 1.5 * IQR

193         dataset_with_labels.loc[dataset_with_labels[column] > upper_limit,
            column] = upper_limit
194         dataset_with_labels.loc[dataset_with_labels[column] < lower_limit,
            column] = lower_limit
```

```
195  dataset_with_labels.to_csv("DataSet_With_Labels_Less_NaN
     _without_outliers.csv", index=False)

196  """# Selección de Características a Estudiar

197  Correlación de las características respecto al valor label
198  """

199  #from google.colab import drive
200  #drive.mount('/content/drive')

201  from scipy.stats import pearsonr

202  dataset_with_labels = pd.read_csv
     ("DataSet_With_Labels_Less_NaN_without_outliers.csv")

203  column_corr_pvalue = []
204  columns =
         dataset_with_labels.drop(columns=
             ["Participant","LABEL"]).columns

205  for column in columns:
206      corr, p_value = pearsonr(dataset_with_labels[column],
             dataset_with_labels["LABEL"].replace({'TD':0,'ASD':1}))

207      if not math.isnan(corr):
208          column_corr_pvalue.append([column,corr,p_value])
209      else:
210          column_corr_pvalue.append([column,0,0])

211  column_corr_pvalue.sort(key=lambda x:
         abs(x[1]),reverse=True)

212  for element in column_corr_pvalue:
213      print(f"-> {element[0]}: Correlation = {element[1]:.3f} ,
214      P-Value: {element[2]:.3f}")
```

```
215         """Selección de las características a estudiar
216         (según correlación)"""
            features = [

217         "Gaze Vector Right Y",
218         "Gaze Vector Right Z",
219         "Gaze Vector Left Y",
220         "Gaze Vector Left Z",
221         "Point of Regard Right X [px]",
222         "Point of Regard Left X [px]",
223         "Point of Regard Left Y [px]",
224         "Point of Regard Right Y [px]",

225         "Pupil Diameter Right [mm]",
226         "Pupil Diameter Left [mm]",
227         "Eye Position Right X [mm]",
228         "Pupil Position Left X [px]",
229         "Eye Position Left Y [mm]",
230         "Eye Position Right Y [mm]",
231         "Eye Position Right Z [mm]",
232         "Eye Position Left Z [mm]",

233         "Pupil Position Right Y [px]",
234         "Pupil Position Left Y [px]",
235         "Pupil Position Right X [px]",
236         "Pupil Size Left Y [px]",
237         "Pupil Size Left X [px]",
238         "Pupil Size Right Y [px]",
239         "Pupil Size Right X [px]",
240         "Eye Position Left X [mm]"

241         #   "Gaze Vector Right X",    #Correlación 0
242         #   "Gaze Vector Left X"      #Correlación 0

243         ]

244         """**Pre Procesamiento de datos antes del entrenamiento**
245         ---
246         Separación e intercalación de ambos casos de estudio
            para crear los dos subconjuntos de datos (train y test)
247         """

248         dataset_with_labels_ASD =
            dataset_with_labels[dataset_with_labels["LABEL"]=="ASD"]
249         dataset_with_labels_TD =
            dataset_with_labels[dataset_with_labels["LABEL"]=="TD"]
```

```
250     id_ASD = dataset_with_labels_ASD["Participant"].unique()
251     id_TD = dataset_with_labels_TD["Participant"].unique()
        [:len(id_ASD)]
252     id_dataset = [value for par in zip(id_ASD, id_TD)
        for value in par]

253     dataset_with_labels_train_mix =
            pd.DataFrame(columns=dataset_with_labels.columns)
254     dataset_with_labels_test_mix = []

255     len_train=int(len(id_dataset)//1.25)

256     for value in id_dataset[:len_train]:
257         dataset_with_labels_train_mix =
                    pd.concat([dataset_with_labels_train_mix,
                    dataset_with_labels[dataset_with_labels
                ["Participant"] ==int(value)]])

258     for value in id_dataset[len_train:]:
259         dataset_with_labels_test_mix.append
            ([value,dataset_with_labels[dataset_with_labels
            ["Participant"]==int(value)]])

260     """Separación entre datos y etiquetas para cada uno de los
        subconjuntos"""

261     dataset_train_label =
        dataset_with_labels_train_mix["LABEL"]
         .replace({'TD':0,'ASD':1})

262     dataset_train_data =
        dataset_with_labels_train_mix[features].astype(float)

263     dataset_test_data = []
264     dataset_test_label = []

265     for element in dataset_with_labels_test_mix:
266         dataset_test_label.append(element[1]["LABEL"]
        .replace({'TD':0,'ASD':1}))

267     dataset_test_data.append(element[1][features].astype(float))
```

```
268                 """# Funciones para entrenamiento y test de los Modelos"""

269             def model_fit_kfold(model,epochs = 50,folder_init = 1):

                    kfolder = KFold(n_splits=10, shuffle=False)
270                 folder_number =  1
271                 acc_per_fold = []
272                 loss_per_fold = []

273                 #Iteración por folder

274                 data = dataset_train_data
275                 label = dataset_train_label

276                 for test_data , test_label in
                    zip(dataset_test_data,dataset_test_label):

277                     data = np.concatenate((data, test_data), axis=0)
278                     label = np.concatenate((label, test_label), axis=0)

279                     print("Use of ",len(features)," Features\n")
280                     total_start_time = time.time()

281                     for train , test in kfolder.split(data,label):
282                         participant_iter = participant_start
283                         if folder_number == folder_init:
284                             model.compile(loss=
                                  "sparse_categorical_crossentropy",
                              optimizer="Adamax",
                              metrics=["sparse_categorical_accuracy"] )

285                             print('------------------------------------------------------------')
286                             print(f'-> Training for folder number
                                 {folder_number}')

287                             folder_start_time = time.time()

288                             model_fit = model.fit(data[train],label[train],epochs =
                            epochs,verbose=1)

289                             folder_training_time = time.time() - folder_start_time
```

```
290                 print('--> Training Time in folder: ',folder_training_time)
291                 print('------------------------------------------------------------------')
292                 print('  R  E  S  U  L  T  S  Folder
                    ',folder_number)
293                 print('------------------------------------------------------------------')

294                 values = model.evaluate(data[test],label[test])
295                 print(f'--> Score for folder {folder_number}:
                    {model.metrics_names[0]} of {values[0]};
                    {model.metrics_names[1]} of {values[1]*100}%\n')

296                 acc_per_fold.append(values[1] * 100)
297                 loss_per_fold.append(values[0])

298                 folder_init += 1

299             folder_number = folder_number + 1

300         total_training_time = time.time() - total_start_time

301         #Resumen por folder

302         print('------------------------------------------------------------------------')
303         print('-> Score per fold')

304         for i in range(0, len(acc_per_fold)):
305             print('------------------------------------------------------------------')
306             print(f'--> Fold {i+1} - Loss: {loss_per_fold[i]} - Accuracy:
            {acc_per_fold[i]}%')

307             print('------------------------------------------------------------------')
308             print('-> Average scores for all folds:')
309             print(f'--> Accuracy: {np.mean(acc_per_fold)} (+-
            {np.std(acc_per_fold)})')
310             print(f'--> Loss: {np.mean(loss_per_fold)}')
311             print(f'--> Training Time: {total_training_time}')
312             print('------------------------------------------------------------------\n')

313     def sequential_model():

314         model = tf.keras.models.Sequential()
315         model.add(tf.keras.layers.Dense(units=64, activation='relu'))
316         model.add(tf.keras.layers.Dropout(0.2))
317         model.add(tf.keras.layers.Dense(units=len(features),
             activation='relu'))
318         model.add(tf.keras.layers.Dense(units=2,
             activation='softmax'))
```

```
319         return model

320     def sequential_lstm_model():

321         model = tf.keras.models.Sequential()
322         model.add(LSTM(64, input_shape=(len(features),1)))
323         model.add(tf.keras.layers.Dense(units=64, activation='relu'))
324         model.add(tf.keras.layers.Dropout(0.2))
325         model.add(tf.keras.layers.Dense(units=len(features),
            activation='relu'))
326         model.add(tf.keras.layers.Dense(units=2,
            activation='softmax'))
327         return model

328     def sequential_convolutional_model():

329         model = tf.keras.models.Sequential()
330         model.add(tf.keras.Input(shape=(len(features),1)))

331         model.add(tf.keras.layers.Conv1D(filters=16, kernel_size=3,
            activation='relu', padding='same'))

332         model.add(tf.keras.layers.MaxPooling1D())
333         model.add(tf.keras.layers.Conv1D(filters=32, kernel_size=3,
            activation='relu', padding='same'))

334         model.add(tf.keras.layers.MaxPooling1D())
335         model.add(tf.keras.layers.Flatten())

336         model.add(tf.keras.layers.Dropout(0.2))
337         model.add(tf.keras.layers.Dense(units=32, activation='relu'))
338         model.add(tf.keras.layers.Dense(units=2,
            activation='softmax'))
339         return model

340     def sequential_convolutional_lstm_model():

341         model = tf.keras.models.Sequential()
342         model.add(tf.keras.Input(shape=(len(features),1)))

343         model.add(tf.keras.layers.Conv1D(filters=16, kernel_size=3,
            activation='relu', padding='same'))
344         model.add(tf.keras.layers.MaxPooling1D())

345         model.add(tf.keras.layers.Conv1D(filters=32, kernel_size=3,
            activation='relu', padding='same'))
```

```
346             model.add(tf.keras.layers.MaxPooling1D())
347             model.add(LSTM(64, input_shape=(128,1)))
348              model.add(tf.keras.layers.Flatten())

349             model.add(tf.keras.layers.Dropout(0.2))

350             model.add(tf.keras.layers.Dense(units=32, activation='relu'))
351             model.add(tf.keras.layers.Dense(units=2,
                activation='softmax'))
352         return model

353     def gru_model():

354             model = tf.keras.models.Sequential()
355             model.add(tf.keras.Input(shape=(len(features),1)))

356             model.add(tf.keras.layers.GRU(32))
357             model.add(tf.keras.layers.Dense(len(features)))
358             model.add(tf.keras.layers.Dropout(0.2))

359             model.add(tf.keras.layers.Dense(units=len(features),
               activation='relu'))
360             model.add(tf.keras.layers.Dense(units=2,
                activation='softmax'))
361             return model

362     #Función para parar el entorno de Google Colab

363     #def parar_entorno():
364     #  from google.colab import runtime
365     #  runtime.unassign()

366     """# Implementación y entrenamiento del Modelo Secuencial

367     Modelo Secuencial
368     """

369     sequential = sequential_model()
370     model_fit_kfold(sequential,50,1)

371     """Modelo LSTM"""

372     lstm = sequential_lstm_model()
373     model_fit_kfold(lstm,50,1)
```

```
374         """Modelo Convolucional"""

375         convolutional = sequential_convolutional_model()
376         model_fit_kfold(convolutional,50,1)

377         """Modelo Convolucional & LSTM"""

378         convolutional_lstm = sequential_convolutional_lstm_model()
379         model_fit_kfold(convolutional_lstm,50,1)

380         """Modelo GRU"""

381         gru = gru_model()
382         model_fit_kfold(gru,50,1)
```