Instituto de Biomecánica Clínica

Pre-Fall – Sistema inteligente para la prevención y predicción de caídas

E3.4 – Sistema experto para la prevención de caídas y evaluación de la eficacia de la rehabilitación

Proyecto	Pre-Fall – Sistema inteligente para la prevención y predicción de caídas
Entregable	E3.4 – Sistema experto para la prevención de caídas y evaluación de la eficacia de la rehabilitación

Contenido

1 Introducción

Este entregable está enmarcado en la tarea "T3.4: Sistema experto para la prevención de caídas y evaluación de la eficacia de la rehabilitación", perteneciente al paquete de trabajo "PT3 – Sistema experto de prevención de caídas" dentro del proyecto PRE-FALL. En este documento se presentarán las secciones más relevantes del software del sistema experto de recomendación.

2 Descripción del código

El sistema experto de recomendación consiste en un script de Python al que se le pasa como argumento uno o varios ficheros relativos a un paciente y este devuelve el riesgo de caída del mismo en forma de porcentaje, así como imprimir por pantalla al profesional las recomendaciones indicadas en el entregable "E3.3 Validación de los modelos de aprendizaje automático".

```
#Imports básicos
import pandas as pd
import numpy as np
import libraries
import random
import os
import sys
import io
import warnings
from joblib import load

''' ARGUMENTOS PARA PERSONALIZAR EL SCRIPT '''
#Argumento para imprimir por consola o no los resultados. Por defecto No
verbose=False
#Argumento para almacenar el resultado de las predicciones del modelo en un fichero txt. Por defecto Sí
save_txt=True


''' ARGUMENTOS DEL SCRIPT '''
# Obtener los argumentos del script
argumentos = sys.argv[1:]

#Si no se pasan argumentos, no se eejcuta el script
if len(argumentos)==0:
    print('Error: no se ha especificado ningún argumento')
    sys.exit(1)

#Almacenamiento de ficheros
ficheros=[]

#Si se le pasa solo un argumento
if len(argumentos)==1:

    #Leemos el argumento
    ruta = argumentos[0]

    #Si es un directorio
    if os.path.isdir(ruta):
        for archivo in os.listdir(ruta):
            ficheros.append(os.path.join(ruta, archivo))
    #Si es un nombre de fichero
    else:
        ficheros.append(ruta)

#Si se le pasan varios argumentos (ficheros)
else:
    for a in argumentos:
        ficheros.append(a)
```

En primer lugar, se comprueba que los argumentos pasados al script son correctos y posteriormente el número de ellos que corresponde al número de ficheros proporcionados al sistema.

El siguiente paso consiste en preparar el modelo entrenado tal y como se indicó en los entregables "E3.2 Modelos iniciales de aprendizaje automático" y "E3.3 Validación de los modelos de aprendizaje automático" así como un fichero de salida en caso de que el usuario así lo haya indicado.

```
''' PREPARACION DEL MODELO '''
modelo = load('modelo.joblib')

''' PREPARACION FICHERO SALIDA SI ES NECESARIO '''
if save_txt:
    prefix = "resultado"
    extension = ".txt"

    # Obtener la lista de archivos en el directorio
    directorio='./output'
    files = os.listdir(directorio)

    # Encontrar el archivo más reciente y su número
    latest_file_num = 0
    for file in files:
        if file.startswith(prefix) and file.endswith(extension):
            file_num = int(file[len(prefix):-len(extension)])
            if file_num > latest_file_num:
                latest_file_num = file_num

    # Crear un nuevo archivo con el siguiente número
    next_file_num = latest_file_num + 1
    next_file_name = prefix + str(next_file_num) + extension
    next_file_path = os.path.join(directorio, next_file_name)
```

A continuación, se calcula la probabilidad de caída de los valores de los ficheros del paciente. A cada fichero se le realiza un preprocesado para ajustar los datos puros de cada fichero de mediciones a datos que pueda manejar el modelo de la forma explicada en el entregable "E3.1 Procedimiento de depuración y preprocesado de los datos" incluyendo el filtro de los datos del acelerómetro como la identificación de las fases de la marcha.

```
''' OBTENEMOS LA PROBABILIDAD DE CAIDA DE CADA FICHERO '''
with warnings.catch_warnings():
    warnings.simplefilter('ignore')
    with open(next_file_path, 'w') as file:
        for fichero in ficheros:

            # Abrir el archivo en modo lectura
            with open(fichero, 'r') as f:
                # Leer el contenido del archivo
                contenido = f.read()

            # Reemplazar los tabuladores por espacios en memoria
            contenido = contenido.replace('\t', ' ')

            # Crear un objeto DataFrame de Pandas a partir del contenido
            datos = pd.read_csv(io.StringIO(contenido), delim_whitespace=True)

            #Renombramos columnas para adaptarlo al script libraries
            datos.rename(columns={'ACC_X': 'ax', 'ACC_Y': 'ay', 'ACC_Z': 'az'}, inplace=True)

            # Filtro de outliers general. Saltamos los primeros segundos para eliminar datos anomalos
            datos = libraries.filtro_acelerometro(datos.iloc[500:-500, :])

            #Definimos que el sujeto está caminando durante todo el proceso de medición. *A futuro esto puede modificarse
            datos['caminar'] = [1 for i in range(len(datos))]

            #Obtenemos las fases de la marcha
            with warnings.catch_warnings():
                warnings.simplefilter('ignore')
                # Filtro de tramos caminados
                if any(datos['caminar'] == 1):
                    if any(datos['caminar'][-10:] == 1):
                        datos['caminar'].iloc[-10:] = 0
                    datos['estado']=libraries.fases_marcha_global(datos)

            # Agrupa los datos por fases
            #datos=datos.dropna()
            fases = datos.groupby('estado')
```

Posteriormente se define la estructura del dataframe con el que trabajará directamente el modelo y se calculan los valores medios de los ejes de los sensores para cada fase de la marcha.

```
            # Crea un dataframe vacío para almacenar las características
            datos_final = pd.DataFrame(columns=['duracion_f1',
                                                'lax_mean_f1', 'lay_mean_f1', 'laz_mean_f1',
                                                'gx_mean_f1', 'gy_mean_f1', 'gz_mean_f1',
                                                'mx_mean_f1', 'my_mean_f1', 'mz_mean_f1',
                                                'qx_mean_f1', 'qy_mean_f1', 'qz_mean_f1','qw_mean_f1',
                                                'duracion_f2',
                                                'lax_mean_f2', 'lay_mean_f2', 'laz_mean_f2',
                                                'gx_mean_f2', 'gy_mean_f2', 'gz_mean_f2',
                                                'mx_mean_f2', 'my_mean_f2', 'mz_mean_f2',
                                                'qx_mean_f2', 'qy_mean_f2', 'qz_mean_f2','qw_mean_f2',
                                                'duracion_f3',
                                                'lax_mean_f3', 'lay_mean_f3', 'laz_mean_f3',
                                                'gx_mean_f3', 'gy_mean_f3', 'gz_mean_f3',
                                                'mx_mean_f3', 'my_mean_f3', 'mz_mean_f3',
                                                'qx_mean_f3', 'qy_mean_f3', 'qz_mean_f3','qw_mean_f3',
                                                'duracion_f4',
                                                'lax_mean_f4', 'lay_mean_f4', 'laz_mean_f4',
                                                'gx_mean_f4', 'gy_mean_f4', 'gz_mean_f4',
                                                'mx_mean_f4', 'my_mean_f4', 'mz_mean_f4',
                                                'qx_mean_f4', 'qy_mean_f4', 'qz_mean_f4','qw_mean_f4',
                                               ])
```

```
# Itera sobre cada grupo de fases
i=1
for fase, grupo in fases:
    # Calcula la duración de la fase
    duracion = grupo['TIME'].iloc[-1] - grupo['TIME'].iloc[0]
    duracion=len(grupo)/100

    #CALCULAMOS VALORES MEDIOS DE SENSORES PARA CADA FASE DE MARCHA
    #Aceleracion lineal
    lax_mean = grupo['LACC_X'].mean()
    lay_mean = grupo['LACC_Y'].mean()
    laz_mean = grupo['LACC_Z'].mean()

    #giroscopio
    gx_mean = grupo['GYR_X'].mean()
    gy_mean = grupo['GYR_Y'].mean()
    gz_mean = grupo['GYR_Z'].mean()

    #magnetometro
    mx_mean = grupo['MAG_X'].mean()
    my_mean = grupo['MAG_Y'].mean()
    mz_mean = grupo['MAG_Z'].mean()

    #quaterniones
    qx_mean = grupo['QUAT_X'].mean()
    qy_mean = grupo['QUAT_Y'].mean()
    qz_mean = grupo['QUAT_Z'].mean()
    qw_mean = grupo['QUAT_Z'].mean()

    #CALCULAMOS DESVIACION TIPICA DE SENSROES PARA CADA FASE DE MARCHA
    #Aceleracion lineal
    lax_mean = grupo['LACC_X'].std()
    lay_mean = grupo['LACC_Y'].std()
    laz_mean = grupo['LACC_Z'].std()

    #giroscopio
    gx_mean = grupo['GYR_X'].std()
    gy_mean = grupo['GYR_Y'].std()
    gz_mean = grupo['GYR_Z'].std()

    #magnetometro
    mx_mean = grupo['MAG_X'].std()
    my_mean = grupo['MAG_Y'].std()
    mz_mean = grupo['MAG_Z'].std()

    #quaterniones
    qx_mean = grupo['QUAT_X'].std()
    qy_mean = grupo['QUAT_Y'].std()
    qz_mean = grupo['QUAT_Z'].std()
    qw_mean = grupo['QUAT_Z'].std()
```

```
# Agrega las características al dataframe de características
datos_final = datos_final.append({'duracion_f'+str(i): duracion,
                                  'lax_mean_f'+str(i): lax_mean, 'lay_mean_f'+str(i): lay_mean, 'laz_mean_f'+str(i): laz_mean,
                                  'gx_mean_f'+str(i): gx_mean, 'gy_mean_f'+str(i): gy_mean, 'gz_mean_f'+str(i): gz_mean,
                                  'mx_mean_f'+str(i): mx_mean, 'my_mean_f'+str(i): my_mean, 'mz_mean_f'+str(i): mz_mean,
                                  'qx_mean_f'+str(i): qx_mean, 'qy_mean_f'+str(i): qy_mean, 'qz_mean_f'+str(i): qz_mean,'qw_mean_f'+str(i): qw_mean,

                                  }, ignore_index=True)
```

El siguiente paso es concatenar los resultados obtenidos e imprimirlos en un fichero dentro de la carpeta output del directorio que contendrá los valores de probabilidad de caída estimados por el modelo para el paciente en cuestión.

```
#Combinamos los resultados
datos_final = pd.concat([datos_final.iloc[0], datos_final.iloc[1], datos_final.iloc[2],datos_final.iloc[3]]).dropna()

# Crear un nuevo dataframe con una sola fila y las columnas resultantes
datos_final = pd.DataFrame([datos_final.tolist()], columns=datos_final.index)

#Predecimos la probabilidad de caida con el modelo
prediccion=modelo.predict_proba([datos_final.iloc[0].tolist()])[0][1]
texto="Probabildiad de caida del paciente fichero"+str(fichero)+": "+str(prediccion)
if verbose:
    print(texto)
if save_txt:
    file.write(texto+"\n")
```

Finalmente, se analizan los valores de probabilidad y se le proporcionan recomendaciones al profesional en función del valor obtenido.

```
#Comprobamos el valor de la probabilidad calculada por el modelo para indicarle al profesional
#algunas recomendaciones de cara a tratamiento y rehabilitacion del paciente
if prediccion>0.75:
    print("Riesgo alto de caída.")
    print("Recomendacion: Ejercicios de potenciación de miembros inferiores en espaderas (agarrado/a a una barra para mantener el "+
          "equilibrio y con silla detrás), practicar la marcha en barras paralelas y con andador siempre con supervisión y acompañado "+
          "con la silla detrás. Rehabilitación 3 o 4 días por semana a discreción del personal médico.")
elif prediccion>0.5:
    print("Riesgo medio de caída")
    print("Recomendación: Potenciación de miembros inferiores con ejercicios de musculación en las piernas y columna lumbar, practicar "+
          "subidas y bajadas de escaleras, marcha con ayudas técnicas (andador, dos/una muleta). Rehabilitación 3 días por semana.")
elif predicción>0.25:
    print("Riesgo bajo de caida")
    print("Recomendación: Trabajo de la marcha en circuito o caminar con obstáculos para mejorar el equilibrio en bipedestación. "+
          "Rehabilitación 2 días por semana."
else:
    print("Riesgo muy bajo de caida")
    print("No es necesario realizar tareas de rehabilitación salvo que el profesional lo considere aporpiado")
```