Universidad Simón Bolívar

Dpto. de Tecnología y Electrónica

EC7817 - Tópico especial II - Inteligencia Artificial en Biomédica

Proyecto: Predicción del consumo energético metabólico humano

Autor: William Chacón

Profesor: Miguel Altuve

August 4, 2022

El consumo energético de cada individuo varías de acuerdo a la salud, genética y actividad que éste realiza. El reducir el consumo energético de requerido para una actividad permite prolongar el tiempo de realización de la misma. Esto puede ser de mucha utilidad en el campo de la medicina, deportiva y hasta en el campo laboral.

Diversos estudios demuestran que las personas con discapacidad motriz suelen requerir más energía para realizar actividades de menor capacidad, lo que limita aún más su movilidad [1]. Diversos dispositivos robóticos (prótesis motorizadas, exoesqueletos, ente otros) se han desarrollado para reducir el costo energético de una persona al realizar actividades, tanto para paciente con carencia de movilidad como para personas sanas que debe desempeñar tareas de alta demanda energética. Estos dispositivos se componen por sensores, sistemas de control y actuadores, que deben ajustarse al paciente [1].

Si bien, en muchos casos, sobre todo en el área clínica, la estimación del consumo energético la realiza manualmente el doctor de acuerdo a sus observaciones y la retroalimentación verbal del paciente; existen algoritmos de estimación de consumo energético de un paciente. Uno es el algoritmo "body-in-the-loop", que consiste en un proceso de optimización de ajuste iterativo y automático de los parámetros de los dispositivos robóticos de asistencia para minimizar una función de costo fisiológico mientras una persona usa el dispositivo. La evaluación automática del costo energético es actualmente uno de los factores que limitan la velocidad en la traducción de estos algoritmos a dispositivos de asistencia del mundo real [1].

La estimación de consumo energético actualmente se realiza mediante la calorimetría, que se basa en la medición de consumo de oxígeno y producción de dióxido de carbono de paciente. Si bien, permite obtener un resultado, su uso prolongado no es viable debido a al equipo de medición, que consiste en una máscara que debe ir colocada en la boca en todo momento, la cual se conecta a un equipo de procesamiento [1].

En vista de lo impráctico del método más usado de medición de consumo energético, nace la necesi-

dad de buscar alternativas para obtener una medición precisa del metabolismo de cada paciente a partir de señales biológicas de diferente naturaleza, que pudieran ser de fácil medición con equipos compactos que no sea incómodos. En este proyecto, se pretende presentar una primera aproximación de una posible solución mediante modelos de Machine Learning que pudieran usar señales diferentes de usuarios y que demuestren ser capaces de adaptarse a cada individuo con suficiente precisión como para ser considerado apropiado para su uso.

Objetivo General

- Presentar diferentes modelos de Machine Learning capaces de adaptarse al problema a estudiar con suficiente precisión (mayor al 95

Código

Instanciación de bibliotecas

[2]:
```
# Se instancian las bibliotecas a implementar

import numpy as np
#import csv
#import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cbook as cbook
#import statistics as stats

import scipy
from scipy import stats
import scipy.cluster as scpcl
import scipy.cluster.hierarchy as scpch
import scipy.io as scpio
import scipy.signal as scpsg
import scipy.ndimage as scpni
import scipy.interpolate as scpnt
#from scipy.cluster.hierarchy import dendrogram
#from scipy.stats import boxcox
#from scipy.special import inv_boxcox

import sklearn
import sklearn.model_selection as sklms
import sklearn.linear_model as skllm
import sklearn.metrics as sklmt
import sklearn.neural_network as sklnn
import sklearn.cluster as sklcl
#from sklearn import cross_validation
#import sklearn.cross_validation as sklcv
#from sklearn.neural_network import MLPClassifier
#from sklearn.model_selection import train_test_split


# presentate data in table format
from prettytable import PrettyTable


# to load .mat data file
import mat73
import h5py
```

```
# modelo de RNN
import torch
import torch.nn as trnn
import torch.optim as trOptim
import torch.nn.functional as trFnl



#format of print for numpy float
np.set_printoptions(precision = 5, formatter = {'float_kind': lambda x: "{0:0.
 ↪5f}".format(x)}, threshold = 40, edgeitems = 20)
```

[3]:
```
# Configuration to display center plots

from IPython.core.display import HTML

HTML("""
<style>
.output_png {
    display: table-cell;
    text-align: center;
    vertical-align: middle;
}
</style>
""")
```

[3]:
```
<IPython.core.display.HTML object>
```

Variables Globales implementadas a lo largo del código

[4]:
```
# names
subjname = np.char.add(np.array(['Subject']*10).astype(np.str_), np.char.add( np.
 ↪floor_divide(np.arange(10)+1, 10).astype(np.str_), np.remainder(np.
 ↪arange(10)+1, 10).astype(np.str_)) )
filename = np.char.add( subjname, np.array(['.mat']*10).astype(np.str_) )
actvname = np.array(['Backwards', 'Cycling', 'Incline', 'Running', 'Walking']).
 ↪astype(np.str_)
measname = np.array(['APDM_Accel', 'EMG', 'Empatica_Accel', 'Empatica_Physio',␣
 ↪'Metabolics_System']).astype(np.str_)
varsname = np.array(['Data', 'Labels', 'SamplingRate']).astype(np.str_)
```

[5]:
```
def callData( data, i, j, k, l ):

```

```
    return data[i][ subjname[i] ][ actvname[j] ][ measname[k] ][ varsname[l] ][:
↪,:]

#endfunction


def callData1( data, i, j, k ):

    return data[i][ subjname[i] ][ actvname[j] ][ measname[k] ]

#endfunction


def callData2( data, i, j ):

    return data[i][ subjname[i] ][ actvname[j] ]

#endfunction
```

[6]:
```
#filtros
def filtros(data, isubj, iactv, imeas):
    IirHP30 = scpsg.iirdesign( wp=31, ws=29, gpass=1, gstop=80, analog=False,␣
↪ftype='ellip', fs=callData( data, isubj, iactv, imeas, 2 )[0][0] )
    IirLP350 = scpsg.iirdesign( wp=349, ws=351, gpass=1, gstop=80, analog=False,␣
↪ftype='ellip', fs=callData( data, isubj, iactv, imeas, 2 )[0][0] )

    return IirHP30, IirLP350
```

[83]:
```
# metodologías de la regresión lineal
methodName = ['5k entrenamiento','5k evaluación','5k toda la data','10k␣
↪entrenamiento','10k evaluación','10k toda la data']
```

[ ]:
```

```

Funciones implementadas a lo largo del código

[ ]:
```

```

[7]:
```
# Cálculo de los Parámetros estadísticos de interés

def StatsVars( data, title ):


    # Inicialización
```

```
    data_Mean = np.zeros((np.size(data,1),1))       #media
    data_Median = np.zeros((np.size(data,1),1))     #mediana
    data_Mode = np.zeros((np.size(data,1),1))       #moda
    data_Max = np.zeros((np.size(data,1),1))        #máximo
    data_Min = np.zeros((np.size(data,1),1))        #mínimo
    data_Range = np.zeros((np.size(data,1),1))      #rango
    data_Desv = np.zeros((np.size(data,1),1))       #desviación
    #data_Skew = np.zeros((np.size(data,1),1))       #asimetría
    #data_Kurt = np.zeros((np.size(data,1),1))       #curtosis


    # cálculo de los parámetros estadísticos

    data_Mean = np.mean(data, axis = 0)
    data_Median = np.median(data, axis = 0)
    data_Mode = stats.mode(data, axis = 0)
    data_Max = np.max(data, axis = 0)
    data_Min = np.min(data, axis = 0)
    data_Range = data_Max - data_Min
    data_Desv = np.std(data, axis = 0)
    #data_Skew = stats.skew(data, axis = 0, bias = 0)
    #data_Kurt = stats.kurtosis(data, axis = 0, bias = 0)

    # Se inserta el formato de presentación de los datos
    np.set_printoptions(formatter={'float': lambda x: "{:.5e}".format(x)},␣
↪suppress= True)

    # Presentación de los valores de los parámetros estadísticos estudiados

    print('la media de los atributos para la ' + title + ' es:');
    print(data_Mean);
    print();
    print('la mediana de los atributos para la ' + title + ' es:');
    print(data_Median);
    print();
    print('la moda de los atributos para la ' + title + ' es:');
    print(data_Mode);
    print();
    print('el valor máximo de los atributos para la ' + title + ' es:');
    print(data_Max);
    print();
    print('el valor mínimo de los atributos para la ' + title + ' es:');
    print(data_Min);
    print();
    print('lel rango de los atributos para la ' + title + ' es:');
    print(data_Range);
```

```
    print();
    print('la desviación estándar de los atributos para la ' + title + ' es:');
    print(data_Desv);
    print();
    #print('la asimetría de los atributos para la ' + title + ' es:');
    #print(data_Skew);
    #print();
    #print('el curtosis de los atributos para la ' + title + ' es:');
    #print(data_Kurt);
    #print();


    # retorna las variables estadísticas de interés

    return data_Mean, data_Median, data_Mode, data_Max, data_Min, data_Range,␣
 ↪data_Desv, data_Skew, data_Kurt;

#endfunction
```

[8]:
```
# funciones de procesamiento

def APDLprocessing( data, SampRate ):

    aux = np.copy( data )
    aux[2:,:] = scpsg.savgol_filter( data[2:,:], window_length= int(SampRate/0.
 ↪1) +1, polyorder=3, deriv=0, delta=1.0, axis=1, mode='interp')
    return aux

#endfunction


def ECGprocessing( data, SampRate ):

    aux = np.copy( data )
    aux[2:,:] = scpsg.filtfilt( IirHP30[0][:], IirHP30[0][:], data[2:,:], axis=1␣
 ↪)
    aux[2:,:] = scpsg.filtfilt( IirLP350[0][:], IirLP350[0][:], aux[2:,:],␣
 ↪axis=1 )
    aux[2:,:] = scpsg.savgol_filter( aux[2:,:], window_length= int(SampRate) +1,␣
 ↪polyorder=3, deriv=0, delta=1.0, axis=1, mode='interp')
    return aux

#endfunction

def EpAcProcessing( data, SampRate ):
```

```
    aux = np.copy( data )
    aux[2:,:] = scpsg.savgol_filter( data[2:,:], window_length= int(SampRate/0.
↪1) +1, polyorder=3, deriv=0, delta=1.0, axis=1, mode='interp')
    return aux

#endfunction


def PhysProcessing( data, SampRate ):

    aux = np.copy( data )
    aux[2:,:] = scpsg.savgol_filter( data[2:,:], window_length= int(SampRate/0.
↪1) +1, polyorder=3, deriv=0, delta=1.0, axis=1, mode='interp')
    return aux

#endfunction

def MetaProcessing( data, SampRate ):

    aux = np.copy( data )
    aux[2:np.size( aux[:,:], 0) -2,:] = scpsg.savgol_filter( data[2:np.size(␣
↪data[:,:], 0)-2 ,:], window_length= int(SampRate) +1, polyorder=3, deriv=0,␣
↪delta=1.0, axis=1, mode='interp')
    aux[-1,:] = scpsg.savgol_filter( aux[-1,:], window_length= int(SampRate) +1,␣
↪polyorder=3, deriv=0, delta=1.0, axis=0, mode='interp')
    return aux

#endfunction


def filtradoSeñal( data, SampRate, i ):

    # Se eliminan los vectores nan
    data = np.delete( data, np.where( np.isnan(data) )[0], axis=0 )


    # Se filtra la señal de acuerdo al tipo de señal
    if i == 0:
        return APDLprocessing( data, SampRate )
    elif i == 1:
        return ECGprocessing( data, SampRate )
    elif i == 2:
        return EpAcProcessing( data, SampRate )
    elif i == 3:
        return PhysProcessing( data, SampRate )
    elif i == 4:
        return MetaProcessing( data, SampRate )
```

```
    '''
    # Se filtra la señal de acuerdo al tipo de señal
    filtProc = {
        0: APDLprocessing( data, SampRate ),
        1: ECGprocessing( data, SampRate ),
        2: EpAcProcessing( data, SampRate ),
        3: PhysProcessing( data, SampRate ),
        4: MetaProcessing( data, SampRate ),
    }
    return filtProc.get(i, None )
    '''

#endfunction
```

[9]:
```
def proc1ActData( data, procDName, isubj, iactv ):

    for i in range( np.size(measname,0) ):
        #callData1( data, 0, 0, i ).create_group("dataprcs")
        #del callData1( data, 0, 0, i )["dataprcs1"]
        callData1( data, isubj, iactv, i )[procDName] = filtradoSeñal(␣
 ↪callData(data, isubj, iactv, i, 0 )[:,:], callData(data, isubj, iactv, i, 2␣
 ↪)[0][0], i )

    #endfor

#endfunction


def procAllActData( data, procDName, isubj ):

    for i in range( np.size(actvname,0) ):

        proc1ActData( data, procDName, isubj, i )

    #endfor

#endfunction


def procAllSubActvData( data, procDName ):

    for i in range( np.size(subjname,0) ):

        for j in range( np.size(actvname,0) ):

            proc1ActData( data, procDName, i, j )
```

```
        #endfor

    #endfor

#endfunction
```

[10]:
```
# interpolación para remuestreo de las señales
def signResamp( data, fr, tmax ):

    tnew = np.arange( 0, tmax, 1/fr )

    interpFunc = scpnt.interp1d( data[0,:], data[1:,:], kind='linear', axis=1,␣
 ↪copy=True, bounds_error=None, fill_value="extrapolate" )
    Daux = np.zeros( ( np.size( data[:,:],0 ), np.size( tnew,0 ) ) )

    Daux[0,:] = tnew
    Daux[1:,:] = interpFunc(tnew)

    return Daux

#endfunction


# tiempo final del remuestreo
def tmaxSamp( data ):

    iteraux = np.zeros( (np.size(measname,0)) )

    for i in range( np.size(measname,0) ):
        iteraux = np.max( data[ measname[i] ]['Data'][0,:] )

    #endfor

    return np.min( iteraux )


#endfuntion


# tiempo inicial del remuestreo
def tminSamp( data ):

    iteraux = np.zeros( (np.size(measname,0)) )

    for i in range( np.size(measname,0) ):
        iteraux = np.min( data[ measname[i] ]['Data'][0,:] )
```

```
    #endfor

    return np.max( iteraux )


#endfuntion


# data (toda) de una actividad remeustrada
def ActvReSampData( data, name, isubj, iactv ):

    tmax = tmaxSamp( callData2( data, isubj, iactv) )
    tmin = tminSamp( callData2( data, isubj, iactv) )

    dpaux0 = signResamp( callData1(data, isubj, iactv, 0)[name][:,:], 500, tmax )
    dpaux1 = signResamp( callData1(data, isubj, iactv, 1)[name][:,:], 500, tmax )
    dpaux2 = signResamp( callData1(data, isubj, iactv, 2)[name][:,:], 500, tmax )
    dpaux3 = signResamp( callData1(data, isubj, iactv, 3)[name][:,:], 500, tmax )
    dpaux4 = signResamp( callData1(data, isubj, iactv, 4)[name][:,:], 500, tmax )

    # promedio del consumo energético de los modelos Garby and Astrup 1987 y␣
 ↪Péronnet and Massicotte 1991
    EnerExp = ( (16.89 *dpaux4[2,:] + 4.84 *dpaux4[3,:])/1000 + (16.04␣
 ↪*dpaux4[2,:] + 4.94 *dpaux4[3,:])/1000 ) /2

    callData2( data, isubj, iactv )[name] = np.concatenate( ( [dpaux0[0,:]],␣
 ↪[EnerExp], dpaux0[2:,:], dpaux1[2:,:], dpaux2[2:,:], dpaux3[2:,:], dpaux4[2:,:
 ↪] ), axis=0 )

    del dpaux0, dpaux1, dpaux2, dpaux3, dpaux4
    #return EnerExp

#endfunction


def AllActvReSampData( data, name, isubj ):

    for i in range( np.size(actvname,0) ):

        ActvReSampData( data, name, isubj, i )

    #endfor

#endfunction
```

```
def AllSunjActvReSampData( data, name ):

    for i in range( np.size(subjname,0) ):
        #print( 'i = ', i)

        for j in range( np.size(actvname,0) ):
            #print('j = ', j)

            ActvReSampData( data, name, i, j )

        #endfor

    #endfor

#endfunction
```

[11]:
```
def delAllMeasData( data, procDName, isubj, iactv ):

    for i in range( np.size(measname,0) ):
        #callData1( data, 0, 0, i ).create_group("dataprcs")
        del callData1( data, isubj, iactv, i )[procDName]
        #callData1( data, isubj, iactv, i )[procDName] = filtradoSeñal(
↪callData(data, isubj, iactv, i, 0 )[:,:], callData(data, isubj, iactv, i, 2
↪)[0][0], i )

    #endfor

#endfunction


def delAllActvMeasData( data, procDName, isubj ):

    for i in range( np.size(actvname,0) ):

        del1ActData( data, procDName, isubj, i )

    #endfor

#endfunction


def delAllSubActvMeasData( data, procDName ):

    for i in range( np.size(subjname,0) ):

        for j in range( np.size(actvname,0) ):
```

```
            del1ActData( data, procDName, i, j )

        #endfor

    #endfor

#endfunction


def delAct1Data( data, procDName, isubj, iactv ):

    #callData2( data, 0, 0 ).create_group("dataprcs")
    del callData2( data, isubj, iactv )[procDName]
    #callData2( data, isubj, iactv )[procDName] = filtradoSeñal( callData(data,␣
↪isubj, iactv, i, 0 )[:,:], callData(data, isubj, iactv, i, 2 )[0][0], i )

    #endfor

#endfunction


def delAllActvMeasData( data, procDName, isubj ):

    for i in range( np.size(actvname,0) ):

        #callData2( data, i, 0 ).create_group("dataprcs")
        del callData2( data, isubj, i )[procDName]
        #callData2( data, i, iactv )[procDName] = filtradoSeñal( callData(data,␣
↪isubj, iactv, i, 0 )[:,:], callData(data, isubj, iactv, i, 2 )[0][0], i )

    #endfor

#endfunction


def delAllSubjAllAct1Data( data, procDName ):

    for i in range( np.size(subjname,0) ):

        for j in range( np.size(actvname,0) ):

            #callData2( data, i, 0 ).create_group("dataprcs")
            del callData2( data, i, j )[procDName]
            #callData2( data, i, iactv )[procDName] = filtradoSeñal(␣
↪callData(data, isubj, iactv, i, 0 )[:,:], callData(data, isubj, iactv, i, 2␣
↪)[0][0], i )
```

```
        #endfor

    #endfor

#endfunction
```

[ ]:

[12]:
```
def signalplot( xdata, ydata, title, figsize=(10,4) ):

    figure = plt.fig( figsize = figsize )

    plt.plot( xdata, ydata )
    plt.xlabel('time [s]')
    plt.ylabel('signal')
    plt.title(title)

#endfunction
```

[13]:
```
# función para calcular el Residual Sum of Squares (RSS)

def RSS(y_real, y_calc):
    return np.sum ( ( y_real - y_calc ) * ( y_real - y_calc ) )

#endfunction


# R^2 del entrenamiento

def R2entPrint( R2ent, dataTitle ):

    if np.sum(R2ent) is not None:
        print(dataTitle + ': R^2 de cada iteración con los de entrenamiento:')
        print(R2ent)
        print('\n')

#endfunction


# RSS del entrenamiento

def RSSentPrint( RSSent, dataTitle ):

    if np.sum(RSSent) is not None:
        print(dataTitle + ': RSS de cada iteración con los datos de␣
 ↪entrenamiento:')
        print(RSSent)
```

```
        print('\n')

#endfunction


# R^2 de la evaluación

def R2evaPrint( R2eva, dataTitle ):

    if np.sum(R2eva) is not None:
        print(dataTitle + ': R^2 de cada iteración con los datos de prueba:')
        print(R2eva)
        print('\n')

#endfunction


# R^2 de la evaluación

def RSSevaPrint( RSSeva, dataTitle ):

    if np.sum(RSSeva) is not None:
        print(dataTitle + ': RSS de cada iteración con los datos de prueba:')
        print(RSSeva)
        print('\n')

#endfunction


# R^2 usando la función cross_val_score

def R2evaCVSPrint( R2CValScore, dataTitle ):

    if np.sum(R2CValScore) is not None:
        print('error del modelo usando el comando cross_val_score:')
        print(R2CValScore)
        print('\n')

#endfunction


# presentar todos los errores resultados del modelo

def modelErrorPrint( dataTitle=None , RSSent=None , RSSeva=None , R2ent=None ,␣
 ↪R2eva=None , R2CValScore=None ):

    # formato de los errores
```

```
    np.set_printoptions(formatter={'float': lambda x: "{0:0.5f}".format(x)},␣
↪suppress = True)

    # errores de interés obtenidos
    RSSentPrint( RSSent, dataTitle )

    RSSevaPrint( RSSeva, dataTitle )

    R2entPrint( R2ent, dataTitle )

    R2evaPrint( R2eva, dataTitle )

    R2evaCVSPrint( R2CValScore, dataTitle )

#endfunction
```

[14]:
```
# Linear Regression & data-split

def LinearRegrDSplit( data, trdSplit, varShuffle = False, randomState = None,␣
↪stratifyData = None ):

    # Se instancian los data set de entrenamiento y evaluación
    Xtrain, Xtest, Ytrain, Ytest = sklms.train_test_split( data[:,1:], data[:
↪,0], stratify = stratifyData, random_state = randomState, train_size =␣
↪trdSplit )

    # Se instancia el modelo
    model = skllm.LinearRegression()

    # Variables de interés a almacenar
    #RSSent = np.zeros((1))
    #RSSeva = np.zeros((1))
    #R2ent = np.zeros((1))
    #R2eva = np.zeros((1))

    # Se entrena el modelo
    model.fit( Xtrain, Ytrain )

    # Se prueba el modelo (datos de entrenamiento: resultados del modelo)
    modelOutTrain = model.predict(Xtrain)

    # Evaluación de la predicción (datos de entrenamiento)
    RSSent = RSS( Ytrain , modelOutTrain )
    R2ent = model.score(Xtrain, Ytrain)     # Root Mean Square

    # Se prueba el modelo (datos de prueba: resultados del modelo entrenado)
    modelOutEval = model.predict(Xtest)
```

```
    # Evaluación de la predicción (datos de prueba)
    RSSeva = RSS( Ytest , modelOutEval )
    R2eva = model.score( Xtest, Ytest )      # Root Mean Square

    return RSSent , RSSeva , R2ent , R2eva , model

#endfunction
```

[15]:
```
def plot1Signal( xdata, ydata, title, xlabel='Time [s]', ylabel='signal',␣
↪figsize=(15,5), color='b' ):

    fig = plt.figure(figsize = figsize)

    plt.plot( xdata, ydata, c=color )

    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    #plot
    plt.show()

#endfunction


def plotAllSignal( data, title, xlabel=None, ylabel=None, figsize=(15,5),␣
↪color='b' ):


    if xlabel == None:
        xlabel = ['Time [s]']*np.size(data,0)

    if ylabel == None:
        ylabel = ['signal']*np.size(data,0)


    for i in range( np.size(data,0)-2 ):

        if  np.size( np.where( np.isnan(data[i+2,:]) ) ) == 0:

            plot1Signal( xdata=data[0,:], ydata=data[i+2,:], title=title[i],␣
↪xlabel=xlabel[i], ylabel=ylabel[i], figsize=figsize, color=color )

        #endif

    #endfor
```

```
#endfuntion


def plotAllMeasSingal( data, measname, title='', figsize=(15,5) ):

    color= ['r','g','b','m','c','y']

    for i in range( np.size(measname) ):

        ftitle = np.char.add( np.array([ title + measname[i] +  ' - signal ']*(␣
↪np.size( data[ measname[i] ][ 'Data' ],0 ) -2 )).astype(np.str_), (np.arange(␣
↪np.size( data[ measname[i] ][ 'Data' ],0 ) -2) +1).astype(np.str_))

        plotAllSignal( data[ measname[i] ][ 'Data' ][:,:], ftitle,␣
↪figsize=figsize, color=color[i] )

    #endfor

#endfunction
```

[123]:
```
allResults = np.array( [ mR2ent5K, mR2eva5K, mR2all5K, mR2ent10K, mR2eva10K,␣
↪mR2all10K ] )

def LRallTables( allResults ):

    table = np.empty( np.size(allResults,0), dtype=object )

    for j in range( np.size(allResults,0) ):

        table[j] = PrettyTable()
        table[j].field_names = np.concatenate(([ "subjects"], actvname, ["mean"]␣
↪+ ["SD"]), axis=0)

        for i in range( np.size(subjname,0) ):

            table[j].add_row(  np.concatenate( ([subjname[i]], np.
↪round(allResults[j,i,:],3), np.round([np.mean(allResults[j,i,:])],3), np.
↪round([np.std(allResults[j,i,:])],3) ), axis=0) )

        #endfor

        table[j].add_row( np.concatenate( (['media'], [np.round(np.
↪mean(allResults[j,:,k]),3) for k in range( np.size(allResults,2) )], ["--"],␣
↪["--"] ), axis=0) )
```

```
        table[j].add_row( np.concatenate( (['STD'], [np.round(np.
↪std(allResults[j,:,k]),3) for k in range( np.size(allResults,2) )], ["--"],␣
↪["--"] ), axis=0) )

    #endfor

    return table

#end fuction
```

[94]:
```
# presentación de los resultados en tabla
def printResults( ResultTables, title, methodName ):


    for i in range( np.size(allResults,0) ):

        print(title + methodName[i])
        print(ResultTables[i])
        print()

    #endfor

#endfunction
```

[ ]:
```

```

Clases creadas

[ ]:
```

```

[22]:
```
# Clase que almacena los datos de interés del modelo
class ModelsInfo:

    def __init__(self, model_=None, RSSent_=None, RSSeva_=None, R2ent_=None,␣
↪R2eva_=None, Betas_=None, R2CValScore_=None, RSS_=None, R2_=None, Out_=None):

        self.model_ = model_
        self.RSSent_ = RSSent_
        self.RSSeva_ = RSSeva_
        self.R2ent_ = R2ent_
        self.R2eva_ = R2eva_
        self.Betas_ = Betas_
        self.R2CValScore_ = R2CValScore_
```

```
        self.RSS_ = RSS_
        self.R2_ = R2_
        self.Out_ = Out_

    #endfunction

#endclass
```

[23]:
```
# El objetivo de esta clase es añadirle atributos adicionales a los arreglos␣
↪numpy de relevancia para el estudio
# como lo son las variables estadísticas de interés.

class StatsArr(np.ndarray):

    def __new__(cls, input_array, mean_=None, median_=None, mode_=None,␣
↪max_=None, min_=None, range_=None, desv_=None,
                skew_=None, kurt_=None, dataCorr_=None, Q1_=None, Q2_=None,␣
↪Q3_=None, IQR_=None, TransfType_=None,
                Lambdas_=None):
        # Input array is an already formed ndarray instance
        # We first cast to be our class type
        obj = np.asarray(input_array).view(cls)
        # add the new attribute to the created instance
        obj.mean_ = mean_
        obj.median_ = median_
        obj.mode_ = mode_
        obj.max_ = max_
        obj.min_ = min_
        obj.range_ = range_
        obj.desv_ = desv_
        obj.skew_ = skew_
        obj.kurt_ = kurt_
        obj.dataCorr_ = dataCorr_
        obj.Q1_ = Q1_
        obj.Q2_ = Q2_
        obj.Q3_ = Q3_
        obj.IQR_ = IQR_
        obj.Transftype_ = TransfType_
        obj.Lambdas_ = Lambdas_
        # Finally, we must return the newly created object:
        return obj

    #endfunction

    def __array_finalize__(self, obj):
        # see InfoArray.__array_finalize__ for comments
        if obj is None: return
```

```
        self.mean_   = getattr(obj, 'mean_', None)
        self.median_ = getattr(obj, 'median_', None)
        self.mode_   = getattr(obj, 'mode_', None)
        self.max_    = getattr(obj, 'max_', None)
        self.min_    = getattr(obj, 'min_', None)
        self.range_  = getattr(obj, 'range_', None)
        self.desv_   = getattr(obj, 'desv_', None)
        self.skew_   = getattr(obj, 'skew_', None)
        self.kurt_   = getattr(obj, 'kurt_', None)
        self.dataCorr_ = getattr(obj, 'dataCorr_', None)
        self.Q1_ = getattr(obj, 'Q1_', None)
        self.Q2_ = getattr(obj, 'Q2_', None)
        self.Q3_ = getattr(obj, 'Q3_', None)
        self.IQR_ = getattr(obj, 'IQR_', None)
        self.TransfType_ = getattr(obj, 'TransfType_', None)
        self.Lambdas_ = getattr(obj, 'Lambdas_', None)
        # We do not need to return anything

    #endfunction

#endclass
```

[ ]:

1 Información relevante de la base de datos

La data se encuentra estructurada por un conjunto de muestra de 10 sujetos, quienes fueron sometidos a 5 actividades de las siguientes: caminar, caminar hacia atrás, correr, caminar en plano inclinado, caminar en escaleras o en bicicleta estática. Cada sujeto posee estas actividades posee 5 grupos de señales clasificadas de acuerdo a su naturaleza y ubicación en el individuo de prueba. Por último, cada una de las señales de dicha clasificación posee 3 atributos: data, frecuencia de muestreo y nombres (ver Figura 1)

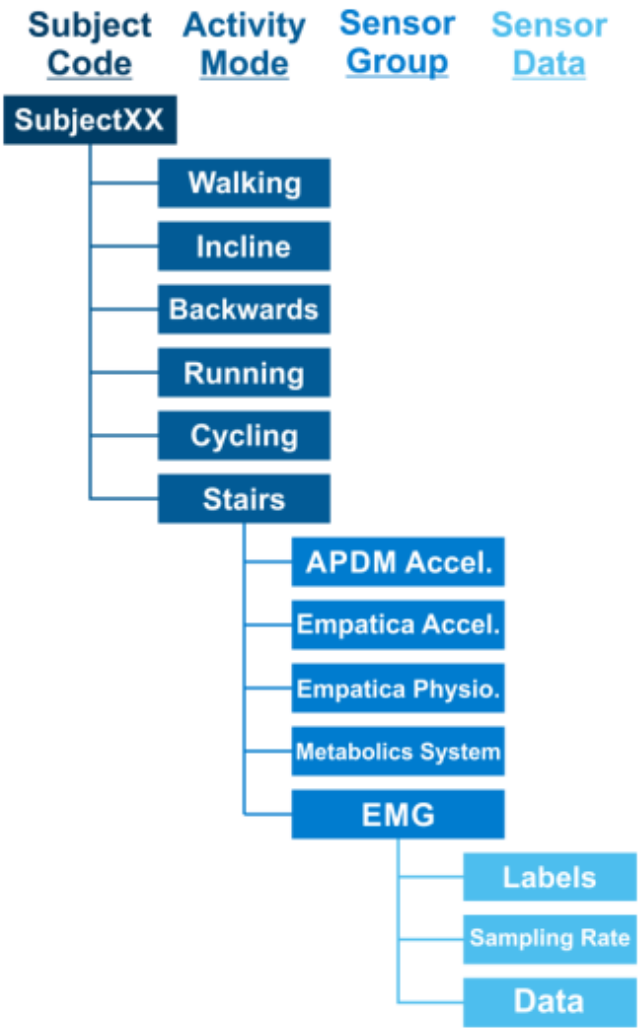

Figura 1: Estructura de la data [2]

Para cada uno de estos procesos se recolectó data de diferentes músculos en forma de señales EMG y en señales de aceleración, velocidad angular y campo magnético de acelerómetros de 3 ejes, en su mayoría ubicado en las piernas. también se midió la temperatura y actividad electrodermica de los individuos, así como otras señales metabólicas, como consumo de oxígeno, producción de dióxido de carbono, nivel de saturación de respiración, entre otros. Todas las señales se presentan en las Figuras 2 y 3.

Sensor Groups	Signals Contained in Data Matrix		
EMG (16 signals)	Left EMG (V) (8 signals)	Gluteus maximus	
		Rectus femoris	
		Vastus lateralis	
		Semitendinosis	
		Biceps femoris	
		Medial gastrocnemius	
		Soleus	
		Tibialis anterior	
	Right EMG (V) (8 signals)	Gluteus maximus	
		Rectus femoris	
		Vastus lateralis	
		Semitendinosis	
		Biceps femoris	
		Medial gastrocnemius	
		Soleus	
		Tibialis anterior	
APDM Accel (54 signals)	Waist (9 signals)	Acceleration (m/s^2)	x, y, z
		Angular Velocity (rad/s)	x, y, z
		Magnetic Field (uT)	x, y, z
	Chest (9 signals)	Acceleration (m/s^2)	x, y, z
		Angular Velocity (rad/s)	x, y, z
		Magnetic Field (uT)	x, y, z
	Left Ankle (9 signals)	Acceleration (m/s^2)	x, y, z
		Angular Velocity (rad/s)	x, y, z
		Magnetic Field (uT)	x, y, z
	Right Ankle (9 signals)	Acceleration (m/s^2)	x, y, z
		Angular Velocity (rad/s)	x, y, z
		Magnetic Field (uT)	x, y, z
	Left Foot (9 signals)	Acceleration (m/s^2)	x, y, z
		Angular Velocity (rad/s)	x, y, z
		Magnetic Field (uT)	x, y, z
	Right Foot (9 signals)	Acceleration (m/s^2)	x, y, z
		Angular Velocity (rad/s)	x, y, z
		Magnetic Field (uT)	x, y, z

Table 1: Signals Contained in each Sensor

Figura 2: Señales presentes en cada grupo - parte 1 [2]

Entre los puntos importantes a mencionar de la data está el hecho de que, si bien cada clasificación de señales posee su propia tasa de muestreo, cada una posee 2 señales adicionales: una que indica el paso temporal del registro de la señal y otra que simplemente indica la actividad que se realiza con un código. esta segunda señal adicional es despreciable y se descarta al momento de realizar el procesado de las señales. Por otro lado, la señal temporal en cada clasificación no comienza ni termina en el mismo instante de tiempo, lo que necesariamente obliga a usar la interpolación como método de re muestreo para asegurar que todas las señales procesadas mantengan un mismo vector de tiempo de igual espaciado.

Empatica Accel (6 signals)	Left Wrist (3 signals)	Acceleration (m/s^2)	x, y, z
	Right Wrist (3 signals)	Acceleration (m/s^2)	x, y, z
Empatica Physiological (4 signals)	Left Wrist (2 signals)	Electrodermal Activity (uS)	
		Skin Temperature (degC)	
	Right Wrist (2 signals)	Electrodermal Activity (uS)	
		Skin Temperature (degC)	
Metabolics System (7 signals)	VO2 (mL/s)		
	VCO2 (mL/s)		
	Respiratory Exchange Ratio (RER)		
	Breath Frequency (1/min)		
	Minute Ventiltation (L/min)		
	Oxygen Saturation SpO2 (%)		
	Heart Rate (1/min)		

Table 1: Signals Contained in each Sensor Group (Continued)

Figura 3: Señales presentes en cada grupo - parte 2 [2]

2 Procesamiento de la data

Para el procesamiento de las señales, se realizó un proceso similar a aplicado en la referencia [1]: básicamente el conjunto de señales EMG fue filtrado con un filtro pasa altos con frecuencia de corte de 30 y un filtro pasa bajos con frecuencia de corte de 350. El resto de las señales no fue sometida a ningún tipo de filtrado de este tipo.

Posteriormente se aplicó un filtro que suavizara la señal ante los efectos de ruido. En el caso de este trabajo, se implementó el filtro Savitzky-Golay con características adaptadas a cada señal de la siguiente manera:

- Señales APDL Accel: Se aplicó un filtro de orden 3 con una ventana de 10 segundo.
- Señales EMG: Se aplicó un filtro de orden 3 con una ventana de 1 segundo.
- Señales Epatical Accel: Se aplicó un filtro de orden 3 con una ventana de 10 segundo.
- Señales Epatical Psysio: Se aplicó un filtro de orden 3 con una ventana de 10 segundo.
- Señales Metabolics System: Se aplicó un filtro de orden 3 con una ventana de 1 segundo para todas las señales exceptuando la señal de nivel de saturación de oxígeno.

En [1] usan un filtro lineal gaussiano, pero, por factores de costo computacional, y para realizar otra prueba con otro tipo de filtro, se seleccionó el filtro Savitzky-Golay.

Posteriormente, debido a que las señales poseen diferentes tasas de muestreo e inician y finalizan en instantes de tiempo diferentes, se aplicó una interpolación linal a una frecuencia de muestreo específica, usando el instante de tiempo inicial mayor de todas las señales como instante de inicio y el instante final menor de todas las señales como punto final de la señal. Esto implica que, además de remuestrear las señales, también se truncó parte de la información al momento de realizar

el remuestreo. Esto se hizo con la finalidad de evitar la extrapolación (creación de datos que necesariamente introducen error) para un rango de tiempo menor a 5 segundo de señales que duran más de 10 minutos.

Todas las señales fueron remuestreadas a 500 Hz para este caso, se escogió 500 Hz como la tasa de muestreo para evitar perder información valiosa de las señales EMG, pero manteniendo un tamaño de data que, si bien sigue siendo grande, es manejable dentro de las capacidades computacionales disponibles.

[16]:
```
# lectura de la data
data = np.empty( np.size(filename,0), dtype=object)
for i in range(np.size(filename,0)): data[i] = h5py.File( filename[i], 'r+')
```

[17]:
```
#instanciación de los filtros para la señal EMG
IirHP30, IirLP350 = filtros(data, 0, 0, 1)
```

[18]:
```
# filtrado de toda la data
procAllSubActvData( data, "procsData1" )
```

[21]:
```
# remuestreado de la data
AllSunjActvReSampData( data, "procsData1" )
```

[20]:
```
#callData1( data, 1, 0, 0 ).keys()
#del callData2( data, 0, 0)["procsData1"]
#del callData2( data, 0, 1)["procsData1"]
#callData2( data, 0, 0)["procsData1"] = 0
#print( callData2( data, 0, 0 ).keys() )
#print( callData2( data, 0, 1 ).keys() )
#print( callData2( data, 0, 2 ).values() )
#callData1( data, 0, 1, 4)['Data']
```

3 Modelo Implementado

En esta experiencia, como primera iteración, se decidió implementar un modelo de regresión lineal basado en la metodología de [1], quienes implementaron ese tipo de modelo para estimar los valores de consumo energético.

En esta ocasión, solo se aplicará para cada individuo por separado a fin de mostrar la convergencia del método, con el fin de tener un primer resultado que certifique la validez del mismo.

Para comparar los resultados, en vista de tener directamente de los datos el consumo de oxígeno y producción de dióxido de carbono, pero no tener de forma explícita el valor del nitrógeno excretado, siguiendo los modelos que no dependen de este valor faltante según la Figura 4, la curva "teórica" de consumo de energía se obtuvo del promediar el modelo Garby and Astrup 1987 y el modelo de Péronnet and Massicotte 1991.

Table 1. Published equations for calculating the energetic expenditure, where $\dot{V}O_2$ and $\dot{V}CO_2$ is in L·s^{-1}, and N is in g.

	Equation
Lusk 1924[a]	$16.00\,\dot{V}O_2 + 5.15\,\dot{V}CO_2 - 7.80N$
Weir 1949	$16.50\,\dot{V}O_2 + 4.62\,\dot{V}CO_2 - 9.06N$
Weir[b] (corrected)	$16.62\,\dot{V}O_2 + 4.51\,\dot{V}CO_2 - 9.22N$
Garby and Astrup 1987	$16.04\,\dot{V}O_2 + 4.94\,\dot{V}CO_2$
Ferrannini 1988 (glucose)	$16.15\,\dot{V}O_2 + 4.82\,\dot{V}CO_2 - 4.79N$
Ferrannini 1988 (glycogen)	$16.38\,\dot{V}O_2 + 4.64\,\dot{V}CO_2 - 4.51N$
Brockway 1987	$16.58\,\dot{V}O_2 + 4.51\,\dot{V}CO_2 - 5.90N$
Péronnet and Massicotte 1991	$16.89\,\dot{V}O_2 + 4.84\,\dot{V}CO_2$
Jeukendrup and Wallis 2005 (intensities between 40%–50% $\dot{V}O_{2max}$)	$18.56\,\dot{V}O_2 + 2.40\,\dot{V}CO_2 - 4.14N$
Jeukendrup and Wallis 2005 (intensities between 50%–75% $\dot{V}O_{2max}$)	$18.71\,\dot{V}O_2 + 2.30\,\dot{V}CO_2 - 4.14N$

Note: N, nitrogen; $\dot{V}CO_2$, carbon dioxide production; $\dot{V}O_2$, oxygen uptake; $\dot{V}O_{2max}$, maximal oxygen uptake.
[a]As calculated by Abramson (1943).
[b]Weir's equation as corrected in Brockway (1987).

Figura 4: Tabla de ecuaciones calorímetras [2]

Dichos valores fueron calculados al momento de remuestrear la data, aprovechando que se construye una nueva matriz con todas las señales de interés respecto al vector de tiempo correspondiente de referencia.

Una vez se tienen todas las señales remuestradas a la misma tasa de muestreo y se ha calculado la referencia, se procede a implementar el modelo. En este caso, al no realizar ningún tipo de clasificación, solamente se dispone de valores de error como el ERMS para validar el resultado obtenido. En esta primera experiencia, se escoge aplicar un modelo de validación cruzada de 5 iteraciones y otro de 10 iteraciones para comparar resultados.

[30]:
```
# Instancias de las variables de interés para el modelo de regresión lineal con␣
 ↪validación cruzada de 5 iteraciones
mR2ent5K = np.zeros( (np.size(subjname,0), np.size(actvname,0)) )
mR2eva5K = np.zeros( (np.size(subjname,0), np.size(actvname,0)) )
mR2all5K = np.zeros( (np.size(subjname,0), np.size(actvname,0)) )
LRmodel5K = np.empty( (np.size(subjname,0), np.size(actvname,0)), dtype = object␣
 ↪)


# Instancias de las variables de interés para el modelo de regresión lineal con␣
 ↪validación cruzada de 5 iteraciones
mR2ent10K = np.zeros( (np.size(subjname,0), np.size(actvname,0)) )
mR2eva10K = np.zeros( (np.size(subjname,0), np.size(actvname,0)) )
mR2all10K = np.zeros( (np.size(subjname,0), np.size(actvname,0)) )
LRmodel10K = np.empty( (np.size(subjname,0), np.size(actvname,0)), dtype =␣
 ↪object )


# ejecución del modelo para todas las actividades de todos los sujetos de pruebas
for j in range( np.size( subjname,0 ) ):
```

```
   for i in range( np.size( actvname,0 ) ):

       #regresión lineal con validación cruzada de 5 iteraciones
       LRmodel5K[j,i] = ModelsInfo()
       LRmodel5K[j,i].RSSent_ , LRmodel5K[j,i].RSSeva_ , LRmodel5K[j,i].R2ent_␣
↪, LRmodel5K[j,i].R2eva_ , LRmodel5K[j,i].model_ = LinearRegrDSplit( data =␣
↪callData2(data, j, i)["procsData1"][1:,:].transpose(), trdSplit = 5 )
       mR2ent5K[j,i] = LRmodel5K[j,i].R2ent_
       mR2eva5K[j,i] = LRmodel5K[j,i].R2eva_
       mR2all5K[j,i] = LRmodel5K[j,i].model_.score(callData2(data, j,␣
↪i)["procsData1"][2:,:].transpose(), callData2(data, j, i)["procsData1"][1,:].
↪transpose())


       #regresión lineal con validación cruzada de 10 iteraciones
       LRmodel10K[j,i] = ModelsInfo()
       LRmodel10K[j,i].RSSent_ , LRmodel10K[j,i].RSSeva_ , LRmodel10K[j,i].
↪R2ent_ , LRmodel10K[j,i].R2eva_ , LRmodel10K[j,i].model_ = LinearRegrDSplit(␣
↪data = callData2(data, j, i)["procsData1"][1:,:].transpose(), trdSplit = 10 )
       mR2ent10K[j,i] = LRmodel10K[j,i].R2ent_
       mR2eva10K[j,i] = LRmodel10K[j,i].R2eva_
       mR2all10K[j,i] = LRmodel10K[j,i].model_.score(callData2(data, j,␣
↪i)["procsData1"][2:,:].transpose(), callData2(data, j, i)["procsData1"][1,:].
↪transpose())

   #endfor

#endfor
```

[ ]:

4 Resultados

A continuación, se presentan los resultados obtenidos del modelo de regresión lineal con validación cruzada de 5 y 10 iteraciones en tablas específicas donde se muestra el valor correspondiente de cada sujeto en cada actividad realizada y su correspondiente media y desviación estándar.

[124]:
```
ResultTables = LRallTables( allResults )
printResults( ResultTables, 'Regresión Lineal. ERMS para ', methodName )
```

```
Regresión Lineal. ERMS para 5k entrenamiento
+-----------+-----------+---------+---------+---------+---------+------+-----+
|  subjects | Backwards | Cycling | Incline | Running | Walking | mean |  SD |
+-----------+-----------+---------+---------+---------+---------+------+-----+
| Subject01 |    1.0    |   1.0   |   1.0   |   1.0   |   1.0   | 1.0  | 0.0 |
| Subject02 |    1.0    |   1.0   |   1.0   |   1.0   |   1.0   | 1.0  | 0.0 |
| Subject03 |    1.0    |   1.0   |   1.0   |   1.0   |   1.0   | 1.0  | 0.0 |
| Subject04 |    1.0    |   1.0   |   1.0   |   1.0   |   1.0   | 1.0  | 0.0 |
| Subject05 |    1.0    |   1.0   |   1.0   |   1.0   |   1.0   | 1.0  | 0.0 |
| Subject06 |    1.0    |   1.0   |   1.0   |   1.0   |   1.0   | 1.0  | 0.0 |
| Subject07 |    1.0    |   1.0   |   1.0   |   1.0   |   1.0   | 1.0  | 0.0 |
| Subject08 |    1.0    |   1.0   |   1.0   |   1.0   |   1.0   | 1.0  | 0.0 |
| Subject09 |    1.0    |   1.0   |   1.0   |   1.0   |   1.0   | 1.0  | 0.0 |
| Subject10 |    1.0    |   1.0   |   1.0   |   1.0   |   1.0   | 1.0  | 0.0 |
|   media   |    1.0    |   1.0   |   1.0   |   1.0   |   1.0   |  --  |  -- |
|    STD    |    0.0    |   0.0   |   0.0   |   0.0   |   0.0   |  --  |  -- |
+-----------+-----------+---------+---------+---------+---------+------+-----+

Regresión Lineal. ERMS para 5k evaluación
+-----------+-----------+---------+---------+---------+---------+-------+-------
+
|  subjects | Backwards | Cycling | Incline | Running | Walking |  mean |   SD
|
+-----------+-----------+---------+---------+---------+---------+-------+-------
+
| Subject01 |   0.975   |  0.881  |  0.987  |  0.991  |  0.952  | 0.957 |  0.04
|
| Subject02 |   0.988   |  0.932  |  0.901  |  0.924  |  0.861  | 0.921 | 0.042
|
| Subject03 |   0.936   |  0.976  |  0.989  |  0.988  |  0.734  | 0.925 | 0.097
|
| Subject04 |    0.95   |  0.019  |  0.988  |  0.958  |   0.97  | 0.777 | 0.379
|
| Subject05 |    0.98   |  0.021  |  0.884  |  0.931  |  0.298  | 0.622 | 0.389
|
| Subject06 |   0.942   |  0.989  |  0.938  |  0.974  |  0.898  | 0.948 | 0.032
|
| Subject07 |    0.58   |  0.346  |  0.904  |  0.974  |  0.697  |  0.7  | 0.227
|
```

Subject08	-0.696	0.929	0.935	0.994	0.896	0.612	0.654
Subject09	0.406	0.664	0.944	0.978	0.411	0.681	0.248
Subject10	0.969	0.974	0.958	0.991	0.67	0.912	0.122
media	0.703	0.673	0.943	0.97	0.739	--	--
STD	0.503	0.377	0.036	0.024	0.217	--	--

Regresión Lineal. ERMS para 5k toda la data

subjects	Backwards	Cycling	Incline	Running	Walking	mean	SD
Subject01	0.975	0.881	0.987	0.991	0.952	0.957	0.04
Subject02	0.988	0.932	0.901	0.924	0.861	0.921	0.042
Subject03	0.936	0.976	0.989	0.988	0.734	0.925	0.097
Subject04	0.95	0.019	0.988	0.958	0.97	0.777	0.379
Subject05	0.98	0.021	0.884	0.931	0.298	0.622	0.389
Subject06	0.942	0.989	0.938	0.974	0.898	0.948	0.032
Subject07	0.58	0.346	0.904	0.974	0.697	0.7	0.227
Subject08	-0.696	0.929	0.935	0.994	0.896	0.612	0.654
Subject09	0.406	0.664	0.944	0.978	0.411	0.681	0.248
Subject10	0.969	0.974	0.958	0.991	0.67	0.912	0.122
media	0.703	0.673	0.943	0.97	0.739	--	--
STD	0.503	0.377	0.036	0.024	0.217	--	--

Regresión Lineal. ERMS para 10k entrenamiento

subjects	Backwards	Cycling	Incline	Running	Walking	mean	SD
Subject01	1.0	1.0	1.0	1.0	1.0	1.0	0.0
Subject02	1.0	1.0	1.0	1.0	1.0	1.0	0.0
Subject03	1.0	1.0	1.0	1.0	1.0	1.0	0.0
Subject04	1.0	1.0	1.0	1.0	1.0	1.0	0.0
Subject05	1.0	1.0	1.0	1.0	1.0	1.0	0.0
Subject06	1.0	1.0	1.0	1.0	1.0	1.0	0.0
Subject07	1.0	1.0	1.0	1.0	1.0	1.0	0.0
Subject08	1.0	1.0	1.0	1.0	1.0	1.0	0.0
Subject09	1.0	1.0	1.0	1.0	1.0	1.0	0.0
Subject10	1.0	1.0	1.0	1.0	1.0	1.0	0.0
media	1.0	1.0	1.0	1.0	1.0	--	--
STD	0.0	0.0	0.0	0.0	0.0	--	--

Regresión Lineal. ERMS para 10k evaluación

subjects	Backwards	Cycling	Incline	Running	Walking	mean	SD
Subject01	0.99	0.993	0.992	0.998	0.995	0.994	0.003
Subject02	0.993	0.998	0.996	0.974	0.925	0.977	0.027
Subject03	0.985	0.989	0.996	0.985	0.988	0.989	0.004
Subject04	0.955	0.925	0.992	0.997	0.991	0.972	0.028
Subject05	0.996	0.99	0.994	0.999	0.993	0.994	0.003
Subject06	0.972	0.991	0.974	0.891	0.976	0.961	0.036
Subject07	0.976	0.957	0.982	0.914	0.978	0.961	0.025
Subject08	0.947	0.991	0.989	0.997	0.977	0.98	0.018
Subject09	0.987	0.933	0.984	0.996	0.991	0.978	0.023
Subject10	0.954	0.98	0.992	0.983	0.929	0.968	0.023
media	0.976	0.975	0.989	0.973	0.974	--	--
STD	0.017	0.025	0.007	0.037	0.024	--	--

+-----------+-----------+---------+---------+---------+---------+-------+-------
+

Regresión Lineal. ERMS para 10k toda la data

+-----------+-----------+---------+---------+---------+---------+-------+-------
+

| subjects | Backwards | Cycling | Incline | Running | Walking | mean | SD |

+-----------+-----------+---------+---------+---------+---------+-------+-------
+

| Subject01 | 0.99 | 0.993 | 0.992 | 0.998 | 0.995 | 0.994 | 0.003 |

| Subject02 | 0.993 | 0.998 | 0.996 | 0.974 | 0.925 | 0.977 | 0.027 |

| Subject03 | 0.985 | 0.989 | 0.996 | 0.985 | 0.988 | 0.989 | 0.004 |

| Subject04 | 0.955 | 0.925 | 0.992 | 0.997 | 0.991 | 0.972 | 0.028 |

| Subject05 | 0.996 | 0.99 | 0.994 | 0.999 | 0.993 | 0.994 | 0.003 |

| Subject06 | 0.972 | 0.991 | 0.974 | 0.891 | 0.976 | 0.961 | 0.036 |

| Subject07 | 0.976 | 0.957 | 0.982 | 0.914 | 0.978 | 0.961 | 0.025 |

| Subject08 | 0.947 | 0.991 | 0.989 | 0.997 | 0.977 | 0.98 | 0.018 |

| Subject09 | 0.987 | 0.933 | 0.984 | 0.996 | 0.991 | 0.978 | 0.023 |

| Subject10 | 0.954 | 0.98 | 0.992 | 0.983 | 0.929 | 0.968 | 0.023 |

| media | 0.976 | 0.975 | 0.989 | 0.973 | 0.974 | -- | -- |

| STD | 0.017 | 0.025 | 0.007 | 0.037 | 0.024 | -- | -- |

+-----------+-----------+---------+---------+---------+---------+-------+-------
+

Es impoEs importante mencionar que los valores del error cuadrático medio fueron directamente calculados de la función score ofrecida como método del modelo una vez instanciado y entrenado.

De acá se puede apreciar que la regresión lineal con validación cruzada de 10 iteraciones refleja una mejora sustancial sobre el mismo método con 5 iteraciones. En ambos casos, ambos obtienen puntajes perfectos al momento de ser entrenados, lo que puede deberse a la gran cantidad de datos que posee cada muestra de cada actividad y a que la señal se comporta como una señal continua, lo que pudiera facilitar la predicción.

Sin embargo, al ejecutar el modelo entrenado contra los datos de evaluación, claramente el modelo con solo 5 iteraciones tiene problemas para predecir varios valores, resultando en baja precisión en

los valores en diferentes actividades de un individuo (que se refleja en la media), causando grandes desviaciones en los resultados. Los resultados no mejoran si se evalúa la media y desviación estándar por sujeto en vez de actividad, ya que existen casos para algunos individuos en todas las actividades donde el modelo no presenta un desempeño apropiado. La discrepancia entre los resultados puede deberse a la gran cantidad de datos que hace insuficiente realizar una validación cruzada de 5 iteraciones, pues el modelo no logra predecir apropiadamente el comportamiento en varios casos (casos que en conjunto incluyen al menos dos casos para cada actividad, causando la disminución del desempeño total tanto en individuos para una actividad como en actividades de un individuo). Cabe destacar que todos son modelos independientes únicamente entrenados con la data de una actividad asociada a un individuo.

Por otro lado, el modelo con validación cruzada de 10 iteraciones muestra ser lo suficientemente preciso y robusto en todos los casos, manteniendo medias por encima del 96.5% para las actividades asociadas a un individuo y para los resultados de todos los individuos en una actividad, proporcionando una desviación estándar del orden de 10^{-2}, lo que muestra la constancia de esta configuración, por lo que es el conjunto de modelos resultantes apropiados para este problema. la constancia de esta configuración, por lo que es el conjunto de modelos resultantes apropiados para este problema.

Resulta importante destacar que, para las pruebas realizadas, los algoritmos ejecutados tardaron alrededor de 35min en solo realizar el filtrado y el remuestreo, lo que señala lo relevante del tamaño de la data al operarla. Adicionalmente, al operar la data y almacenar los valores procesados, el consumo de almacenamiento se incrementó considerablemente, pasando de 8GB a más de 40GB. Esto ocurre porque el filtrado no reduce el tamaño de la data (salvo por la eliminación de señales NaN), lo que se considera como casi duplicar la data; mientras que el remuestrear las señales a 500Hz converge a un aumento sustancial del tamaño de la data para todas las señales, excepto la EMG, que reduce su tamaño a la mitad. Como exiten señales muestreadas a 128Hz, 32Hz y menos de 1Hz, este remuestreo equivale a multiplicar exponencialmente la data, por lo que solo este proceso contribuye consumir más de 30GB de almacenamiento.

Estos factores de requisitos computacionales resultan esenciales para futuras pruebas, a modo de obtener una planificación que permita el uso eficiente de los recursos computacionales presentes.

5 Conclusiones

Del presente trabajo, se pudo demostrar la validez de la aplicación del modelo de regresión lineal, modelo implementado [1]. Sin embargo, cualquier configuración no es lo suficientemente buena como para ser apropiada. Para el presente caso, al trabajar con más de 60 señales de longitudes superiores a las 880000 muestras, resulta muy complicado para los modelos con validación cruzada con bajas cantidades de iteraciones conseguir buenos resultados en alguno de los casos.

Adicionalmente, se pudo tener una compresión del significado de la data y lo que se requiere para trabajar con una data de 8GB. Las limitaciones computacionales, si bien no fueron relevantes en este proceso, salvo por requerimientos de tiempo, el realizar procesamiento de la data de diferentes formas para su posterior almacenamiento conlleva a requerir grandes cantidades de espacio de almacenamiento para poder ser trabajada, lo que puede ser un elemento a tomar en cuenta al realizar diferentes pruebas. Para este proyecto, se consumieron más de 40GB y sólo se realizó un filtrado, suavizado y remuestreo de las señales.

El proyecto se concluye con un resultado satisfactorio, ya sea solo para los valores de un individuo, por alcanzar una precisión superior al 96% en todos los casos para el modelo de regresión lineal con validación cruzada de 10 iteraciones.

6 Siguientes pasos

- De esta primera iteración, se valida la viabilidad del modelo de regresión lineal. No obstante, resulta indispensable realizar pruebas con diferentes modelos que permitan corroborar comportamientos
- puede ser necesario aplicar modelos no supervisados en búsqueda de patrones que permitan tener una mayor compresión de la influencia que refleja cada señal respecto al consumo de energía de la persona y, posiblemente, desarrollar un modelo matemático asociado a las mismas.
- Es prudente dar respuesta a las siguientes preguntas:
  - ¿qué modelos de estimación de costo energético son más precisos?
  - ¿qué elementos pueden ser estandarizar para el modelo?
  - ¿existen patrones de señales que permitan tener mejores resultados?
  - ¿combinar señales (EMG, Aceleración, velocidad y campo magnético del acelerómetro) como una señal de magnitud resultante podrían ofrecer una mejor ventaja?
  - ¿cómo parametrizar las variables para generar modelos que pudieran usar las 10 muestras?
  - No se tienen datos esenciales de cada paciente en específico: peso, edad, altura y sexo. ¿cuánto afecta en el proceso la carencia de los datos?
- Planificar las diferentes pruebas y procesamientos para diferentes modelos puede ser crucial para maximizar el aprovechamiento de los recursos computacionales. Es importante encontrar una forma de poder realizar todas las pruebas necesarias sin requerir capacidad de cómputo extra para las primeras etapas.

7 Referencias Bibliográficas

[1] Ingraham, Kimberly A.; Ferris, Daniel P.; Remy, C. David (2019). Evaluating physiological signal salience for estimating metabolic energy cost from wearable sensors. Link: figshare dataset

[2] Ingraham, Kimberly A.; Ferris, Daniel P.; Remy, C. David (2019). Predicting energy cost from wearable sensors: A dataset of energetic and physiological wearable sensor data from healthy individuals performing multiple physical activities. Journal of Applied Physiology, 126(3), 717–729. doi:10.1152/japplphysiol.00714.2018

[3] Kipp, Shalaya & Byrnes, William & Kram, Rodger. (2018). Calculating metabolic energy expenditure across a wide range of exercise intensities: The equation matters. Applied Physiology, Nutrition, and Metabolism. 43. 10.1139/apnm-2017-0781.

[4] Westenskow, D. R.; Schipke, C. A.; Raymond, J. L.; Saffle, J. R.; Becker, J. M.; Young, E. W.; Cutler, C. A. (1988). Calculation of Metabolic Expenditure and Substrate Utilization from Gas Exchange Measurements. Journal of Parenteral and Enteral Nutrition, 12(1), 20–24. doi:10.1177/014860718801200120