Instituto Politécnico Nacional

Centro de Investigación en Computación

Validación de sensores utilizando redes neuronales

Natalia Sánchez Patiño
Dra. Gina Gallegos García

22 de septiembre 2021

Índice

1. Clasificación de series de tiempo de reconocimiento de actividad humana

El reconocimiento de la actividad humana es el problema de clasificar secuencias de datos de acelerómetro registrados por arneses especializados o teléfonos inteligentes en movimientos conocidos bien definidos.

Los enfoques clásicos del problema implican la elaboración manual de características a partir de datos de series de tiempo basadas en ventanas de tamaño fijo y modelos de aprendizaje automático de entrenamiento, como conjuntos de árboles de decisión. La dificultad es que esta ingeniería de características requiere una gran experiencia en el campo.

Recientemente, se ha demostrado que los métodos de aprendizaje profundo, como las redes neuronales recurrentes como las LSTM y las variaciones que hacen uso de redes neuronales convolucionales unidimensionales o CNN, brindan resultados de vanguardia en tareas desafiantes de reconocimiento de actividades con pocos o ningún dato. ingeniería de características, en lugar de utilizar el aprendizaje de características en datos sin procesar.

2. Reconocimiento de actividad mediante el conjunto de datos de sensores MPU9250

Actualmente se realizaron pruebas con una cantidad muy limitada de datos, por lo que se deberán recopilar más datos con mayor número de personas y con más variedad de situaciones para poder lograr tener un sistema completamente confiable.

En el desarrollo de este trabajo se considera que los sensores que miden parámetros fisiológicos tienen incorporados acelerómetros y giróscopos 3D, es decir, que miden la aceleración lineal y la velocidad angular en los 3 ejes x, y y z.

Para esta base de datos se debe considerar que los sensores personales estén en varias partes del cuerpo conectados de forma inalámbrica entre sí. Las lecturas de las mediciones de los sensores deberán realizarse mientras las personas realizaban distintas tareas tales como: caminar, sentarse en una silla, preparar el desayuno en la cocina, abrir la puerta, sentar y acostarse.

Actualmente la frecuencia de muestro del acelerómetro y giroscopio fue de 20 muestras por segundo, cada una de estas muestras contiene una ventana de 150 mediciones.

2.1. Obtención de los datos para entrenamiento

El código utilizado para la extracción de las muestras se puede ver en el Anexo A.

Este código contiene distintas funciones para poder ver el comportamiento de los sensores en tiempo real, para ello se utilizan las funciones de animación en matplotlib. Dentro de una raspberry esto puede ser algo complicado de observar ya que no siempre se muestra la ventana corriendo fluidamente los datos extraídos. Para ello se crearon las otras 2 funciones con nombre save_and_plot_1 y save_and_plot_2. Estas funciones lo que hacen son extraer los datos del sensor guardarlos en archivos .txt, uno para cada eje del acelerómetro y para cada eje del giroscopio. En estos archivos se escriben 150 mediciones por fila él así las especifican como argumento de la función, de forma predeterminada se extraen 100 veces. aproximadamente esto equivale a obtener datos durante 6 minutos. Al final, estos datos también se grafican y se guardan como una imagen.

Para extraer los datos es necesario correr cada una de las funciones en distintas raspberries de forma simultánea. En la primera raspberry se corre la función save_and_plot_1 y en la segunda raspberry se corre la otra función save_and_plot_2, ambas con el mismo número como parámetro. Esto guardará de forma independiente de los datos este cada uno de los sensores durante el mismo tiempo y posteriormente esto se agregará a la base

de datos que se utilizará para el entrenamiento. Las funciones de save_and_plot también guardarán una imagen con las gráficas obtenidas de cada eje el acelerómetro y giroscopio de los sensores (figuras 1 y 2) para poder analizar de una forma más visual los datos que se recuperan en los archivos de texto.

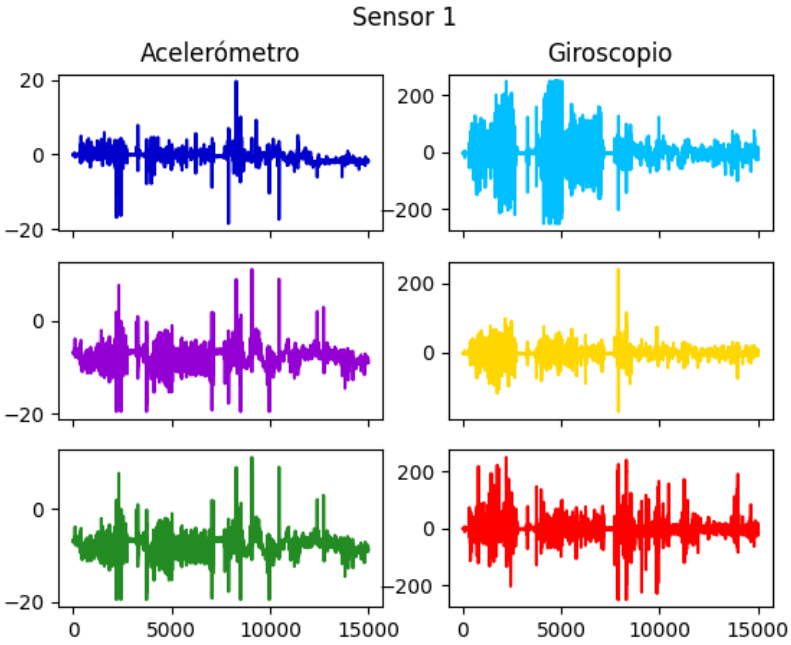

Figura 1: Datos del sensor 1 de manera gráfica

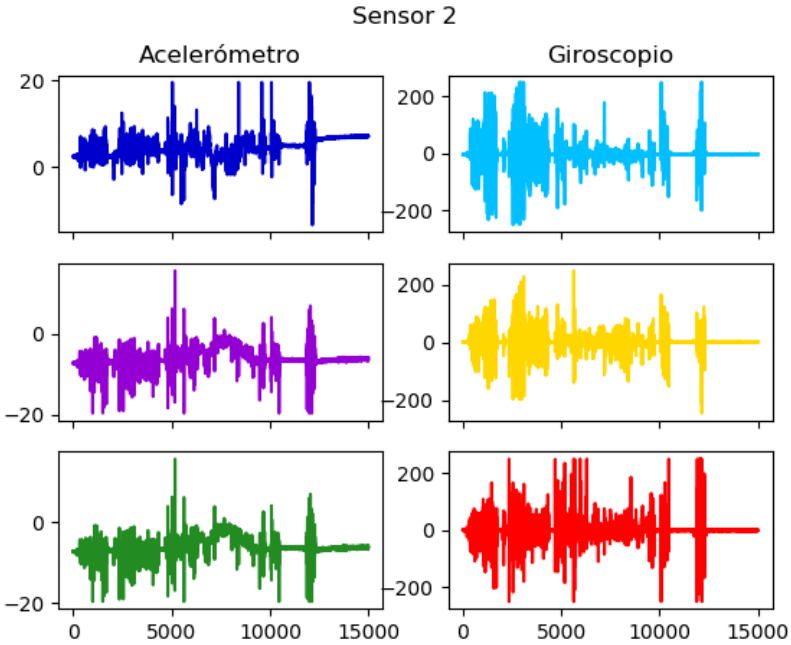

Figura 2: Datos del sensor 2 de manera gráfica

Para completar esta base de datos es necesario agregar el archivo te etiquetas que en la pequeña base de datos realizadas se llama y_train o y_test sea el caso.

Se hacen dos carpetas separadas, una de entrenamiento y otra de pruebas, para poder medir el rendimiento de la red. Los datos obtenidos de los sensores para las pruebas se dividieron en 70 para el entrenamiento y 30 para las pruebas.

3. Materiales

Se necesita la instalación de los siguientes paquetes:

- FaBo9Axis_MPU9250
- mpu6050
- Numpy
- Matplotlib
- Random
- Pandas
- Tensorflow
- Keras

Si existe algún problema con la instalación de tensorflow o keras, se deberá degradar el paquete h5py con el siguiente comando:

Listing 1: Cambio de versión del paquete h5py

```
1    $ pip install 'h5py==2.10.0' --force-reinstall
```

Estas librerías están especificadas para hacer la sustitución del código que se tenía anteriormente de extracción de características y cálculo para la validación de los sensores en la fase cero por una red neuronal que haga la misma validación de forma automática.

Para continuar con la integración de este código junto con estadio ya escrito para el intercambio de archivos cifrados en el sistema completo, se necesita también la otras librerías descritas en el reporte de funcionamiento del protocolo de estos sensores que se pueden encontrar en la página: https://github.com/KevinDelg/Protocolo-sensores. Para que este sistema completo funciones se deberán de instalar también las librerías especificadas en ese reporte.

Posteriormente en la sección 6 se detalla que fue luego sustituido del código anterior y en los anexos se puede encontrar la unificación ya de este código así como en el repositorio de https://github.com/Natalia-SP/Sensors_NN.

4. Desarrollo de la red neuronal

Los modelos de red LSTM son un tipo de red neuronal recurrente que puede aprender y recordar sobre largas secuencias de datos de entrada. Están pensados para su uso con datos que se componen de largas secuencias de datos, de hasta 200 a 400 pasos de tiempo. Pueden ser adecuados para este problema.

El modelo puede admitir múltiples secuencias paralelas de datos de entrada, como cada eje del acelerómetro y los datos del giroscopio. El modelo aprende a extraer características de secuencias de observaciones y cómo mapear las características internas a diferentes tipos de actividades.

El beneficio de usar LSTM para la clasificación de secuencias es que pueden aprender directamente de los datos de series de tiempo sin procesar y, a su vez, no requieren experiencia en el dominio para diseñar manualmente las características de entrada. El modelo puede aprender una representación interna de los datos de la serie temporal e idealmente lograr un rendimiento comparable al de los modelos que se ajustan a una versión del conjunto de datos con características diseñadas.

La arquitectura CNN LSTM implica el uso de capas de red neuronal convolucional (CNN) para la extracción de características en los datos de entrada combinados con LSTM para admitir la predicción de secuencias.

Los CNN LSTM se desarrollaron para problemas visuales de predicción de series de tiempo y la aplicación de generar descripciones textuales a partir de secuencias de imágenes (por ejemplo, videos). Específicamente, los problemas de:

- Reconocimiento de actividades: Generar una descripción textual de una actividad demostrada en una secuencia de imágenes.
- Descripción de imagen: Generación de una descripción textual de una sola imagen.
- Descripción de vídeo: Generación de una descripción textual de una secuencia de imágenes.

Una extensión adicional de la idea de CNN LSTM es realizar las convoluciones de la CNN (por ejemplo, cómo la CNN lee los datos de la secuencia de entrada) como parte de la LSTM. Esta combinación se llama Convolutional LSTM, o ConvLSTM para abreviar, y como CNN LSTM también se usa para datos espacio-temporales.

A diferencia de un LSTM que lee los datos directamente para calcular el estado interno y las transiciones de estado, y a diferencia del LSTM de CNN que interpreta la salida de los modelos de CNN, el ConvLSTM usa convoluciones directamente como parte de la entrada de lectura en las unidades LSTM mismas.

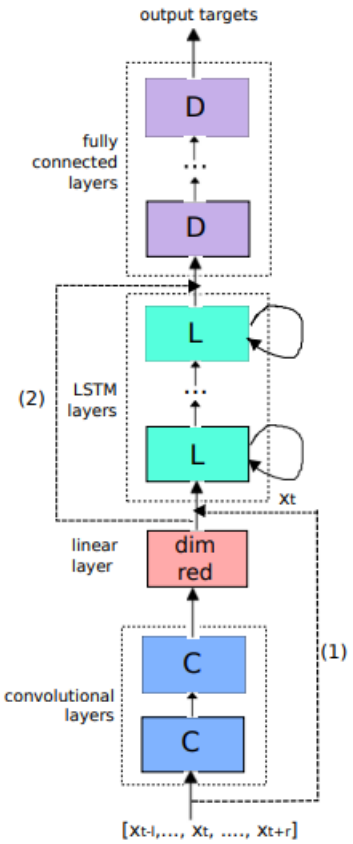

Figura 3: Arquitectura de la red convolucional con LSTM

5. Entrenamiento

El entrenamiento se relizó mediante el código que se puede ver en la sección B. Se utilizó la versión de tensorflow 2.1. Con otras versiones podría dar algunos problemas, si existiera algún problema de versiones a la hora de entrenar se puede utilizar el software en línea "Colab" de Google que permite correr código de Python en notebooks totalmente en línea. En este momento está utilizando la versión de tensorflow 2.6 e igualmente funciona sin problemas el código de entrenamiento dentro de este programa.

El código de entrenamiento buscará los archivos de la base de datos dentro de la misma carpeta o sitio donde se esté corriendo el programa. Buscará la carpeta con el nombre de SensorDataset. Y dentro de esta carpeta busca también las carpetas de entrenamiento con el nombre de train y de pruebas con el nombre de test. Dentro de cada una de estas carpetas encontrará el archivo de etiquetas y otra carpeta con el nombre de Inertial Signals que contendrá los 12 archivos de texto correspondientes a los 3 ejes de acelerómetros y giroscopio para los sensores 1 y 2.

Las funciones de load_dataset y load_dataset_group extraerá los datos de estos archivos guardándolos dentro de arreglos. Además, para las etiquetas, hará un encoding de tipo One Hot. Este tipo de encoding lo que hace es escribir a las etiquetas en función de arreglos. Por ejemplo, sí la etiqueta corresponde a ser aceptado, escribirá ún arreglo de la forma [1, 0]; mientras que si la etiqueta significa rechazado, escribirá el arreglo de la forma [0, 1]. Esto se hace para que las funciones de entrenamiento puedan leer adecuadamente las etiquetas.

El conjunto de etiquetas y los datos se guardan en variables de entrenamiento y prueba. Posteriormente estas variables entran en la función evaluate_model que es donde se encuentra la red neuronal como tal. En este caso es una red convolucional junto con una celda LSTM. Dentro de esta función se define este modelo utilizado y posteriormente se entrena con los datos de la variable trainX (datos de los sensores) y trainy que son las etiquetas para el entrenamiento.

Al finalizar el entrenamiento, se va a guardar el modelo y los pesos de la red neuronal en un archivo con el nombre convlstm_sensors más la fecha y hora en que fue obtenido, en formato h5.

El último paso dentro de esta función es obtener la precisión del modelo. Esto se hace evaluando los resultados obtenidos en los datos de prueba, que son las variables llamadas testX y testy.

Para este caso se decidió correr los entrenamientos 10 veces para saber cómo variaban la precisión, ya que no siempre se obtienen los mismo resultados, aunque se utilicen los mismos datos de entrenamiento. Estos pesos se guardan cada uno con diferente nombre y se puede elegir el que haya resultado con el mejor porcentaje de precisión.

En esta primera aproximación que se hizo con los pocos datos de entrenamiento que se obtuvieron, la precisión fue del 73 % con una varianza del 5 %. Aunque es un porcentaje algo bajo, es adecuado para la poca cantidad de datos que se tenían. Para poder aumentar esta precisión y además obtener un buen funcionamiento de este modelo en casos reales, es necesario aumentar en gran cantidad la base de datos con la que se va a entrenar.

Hasta ahora sólo fueron 10 minutos en total de entrenamiento aproximadamente y 2 minutos y medio para las pruebas aparte de que fueron realizadas únicamente por una persona y la variedad de actividades y movimientos no fue demasiada. Consecuentemente, para poder tener una base robusta que logre subir los porcentajes de precisión, se necesita que las muestras sean extraídas por una mayor cantidad de tiempo, mayor cantidad de personas y con más variedad de situaciones.

Una factor que puede ser tardado en estos casos es etiquetar los datos. En esta primera aproximación se realizó únicamente con 6 minutos en el caso de que los sensores estuvieran en la misma persona y otros 6 minutos donde eran independientes. Además, durante el entrenamiento se pasaron primero todos los datos que si correspondían a la misma persona y después todos los datos que no correspondían.

Para futuras experimentaciones esto podría cambiar, sería mejor entremezclar los datos eso sí teniendo mucho cuidado de que las etiquetas correspondan a las muestra sin error alguno. Para ello podría ser útil las gráficas en tiempo real, pues es más fácil de esa manera identificar qué actividades se realizaron junto con la gráfica de tiempo y saber que secuencia en los datos corresponde a determinada persona y actividad. Ya por último, determinar cual es la etiqueta de esa secuencia, si corresponde o no corresponde al mismo cuerpo.

En aproximaciones más complejas incluso se podría identificar no sólo si los sensores se encuentran dentro de las personas, sino que actividad se está realizando en ese momento. Esto podría ser útil igualmente con fines médicos e incluso podría avisar si la persona se encuentra en una situación de peligro.

De este script entrenamiento lo que se obtendrá al final son los pesos y el modelo de entrenamiento que se encuentra en ese archivo con extensión h5. El modelo final que se va a utilizar debe guardarse con un nombre en específico y este nombre se utilizará para ya la implementación y unificación con él código de las demás fases. Este modelo es el que hará la comparación de ambos sensores y dirá si el sensor es aceptado o rechazado.

6. Integración con el código principal

Para integrar la red neuronal desarrollada con el código principal lo único que se necesita hacer es cambiar algunas de las funciones que recopilan los datos en en el sistema anterior. Estas se remplazan por las descritas en el anexo C. Además, estos códigos de manera completa se pueden ver también en el repositorio de Sensors_NN.

En los anexos sólo se describen las funciones cambiadas pero en el repositorio los archivos con el mismo nombre ya contendrán el sistema completo.

Estas funciones lo que cambia es que ahora lo único que harán es recolectar los datos en un arreglo determinado y se van a comparar usando el modelo creado anteriormente en el entrenamiento. Esto permitirá que la comparación se haga únicamente utilizando la predicción arrojada por el modelo sin tener que extraer ninguna otra característica o hacer alguna otra operación matemática con los datos.

El resto del código sigue funcionando de la manera descrita en el repositorio del protocolo.

A. Código: Extracción de muestras para la base de datos

Listing 2: Obtencion$_d$atos$_e$ntrenamiento.py

```
from mpu6050 import mpu6050
import time
import datetime
from itertools import count
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# En algunos casos es ms fcil utilizar la librera mpu6050, pero se puede cambiar para utilizar la FaBo9Axis_MPU9250
#import FaBo9Axis_MPU9250
#mpu9250 = FaBo9Axis_MPU9250.MPU9250()


def save_data_1(reg=100):
    ''' Solamente guarda los datos en archivos .txt independientes. Toma como argumento el nmero de muestras que se desean tomar. No hay ningn tiempo de espera. '''
    sensor=mpu6050(0x68, bus=1)
    names = ['acc_x_1', 'acc_y_1', 'acc_z_1', 'gyro_x_1', 'gyro_y_1', 'gyro_z_1']
    for n in names:
        globals()[n] =open(n +'.txt','w')
    for i in range(reg):
        for j in range(150):
            #accel = mpu9250.readAccel()
            #gyro = mpu9250.readGyro()
            accel=sensor.get_accel_data()
            gyro=sensor.get_gyro_data()
            datos = [accel['x'], accel['y'], accel['y'], gyro['x'], gyro['y'], gyro['z']]
            for n,d in zip(names,datos):
                globals()[n].write(str(d)+'\t')
        for n in names:
            globals()[n].write('\n')
    for n in names:
        globals()[n].close
    print('Datos guardados')

def plot_graphs_1():
    ''' Grafica en tiempo real mediante una animacin
    los datos obtenidos del eje x del acelermetro.'''
    plt.style.use('fivethirtyeight')

    sensor=mpu6050(0x68, bus=1)
    X = list()
    Y = list()
    for j in range(150):
        accel=sensor.get_accel_data()
        X.append(j)
        Y.append(accel['x'])
        time.sleep(1)
        print(j)

    ani = FuncAnimation(plt.gcf(), animate, interval=1000)

    plt.tight_layout()
    plt.show()


```

```
55 def animate(i):
56     plt.cla()
57     plt.plot(X,Y, label='acc_x')
58     plt.tight_layout()
59
60
61 def save_and_plot_1(reg=100):
62     ''' Guarda los datos en archivos .txt independientes especificando que se trata del sensor 1.
           Toma como argumento el nmero de muestras que se desean tomar. Entre cada toma de muestra se
           espera 10 milisegundos. Al final grfica cada eje del acelermetro y del giroscopio como
           series de tiempo para comprender mejor los movimientos realizados durante la toma de
           muestras. '''
63     from datetime import datetime
64     now = datetime.now()
65     sensor=mpu6050(0x68, bus=1)
66     num = list(range(150*reg))
67     dato = [[],[],[],[],[],[]]
68     names = ['acc_x_1', 'acc_y_1', 'acc_z_1', 'gyro_x_1', 'gyro_y_1', 'gyro_z_1']
69     #name = [ac_x_1, ac_y_1, ac_z_1, gyr_x_1, gyr_y_1, gyr_z_1]
70     for n in names:
71         globals()[n] =open(n +now.strftime(" %d-%m-%Y %H-%M")+'.txt','w')
72     for i in range(reg):
73         for j in range(150):
74             #accel = mpu9250.readAccel()
75             #gyro = mpu9250.readGyro()
76             accel=sensor.get_accel_data()
77             gyro=sensor.get_gyro_data()
78             datos = [accel['x'], accel['y'], accel['y'], gyro['x'], gyro['y'], gyro['z']]
79             for a,d in enumerate(zip(names,datos)):
80                 globals()[d[0]].write(str(d[1])+'\t')
81                 dato[a].append(d[1])
82             time.sleep(0.01)
83         for n in names:
84             globals()[n].write('\n')
85     for n in names:
86         globals()[n].close
87     print('Datos guardados')
88
89     fig, axs = plt.subplots(3, 2, sharex = True)
90     fig.suptitle('Sensor 1')
91     axs[0,0].plot(num,dato[0],'mediumblue')
92     axs[0,0].set_title('Acelermetro')
93     axs[1,0].plot(num,dato[1], 'darkviolet')
94     axs[2,0].plot(num,dato[2], 'forestgreen')
95     axs[0,1].plot(num,dato[3],'deepskyblue')
96     axs[0,1].set_title('Giroscopio')
97     axs[1,1].plot(num,dato[4], 'gold')
98     axs[2,1].plot(num,dato[5], 'red')
99     fig.savefig(now.strftime(" %d-%m-%Y %H-%M")+'.png')
100
101    #fig.show()
102
103
104 def save_and_plot_2(reg=100):
105     ''' Guarda los datos en archivos .txt independientes especificando que se trata del **sensor
           2**. Toma como argumento el nmero de muestras que se desean tomar. Entre cada toma de
           muestra se espera 10 milisegundos. Al final grfica cada eje del acelermetro y del giroscopio
           como series de tiempo para comprender mejor los movimientos realizados durante la toma de
           muestras. '''
106     from datetime import datetime
```

```
    now = datetime.now()
    names = ['acc_x_2', 'acc_y_2', 'acc_z_2', 'gyro_x_2', 'gyro_y_2', 'gyro_z_2']
    sensor=mpu6050(0x68, bus=1)
    num = list(range(150*reg))
    dato = [[],[],[],[],[],[]]
    for n in names:
        globals()[n] =open(n +now.strftime(" %d-%m-%Y %H-%M")+'.txt','w')
    for i in range(reg):
        for j in range(150):
            #accel = mpu9250.readAccel()
            #gyro = mpu9250.readGyro()
            accel=sensor.get_accel_data()
            gyro=sensor.get_gyro_data()
            datos = [accel['x'], accel['y'], accel['y'], gyro['x'], gyro['y'], gyro['z']]
            for a,d in enumerate(zip(names,datos)):
                globals()[d[0]].write(str(d[1])+'\t')
                dato[a].append(d[1])
            time.sleep(0.01)
        for n in names:
            globals()[n].write('\n')
    for n in names:
        globals()[n].close
    print('Datos guardados')

    fig, axs = plt.subplots(3, 2, sharex = True)
    fig.suptitle('Sensor 2')
    axs[0,0].plot(num,dato[0],'mediumblue')
    axs[0,0].set_title('Acelermetro')
    axs[1,0].plot(num,dato[1], 'darkviolet')
    axs[2,0].plot(num,dato[2], 'forestgreen')
    axs[0,1].plot(num,dato[3],'deepskyblue')
    axs[0,1].set_title('Giroscopio')
    axs[1,1].plot(num,dato[4], 'gold')
    axs[2,1].plot(num,dato[5], 'red')
    fig.savefig(now.strftime(" %d-%m-%Y %H-%M")+'.png')


# Se corre la funcin para guardar los datos del sensor 1.
save_and_plot_1(100)
```

B. Código: Entrenamiento de la red neuronal y obtención de pesos

Listing 3: Entrenamiento.py

```
# convlstm model
from numpy import mean
from numpy import std
from numpy import dstack
from pandas import read_csv
import keras.models
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import LSTM
from keras.layers import TimeDistributed
from keras.layers import ConvLSTM2D
from keras.utils import to_categorical
```

```
from matplotlib import pyplot
from datetime import datetime

# Carga un solo archivo como una matriz numpy
def load_file(filepath):
    dataframe = read_csv(filepath, header=None, delim_whitespace=True)
    return dataframe.values

# carga una lista de archivos y regresa como una matriz numrica 3d
def load_group(filenames, prefix=''):
    loaded = list()
    for name in filenames:
        data = load_file(prefix + name)
        loaded.append(data)
    # apila un grupo para que las caractersticas sean de 3 dimensiones
    loaded = dstack(loaded)
    return loaded

# carga el dataset de un grupo, entrenamiento o prueba
def load_dataset_group(group, prefix=''):
    filepath = prefix + group + '/Inertial Signals/'
    # carga todos los archivos como un arreglo
    filenames = list()
    # aceleracin sensor 1
    filenames += ['acc_x_1_'+group+'.txt', 'acc_y_1_'+group+'.txt', 'acc_z_1_'+group+'.txt']
    # giroscopio sensor 1
    filenames += ['gyro_x_1_'+group+'.txt', 'gyro_y_1_'+group+'.txt', 'gyro_z_1_'+group+'.txt']
    # aceleracin sensor 2
    filenames += ['acc_x_2_'+group+'.txt', 'acc_y_2_'+group+'.txt', 'acc_z_2_'+group+'.txt']
    # giroscopio sensor 2
    filenames += ['gyro_x_2_'+group+'.txt', 'gyro_y_2_'+group+'.txt', 'gyro_z_2_'+group+'.txt']
    # carga los datos de entrada
    X = load_group(filenames, filepath)
    # carga las clases de salida
    y = load_file(prefix + group + '/y_'+group+'.txt')
    return X, y

# carga los datasets, devuelve datos y etiquetas de entrenamiento y prueba
def load_dataset(prefix=''):
    # carga todo el entrenamiento
    trainX, trainy = load_dataset_group('train', prefix + 'SensorDataset/')
    print(trainX.shape, trainy.shape)
    # carga todo el test
    testX, testy = load_dataset_group('test', prefix + 'SensorDataset/')
    print(testX.shape, testy.shape)
    # valores de clase de compensacin cero
    trainy = trainy - 1
    testy = testy - 1
    # codificacin one hot en y
    trainy = to_categorical(trainy)
    testy = to_categorical(testy)
    print(trainX.shape, trainy.shape, testX.shape, testy.shape)
    return trainX, trainy, testX, testy

# entrena y evalua el modelo
def evaluate_model(trainX, trainy, testX, testy):
    verbose, epochs, batch_size = 0, 25, 64
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2], trainy.shape[1]
    # reshape en subsecuencias (muestras, pasos de tiempo, filas, columnas, canales)
    n_steps, n_length = 10, 15
```

```
    trainX = trainX.reshape((trainX.shape[0], n_steps, 1, n_length, n_features))
    testX = testX.reshape((testX.shape[0], n_steps, 1, n_length, n_features))
    # define modelo
    model = Sequential()
    model.add(ConvLSTM2D(filters=64, kernel_size=(1,3), activation='relu', input_shape=(n_steps, 1,
        n_length, n_features)))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    # entrena la red
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size, verbose=verbose)
    now = datetime.now()
    model.save('convlstm_sensors'+now.strftime(" %d-%m-%Y %H-%M-%S")+'.h5')
    # evalua modelo
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
    return accuracy

# resumir puntuaciones
def summarize_results(scores):
    print(scores)
    m, s = mean(scores), std(scores)
    print('Precisin: %.3f%% (+/-%.3f)' % (m, s))

# correr un experimento (10 veces)
def run_experiment(repeats=10):
    # cargar los datos
    trainX, trainy, testX, testy = load_dataset()
    # repetir experimento
    scores = list()
    for r in range(repeats):
        score = evaluate_model(trainX, trainy, testX, testy)
        score = score * 100.0
        print('Exp-#%d: %.3f' % (r+1, score))
        scores.append(score)
    # resumen de resultados
    summarize_results(scores)

# correr un experimento
run_experiment()
```

C. Código: Funciones remplazadas en el funcionamiento del sistema completo

Listing 4: B.py

```
from tensorflow import keras

def obtener_datos_B():
    '''
    obtener_datos_B
    Descripcion:  Lee datos del sensor.
    Argumentos:
    Returns:      --regresa un arreglo con los datos del sensor
    '''
    dato = [[],[],[],[],[],[]]
```

```
    mpu9250 = FaBo9Axis_MPU9250.MPU9250()
    for j in range(150):
        accel = mpu9250.readAccel()
        gyro = mpu9250.readGyro()
        datos = [accel['x'], accel['y'], accel['y'], gyro['x'], gyro['y'], gyro['z']]
        for a,d in enumerate(datos):
            dato[a].append(d)
        time.sleep(0.01)
    return dato

def obtener_datos_C(client_sock):
    '''
    obtener_datos_C
    Descripcion:  Recibe los obtenidos del sensor externo
    Argumentos:   --client_sock es la informacion del socket
    Returns:      --regresa un arreglo con los datos del sensor externo
    '''
    while True:
        data = client_sock.recv(4096)
        if not data: break
        data_arr = pickle.loads(data)
    return data_arr

def comparacion_nn(client_sock, model):
    '''
    comparacion_nn

    Descripcion:  Realiza la comparacion de los datos obtenidos y recibidos para
                  posteriormente identificar si pertenece a la red o no, ademas de
                  enviarle una confirmacion al sensor de que pertenece a la red
                  y posteriormente recibir su ID

    Argumentos:   --client_sock es la informacion del socket del sensor
                  --model es el modelo cargado desde el archivo con extension h5

    Returns:      --regresa el id del sensor

    '''
    B = obtener_datos_B()
    C = obtener_datos_C()
    X = B+C
    X_t = np.array([np.array(X).T])
    print(np.shape(X_t))
    X = X_t.reshape((X_t.shape[0], 10, 1, 15, X_t.shape[2]))
    print(np.shape(X))
    pred = model.predict_classes(X)
    if pred == 0:
        resultado = 'Aceptado'
        confirmacion = b'1'
        client_sock.send(confirmacion)
        id_sensor = client_sock.recv(1024)
    else:
        resultado = 'Rechazado'
        id_sensor = b' '
    print(resultado)
    return id_sensor

def run_imu(server_sock):
    '''
    run_imu
```

```
    Descripcion:  Lee datos del sensor y recibe los obtenidos del sensor externo adems de llamar a
        la funcion comparacion_nn()
    Argumentos:    --server_sock es la informacion del socket
    Returns:       --regresa el id del sensor
    '''
    print('Esperando conexion')
    client_sock, client_info = server_sock.accept()
    print("Conexion aceptada de ", client_info)
    model = keras.models.load_model('convlstm_sensors.h5')
    ID = comparacion_nn(client_sock, model)

    return ID
```

Listing 5: C.py

```
def obtener_datos_C():
    '''
    obtener_datos_C
    Descripcion:  Lee datos del sensor.
    Argumentos:
    Returns:       --regresa un arreglo con los datos del sensor
    '''
    dato = [[],[],[],[],[],[]]
    mpu9250 = FaBo9Axis_MPU9250.MPU9250()
    for j in range(150):
        accel = mpu9250.readAccel()
        gyro = mpu9250.readGyro()
        datos = [accel['x'], accel['y'], accel['y'], gyro['x'], gyro['y'], gyro['z']]
        for a,d in enumerate(datos):
            dato[a].append(d)
        time.sleep(0.01)
    return dato


# Iniciar la conexin
sock1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
HOST_SOCK = sys.argv[1]
PORT_SOCK = 5005
server_sock_address = (HOST_SOCK, PORT_SOCK)
print('connecting to {} port {}'.format(*server_sock_address))
sock1.connect(server_sock_address)

# Datos sensor 2
C = obtener_datos_C()
data_string = pickle.dumps(C)

# Enviar datos a B
sock1.send(data_string)

data = sock1.recv(1)
print(data)
if data==b'1':
        sock1.send(sensor_id)
else:
        sock1.close()
```

Referencias

[1] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in 2015 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, 2015, pp. 4580–4584.

[2] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in Advances in neural information processing systems, 2015, pp. 802–810.

[3] J. Brownlee, Long short-term memory networks with python: develop sequence prediction models with deep learning. Machine Learning Mastery, 2017.

[4] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 2625–2634.

[5] S. El Yazidi, "Autenticación de sensores en una red de área corporal con base en características de movimientos comunes," p. 103, 2017.