Departamento de Ingeniería Eléctrica
Facultad de Ciencias Físicas y Matemáticas
Universidad de Chile
EL4106 Inteligencia Computacional

Informe práctica profesional 1

Banco Santander

Diego Irarrázaval I.

Profesor:
Javier Ruiz del Solar.

Auxiliar:
Patricio Loncomilla Z.

Fecha:
24 de septiembre de 2020

Índice

Índice de figuras

Índice de tablas

Índice de Algoritmos

1. Introducción

2. Desarrollo

2.1. Extracción y manejo de datos

En esta sección nos enfocaremos en la extracción de los datos y en el preprocesamiento. Cabe destacar que esta es una de las partes más importantes de la tarea ya que el volumen de datos lo suficientemente grande como para no realizar reducciones previas.

2.1.1. Extracción de los datos

Como se trabajó en `Google Colab`, para la lectura y extracción de datos se utilizó la siguiente función:

```
from google.colab import files
uploaded = files.upload() #elegimos el archivo en el directorio
    correcto

!unzip EMG_data_for_gestures-master.zip #descomprimimos
```

Luego, para la lectura de los archivos `.zip` se utilizó la librería `glob`. El código a continuación muestra como se utilizó:

```
archivos = glob.glob('**/*.txt', recursive=True)
```

El resultado `archivos` es una lista que contiene todas las direcciones de los archivos `.txt` que había en el directorio actual. Es decir, todos los datos más el archivo `README.txt`. A partir de `archivos` se crean dos listas: `archivos_test` y `archivos_train` que contienen las direcciones de los `.txt` de los conjuntos de entrenamiento y prueba (que corresponde a los últimos 8 sujetos, según se pide en el enunciado).

Para extraer los datos de los archivos `.txt` se utilizó `pandas`. De esta forma, se obtuvieron los conjuntos de entrenamiento y de prueba:

```
data_test = [pd.read_csv(fp,sep='\t') for fp in archivos_test]
data_train = [pd.read_csv(fp,sep='\t') for fp in archivos_train]

#Concatenamos las listas de dataFrames obtenidos:
datos_train = pd.concat(data_train)
datos_test = pd.concat(data_test)
```

Con esto ya tenemos a disposición el conjunto de entrenamiento y el de prueba. continuación, se procedió al preprocesamiento y filtrado de los datos.

2.1.2. Preprocesamiento

Para comenzar el prerpocesamiento, se eliminaron los datos con etiqueta '7', según se especificó en el enunciado. Por otro lado, dado que la clasificación/detección debe ser según los datos obtenidos de los sensores y no según el momento en que se realizaron (los gestos se realizan de

forma secuencial: primero un gesto, luego pausa, luego otro gesto y así...), se eliminó la columna 'time'del *dataset*.

Por otro lado, es importante destacar que la cantidad de datos y la no evidente relación entre estos y la clase obliga a aplicar procesamientos a éstos. A continuación se muestra lo obtenido en el canal 1:

En la imagen, no se puede distinguir un nivel característico para cada clase (tal vez con excepción de las pausas).

Ya realizados estos pasos previos, se procedió al tratamiento de ventanas de los datos. Esto consiste en separar los datos en conjuntos (ventanas) de ancho definidas por el usuario. Además luego de crear una ventana, habrá un salto de cierta cantidad de datos. Se implementó la siguiente función para esto:

```
def getVentanas(datos,ancho,salto):
  ventanas = []
  for i in range(int(len(datos)/(ancho + salto))):
    k = i*(ancho + salto)
    k_final = (i+1)*ancho + i*salto
    ventanas.append(datos[k:k_final,:])
  return ventanas
```

A priori, no se puede saber cual es la cantidad óptima de datos por ventanas ni el tamaño de los saltos. Por otro lado, por la forma de la construcción de las ventanas, estas pueden tener múltiples etiquetas o, en el caso de la última ventana, esta puede no tener el ancho requerido por el usuario. Para eliminar estas ventanas, se implementó una función `filtrarVentanas`:

```
def filtrarVentanas(ventanas,ancho):
  i=0
  while i<len(ventanas):
  #for i in range(len(ventanas)):
    if len(ventanas[i]) != ancho:
      del ventanas[i]
    if not(all(x == list(ventanas[i][:,8])[0] for x in list(
     ventanas[i][:,8]))):
      del ventanas[i]
    i+=1
  return ventanas
```

Con esto, obtenemos nuestro conjunto de entrenamiento. Las ventanas luego serán procesadas para extraer características según el *paper* en [1]. Por cada canal en una ventana k, se extrajo 9 características:

- Promedio
- Varianza
- Máximo
- Mínimo
- Rango
- Cruces por cero
- RMS
- Skewness
- Kurtosis

Para realizar esto, se implementó una función `getChar`. Ésta recibe el arreglo de ventanas y entrega un arreglo con las características de cada canal para cada ventana:

	CH1	**CH2**	$\cdots$	**CH8**
1	[m,v,...]	[m,v,...]		[m,v,...]
2				
3				
$\vdots$			$\ddots$	
n				

Tabla 1: Forma de la matriz de características.

3. Resultados

3.1. Clasificador de gestos

Entrenamiento:

3.1.1. Redes neuronales

A continuación se muestran los resultados obtenidos con una red neuronal.

Ancho	Salto	# car	#N. C. Oculta	Accuracy	Train time [s]
200	10	48	30	0.64	226
200	10	24	30	0.61	1836
200	100	24	23	0.50	1247
200	100	48	23	0.67	1883
250	100	48	15	0.64	860
250	100	20	10	0.67	2141

Tabla 2: Resultados obtenidos con una red neuronal

3.1.2. Random Forest

Otro método de clasificación que se implementó es `Random Forest`. Los resultados obtenidos se muestran a continuación:

Ancho	Salto	# car	# estimadores	Accuracy
200	10	24	1000	0.56
200	100	24	1000	0.54
200	100	48	300	0.66
250	100	48	300	0.60
200	100	20	300	0.63
200	100	all	300	0.65

Tabla 3: Resultados para el clasificador con Random Forest

Prueba:

Para evaluar el conjunto de prueba se utilizó el mejor resultado obtenido de Random Forest. Esto debido a mayor simplicidad para evaluar y porque tiene menor tiempo de entrenamiento (utilizando random forest se tienen tiempos de a lo más 20 segundos):

$$Accuracy = 0{,}51$$

3.2. Detector de gestos

Entrenamiento:

Previo a implementar el detector de gestos, se procedió a reescribir las etiquetas para que sólo existan dos clases: '0'(pausa) o '1'(gesto):

```
label_bin_train = [0 if y_train[i] == 0 else 1 for i in range(len(
    y_train))]
```

Se implementó el clasificador usando SVM. Esto debido a que por naturaleza, SVM es bueno para un clasificador binario. Al igual que para el clasificador de gestos se variaron distintos parámetros hasta obtener el mejor detector:

Ancho	Salto	# car	Kernel	Accuracy
250	100	15	Lineal	0.66
250	100	48	rbf	0.70
250	100	48	sigmoidal	0.32
250	100	'all'	rbf	0.71

Tabla 4: Resultados para el detector con SVM

Prueba

Dado los resultados obtenidos con SVM con kernel 'rbf'y utilizando todas las características, se procedió a utilizar el conjunto de prueba correspondiente a los 8 últimos sujetos que se separaron del dataset al inicio del experimento. Los resultados obtenidos se muestran a continuación:

$$\begin{bmatrix} 1185 & 472 \\ 235 & 261 \end{bmatrix}$$

La *accuracy* obtenida es 0.66. Cabe destacar que el tiempo de entrenamiento con kernel rbf era dos o tres veces superior al kernel lineal.

4. Análisis de resultados

A pesar de realizar variedad de experimentos con distintos ancho de ventana, distinto salto, variando también la cantidad de características y distintos aspectos de la red neuronal, los resultados obtenidos son pésimos. Además los tiempos de entrenamiento son altísimos (1800 [s] equivalen a 30 minutos). Por otro lado, aunque no es posible d verificar en la tabla 2, la puntuación obtenida en *accuracy* usualmente no variaba durante el entrenamiento. Esto es, entre la *Epoch 001* y la *Epoch 101*, no había diferencia en la *accuracy*. Esta situación se repitió en varios experimentos.

Tomando en cuenta lo anterior, se procedió a probar distintas combinaciones de RandomForest que a pesar de no obtener resultados mejores a la red neuronal el tiempo de entrenamiento era dos ordenes de magnitud menor. De esta forma, al evaluar en el conjunto de prueba los resultados obtenidos fueron cercanos a lo que se obtendría lanzando monedas. Esto se debe en primer lugar a la gran cantidad de clase 0 que hay en los datos.

Sobre el detector se puede decir que los resultados son mejores de los esperados. Esto tomando en cuenta que en las presentaciones de compañeros los resultados obtenidos eran equivalentes a lanzar monedas.

Bibliografía

[1] Macro-Class Selection for Hierarchical K-NN Classification of inertial sensor data. https://www.crcv.ucf.edu/papers/PECCS_2012.pdf

5. Anexos

Ejemplo entrenamiento red neuronal

Red Nhidden= 23, 24 FEATURES, ANCHO = 200 SALTO = 100 CON CLASE 0

Epoch: 0001
Accuracy validation: 0.507657
Epoch: 0101
Accuracy validation: 0.507657
Epoch: 0201
Accuracy validation: 0.507657
Epoch: 0301
Accuracy validation: 0.507657
Epoch: 0401
Accuracy validation: 0.507657
Epoch: 0501
Accuracy validation: 0.507657
Epoch: 0601
Accuracy validation: 0.507657
Epoch: 0701
Accuracy validation: 0.507657
Epoch: 0801
Accuracy validation: 0.507657
Epoch: 0901
Accuracy validation: 0.507657
Epoch: 1001
Accuracy validation: 0.507657
Epoch: 1101
Accuracy validation: 0.507657
Epoch: 1201
Accuracy validation: 0.507657
Epoch: 1301
Accuracy validation: 0.507657
Epoch: 1401
Accuracy validation: 0.507657
Optimization Finished!
Accuracy validation: 0.507657
Confusion matrix validation

$$\begin{bmatrix} 663 & 0 & 0 & 0 & 0 & 0 & 0 \\ 84 & 0 & 0 & 0 & 0 & 0 & 0 \\ 97 & 0 & 0 & 0 & 0 & 0 & 0 \\ 117 & 0 & 0 & 0 & 0 & 0 & 0 \\ 114 & 0 & 0 & 0 & 0 & 0 & 0 \\ 115 & 0 & 0 & 0 & 0 & 0 & 0 \\ 116 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Tiempo total de entrenamiento: 1247.6460301876068