dfMaker

Una serie de funciones para transformar archivos `.json` a `dataFrames` en `R`

bHerreno

8/12/2021

Índice

1 Introducción **2**
1.1 Una estrcuturación en tres pasos 2

2 `dfMaker`: una función para estrcturar archivos `.json` generados con `OpenPose` **3**
2.1 Carga de archivos y *unlist* de los datos 5
2.2 Ordenacion de los datos y uso de la función `split()` para crar un dataframe definitivo 8
2.3 Ultimas comprobaciones de la función 14

3 `videoMaker` del fotograma al vídeo **15**
3.1 Presentación de la función 15
3.2 Ejemplo 18

4 `dfMaker` cuando un solo vídeo no es suficinete **21**
4.1 Últimas comprobaciones 26

5 conclusión **27**

Referencias **28**

El caos es el orden aún por descifrar

— José Saramago —

---

1 Introducción

OpenPose es "el primer sistema multipersonal en tiempo real que detecta conjuntamente los puntos clave del cuerpo humano, las manos, el rostro y los pies" (Cao et al. 2019). Esta herramienta tiene un gran potencial en infinitud de campos que estudian el movimiento humano, pero en determinadas ocasiones es necesario transformar la estructura de los datos para su posterior análisis. Como sucede en el caso de querer utilizar el lenguaje `R` (R Core Team 2021) sobre los archivos generados por OpenPose.

El motivo de ordenar los datos es el frágil equilibrio entre legibilidad y redundancia. Por un lado, un lenguaje como `R` basado en la programación de objetos y usado principalmente en estadística requiere de datos con estructuras repetitivas que permitan establecer (co)relaciones entre diferentes conjuntos de datos; en el otro lado, un programa pensado para generar grandes cantidades de datos no puede permitirse el lujo de repetir datos o darles una estructura extendida a los mismos (`OpenPose`), dicho programa necesita guardar la mayor cantidad de información en el menor espacio posible. Pero no hay problema, ni contrariedad, en este equilibro, simplemente es necesario mover el equilibrio a las necesidades de nuestra investigación.

`R` permite leer casi cualquier formato conocido —de no ser así, se puede programar para que sí sea así— incluyendo `.json` (Ooms 2014) que es el formato en que se encuentran los datos generados por OpenPose. Lo primero para poder transformar los datos es conocer el formato y la estructura deseados, a la vez que conocemos el formato y la estructura dada (Cuadro 1).

Normalmente la estructura más utilizada y una de las más sencillas para manejar variables y observaciones en `R` es el `dataFrame`, siendo la estructura más adecuada para trabajar con los datos, o al menos, obtener una primera estructura base que permita el estudio de los datos. El problema a resolver es que los datos en crudo (*Raw Data*) presentan una estructura de listas jerárquicas y la lista en el último nivel se encuentra desestructurada.

Estructura dada	Estructura deseada
Listas dentro de listas	`DataFrame`

Formato dado	Formato deseado
`JSON`	Preferiblemente `CSV`, pero vale cualquiera

Cuadro 1: Estructuración y selección formato de los datos generados con OpenPose.

1.1 Una estrcuturación en tres pasos

El objetivo final es obtener `dataFrames` que contengan los datos de los vídeos que se desean comparar, pero previamente hay que estructurar los fotogramas que componen los vídeos, y antes de eso las variables que componen cada fotogramas (Figura 1).

$$\textit{Datos sin procesar} \xrightarrow{\text{frameMaker}} \textit{Fotograma} \xrightarrow{\text{videoMaker}} \textit{Vídeo} \xrightarrow{\text{dfMaker}} \textit{Conjunto de vídeos}$$

Figura 1: Procedimiento de ordenación del os datos

Hay que tener también en cuenta que `frameMaker` obtiene algunas variables a partir del nombre del archivo original (Cuadro 2) y estas variables en caso de presentar una estructura diferente darían como resultado columnas vacías (`NAs`), pero no es problema, pues dichas columnas pueden

Variables obtenidas del archivo	Variables obtenidas del nombre del archivo
u (grado de confianza)	words
x	frame
y	names
points	
typePoint	

Cuadro 2: Variables creadas por `dfMaker`

ser eliminadas con suma facilidad. Por otro lado, podría darse el caso de querer añadir una variable nueva, siendo necesario modificar el código de la función; aun así, las partes principales de la función que afectan a la estructuración de los datos obtenidos de `OpenPose` no se verían afectadas.

En realidad `frameMaker` es la única de las tres funciones que trabaja con los datos originales, el resto (`videoMaker` y `dfMaker`) simplemente repiten la función en el orden y la forma dados para construir un `dataFrame` que contenga todos los fotogramas de un vídeo o de varios vídeos. En otras palabras, `videoMaker` y `dfMaker` simplemente sirven para ahorrar pasos a la hora de copiar y ensamblar los datos, pero no aportan estructura a los datos más allá de una simple cadena de montaje; caso diferente al de `frameMaker` que es el la función esencial que da el orden a los datos.

Ahora bien, `videoMaker` permite guardar todos los fotogramas en un solo archivo dentro de una carpeta conjunta y `dfMaker` repite el número de veces necesario `videoMaker` para juntar los vídeos en un solo `dataFrame`.

2 `dfMaker`: una función para estrcturar archivos `.json` generados con `OpenPose`

Talk is cheap. Show me the code!

— Linus Torvalds (creador del Kernel Linux) —

```
frameMaker<-function(file){
  require(jsonlite)

  rawData<-read_json( path = file)

  rawData<-rawData[2]

  rawData<-rawData[[1]]

  dfPoints=NULL
  for (id in 1:length(rawData)) {

    if(length(rawData)==0) next

    points<-data.frame(unlist(rawData[[id]]),people=id)

    dfPoints=rbind(dfPoints,points)

  }
```

```
  pattern<- sample(T, size=137*3, replace= T) # points triplicates


  pattern<-c(F,pattern)

  if(!is.null(dfPoints)){
    pattern<-rep(pattern, times=max(dfPoints$people))
  }



  dfPoints<-dfPoints[pattern,]
  ##

  type<-rownames(dfPoints)
  type<-gsub("_2d[0-9]*", "", type)

 ###
  words<-gsub("_000000.*", "", file)


  words<-gsub(".*[0-9]_", "", words)


  ###

  frame<-gsub(paste(".*_000", sep = ""), "", file)

  frame<-as.numeric(gsub("_.*","", frame))
  ###

  name<-gsub(paste(".*//", sep = ""), "", file)

  name<-gsub(paste("/.*", sep = ""), "", name)

  ###

    triplet<-c(T,F,F)
  ###
  groups <- c("x", "y", "c") # the variables of the final df
  if(!is.null(dfPoints)){
    pointsDF<-data.frame(split(dfPoints[,1], f = groups),
                         people=dfPoints$people[triplet],
                         typePoint=  type[triplet],
                         point= c(0:24,0:69,0:20,0:20),
                         words=words,
                         frame=frame,
                         name=name) # split in 3 columns

  }else{
    pointsDF=NULL
  }
  return(pointsDF)

}
```

2.1 Carga de archivos y *unlist* de los datos

Los archivos `.json` generados por openPose tienen la misma estructura, la estructura mínima compartida por todos los archivos es una lista compuesta de 2 listas: `version`, y `people`. A su vez, `people` se compone de tantas listas como personas se detecten en el fotograma, mientras que `version` solo contiene el número de la versión de `OpenPose` con el que ha sido generado el archivo, esta última lista simplemente se obvia.

```
library(jsonlite)

sample<-read_json("/data/home/agora/data/trainningDataVideos/videosJSON/2014-10-10_0600_US_KCAL_Enter



str(sample, max.level=1)
```

```
## List of 2
##  $ version: num 1.3
##  $ people :List of 1
```

```
# In the second case, there are 4 lists inside `people` list

sampleMultoipleIDs<- read_json( path = "/data/home/agora/data/trainningDataVideos/videosJSON//2014-10

str(sampleMultoipleIDs,max.level=2)
```

```
## List of 2
##  $ version: num 1.3
##  $ people :List of 4
##   ..$ :List of 9
##   ..$ :List of 9
##   ..$ :List of 9
##   ..$ :List of 9
```

```
rawData<- read_json( path = "/data/home/agora/data/trainningDataVideos/videosJSON//2014-10-10_0600_US

rawData<-rawData[2] #  choose only the `people` list

  rawData<-rawData[[1]] # and  take in account the number of lists inside people
```

Los datos generados por `OpenPose` se encuentran enlistados de forma jerárquica (Figura 2) repitiéndose la misma estructura básica que tan solo cambia en el número de veces que se repiten las listas dentro de `people`; por cada persona detectada por el *software* se añaden 9 listas correspondientes a los puntos 2d y 3d detectados junto al ID que identifica a la persona detectada.

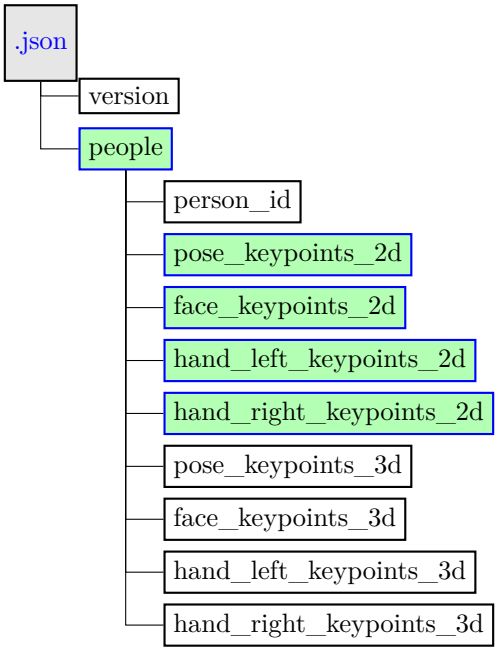

Figura 2: Estructura de los datos generados por `OpenPose`. En verde las listas validas.

```
str(rawData,max.level = 2)
```

```
## List of 4
##  $ :List of 9
##   ..$ person_id              :List of 1
##   ..$ pose_keypoints_2d      :List of 75
##   ..$ face_keypoints_2d      :List of 210
##   ..$ hand_left_keypoints_2d :List of 63
##   ..$ hand_right_keypoints_2d:List of 63
##   ..$ pose_keypoints_3d      : list()
##   ..$ face_keypoints_3d      : list()
##   ..$ hand_left_keypoints_3d : list()
##   ..$ hand_right_keypoints_3d: list()
##  $ :List of 9
##   ..$ person_id              :List of 1
##   ..$ pose_keypoints_2d      :List of 75
##   ..$ face_keypoints_2d      :List of 210
##   ..$ hand_left_keypoints_2d :List of 63
##   ..$ hand_right_keypoints_2d:List of 63
##   ..$ pose_keypoints_3d      : list()
##   ..$ face_keypoints_3d      : list()
##   ..$ hand_left_keypoints_3d : list()
##   ..$ hand_right_keypoints_3d: list()
##  $ :List of 9
##   ..$ person_id              :List of 1
##   ..$ pose_keypoints_2d      :List of 75
##   ..$ face_keypoints_2d      :List of 210
##   ..$ hand_left_keypoints_2d :List of 63
##   ..$ hand_right_keypoints_2d:List of 63
##   ..$ pose_keypoints_3d      : list()
##   ..$ face_keypoints_3d      : list()
##   ..$ hand_left_keypoints_3d : list()
```

```
##   ..$ hand_right_keypoints_3d: list()
##  $ :List of 9
##   ..$ person_id              :List of 1
##   ..$ pose_keypoints_2d      :List of 75
##   ..$ face_keypoints_2d      :List of 210
##   ..$ hand_left_keypoints_2d :List of 63
##   ..$ hand_right_keypoints_2d:List of 63
##   ..$ pose_keypoints_3d      : list()
##   ..$ face_keypoints_3d      : list()
##   ..$ hand_left_keypoints_3d : list()
##   ..$ hand_right_keypoints_3d: list()
```

La selección de la lista en la Figura 2 requiere de una serie de transformaciones que hay que tener en cuenta, pues aunque las listas de los puntos 3d (`keypoints_3d`) al utilizar la función `unlist()` desaparecerán al no contener datos, `person_id` si que contiene un dato y por tanto:

$$person\_id \neq \emptyset$$

.

```
rawData[[2]]->listkeyPoints
listkeyPoints[9]
```

```
## $hand_right_keypoints_3d
## list()
```

```
unlist(listkeyPoints[[9]]) # There is nothing inside
```

```
## NULL
```

```
unlist(listkeyPoints[[1]]) # But here there is something
```

```
## [1] -1
```

Pero, ¿Tiene valor el dato que contiene? En realidad no, esta variable se obtiene por el número de listas que contiene `people`; si `people` tiene 4 listas son 4 las personas detectadas en el fotograma, si es una lista solo hay una persona detectada en el fotograma...

```
rawData[[1]][1]
```

```
## $person_id
## $person_id[[1]]
## [1] -1
```

```
for (i in 1:length(rawData)) {
 print(unlist(rawData[[i]][1]))
}
```

```
## person_id
##        -1
## person_id
##        -1
## person_id
##        -1
## person_id
##        -1
```

Por el momento `person_id` es mejor mantenerla en los datos, pero hay que tener en cuenta que en algún momento hay que eliminarla, simplemente que es mejor obtener todos los datos que contiene el archivo en un solo objeto. Un `dataFrame` es el objeto escogido en este caso, pues el objetivo final es construir este tipo de objeto; el problema es que este `dataFrame` (`dfPoints`) presenta todavía una estructura incipiente, aunque mediante un `loop` es posible añadir la variable

`people` teniendo en cuenta que hay tantas personas detectadas por el *software* como listas contiene la lista `people` (Figura 2).

```
dfPoints=NULL
  for (id in 1:length(rawData)) {

    points<-data.frame(unlist(rawData[[id]]),people=id)

    dfPoints=rbind(dfPoints,points)

  }
```

El resultado es un cuadro donde la categoría *keyPoints* aparece como nombre de las filas, los datos se encuentran amontonados en un vector y una tercera variable se puede apreciar: `people`, que identifica la lista a la que pertenece cada observación (Cuadro 3).

```
  knitr::kable(data.frame(rbind(head(dfPoints), tail(dfPoints))),caption = "`dataFrame` sin estrctura
```

Cuadro 3: `dataFrame` sin estrcturar a partir de los datos sacados de las listas.

	unlist.rawData..id...	people
person_id	-1.000000	1
pose_keypoints_2d1	473.913000	1
pose_keypoints_2d2	142.047000	1
pose_keypoints_2d3	0.872267	1
pose_keypoints_2d4	493.972000	1
pose_keypoints_2d5	188.946000	1
hand_right_keypoints_2d583	0.000000	4
hand_right_keypoints_2d593	0.000000	4
hand_right_keypoints_2d603	0.000000	4
hand_right_keypoints_2d613	0.000000	4
hand_right_keypoints_2d623	0.000000	4
hand_right_keypoints_2d633	0.000000	4

2.2 Ordenacion de los datos y uso de la función `split()` para crar un dataframe definitivo

El dato introducido por `id_person` debe ser eliminado, además el resto de datos deben de ordenarse correctamente en filas y columnas, además de ser necesario establecer las variables con las que constará el `dataFrame` final. Como ya se ha definido las variables que han de conformar el `dataFrame` final (Cuadro 2) el siguiente paso será generar y ordenar dichas variables.

Para empezar es necesario crear un patrón de corte que permita dividir en tres vectores correspondientes a las variables (`x`, `y`, `c`) los datos extraídos. Cada observación (`keyPoint`) se compone de 3 variables que han de ser separadas (Cuadro 3). Otro factor a tener en cuenta es el número de puntos del que se compone cada lista: 137 puntos.

"*OpenPose extracts 137 keypoints. 25 keypoints represent the body pose, 70 are facial keypoints, and there are 21 keypoints per hand representing the hand pose*" (De Coster, Van Herreweghe, y Dambre 2020).

```
 pattern<- sample(T, size=137*3, replace= T) # points triplicates


  pattern<-c(F,pattern)
```

```
  summary(pattern)
```

```
##    Mode   FALSE    TRUE
## logical       1     411
```

`Pattern` es un vector Booleano o lógico que selecciona los 411 `keyPoints` de una persona detectada en el fotograma $(137 * 3)$ y deja fuera la primera observación que se corresponde con `id_person` repitiéndose el patrón tantas veces como personas aparezcan en el fotograma `length(pattern) * lenght(people)`; por supuesto, tiene que coincidir con el número de observaciones del `dataFrame` (`dfPoints`).

```
nrow(dfPoints)==length(rawData)*length(pattern)
```

```
## [1] TRUE
```

```
if(!is.null(dfPoints)){
    pattern<-rep(pattern, times=max(dfPoints$people))
  }

summary(pattern)
```

```
##    Mode   FALSE    TRUE
## logical       4    1644
```

```
which(pattern==FALSE)
```

```
## [1]    1  413  825 1237
```

```
idPosition<-c()
for (i in 1:max(dfPoints$people)) {
  print((i-1)*412+1)
  idPosition[i]<-(i-1)*412+1
}
```

```
## [1] 1
## [1] 413
## [1] 825
## [1] 1237
```

```
rownames(dfPoints)[idPosition]
```

```
## [1] "person_id"  "person_id1" "person_id2" "person_id3"
```

```
idPosition==which(pattern==FALSE)
```

```
## [1] TRUE TRUE TRUE TRUE
```

El patrón creado permite eliminar `person_id` dejando solo los valores correspondientes a las variables espaciales (`x` e `y`) y al grado de confianza (`c`) implicando que ahora el número de observaciones es múltiplo de 412 o $137 * 3$:

$$\frac{nrow(dfPoints)}{max(people) * 3} = 137$$

Si no se cumple la igualdad es que el `dataFrame` no presenta el número correcto de `keyPoints`

```
dfPoints<-dfPoints[pattern,]

nrow(dfPoints)/(max(dfPoints$people)*3)==137
```

```
## [1] TRUE
```

Variable	Constante en ...
`frame`	fotograma
`words`	vídeo
`name`	vídeo

Cuadro 4: constancia de lasvariables obtenidas del nombre

```
magicNumber<-c()
for (i in 1:nrow(dfPoints)) {

 magicNumber[i]<-nrow(dfPoints)/(max(dfPoints$people)*3)==i




}


which(magicNumber==T) ## corroboration 137 is the only valid point
```

```
## [1] 137
```

El nombre de las observaciones es la variable `typePoint` que consiste en un factor que identifica al conjunto (*pose*, *face* , *right hand* y *left hand*) que pertenece la observación. El problema es que al tratarse del nombre de las observaciones añade un carácter numérico al final para diferenciar cada observación del resto, pero si se elimina se puede obtener una variable categórica (factor).

```
head(rownames(dfPoints))
```

```
## [1] "pose_keypoints_2d1" "pose_keypoints_2d2" "pose_keypoints_2d3"
## [4] "pose_keypoints_2d4" "pose_keypoints_2d5" "pose_keypoints_2d6"
```

```
type<-rownames(dfPoints)
  type<-gsub("_2d[0-9]*", "", type)

  levels(as.factor(type)) # categories of points
```

```
## [1] "face_keypoints"       "hand_left_keypoints"  "hand_right_keypoints"
## [4] "pose_keypoints"
```

Como se observa en el Cuadro 2 también hay variables dependientes del nombre del archivo que debe estar correctamente escrito siguiendo el formato del buscador de la UCLA (**aclarar este punto**). `words` corresponde a las palabras que acompañan al gesto durante el vídeo, siendo constante para todos los fotogramas de un vídeo; `frame` es el número de fotograma dentro del vídeo y se obtiene del archivo generado por `OpenPose`, y `name` que es el nombre del vídeo. Todas estas variables se obtienen mediante la función `gsub()`(R Core Team 2021). Además destacar que en el Cuadro 4 se observa para que estructura las observaciones de las variables obtenidas del nombre del archivo son constantes.

```
words<-gsub("_000000.*", "", "/data/home/agora/data/trainningDataVideos/videosJSON//2014-10-10_0600_U

# If there are more than 999 999 frames a 0 must be remove from  "_000000.*"
words<-gsub(".*[0-9]_", "", words)

# A digit[0-9] and a "_" must be exactly before the words

words
```

```
## [1] "from_beginning_to_end"
```

```
frame<-gsub(paste(".*", words,"_", sep = ""), "", "/data/home/agora/data/trainningDataVideos/videosJS


# R recognize it without problem and it permits remove the upper folders names

  frame<-as.numeric(gsub("_.*","", frame))

  frame
```

```
## [1] 128
```

```
# Important! If the video folder is a subfolder the separator must be //
name<-gsub(paste(".*//", sep = ""), "", "/data/home/agora/data/trainningDataVideos/videosJSON//2014-1

name<-gsub(paste("/.*", sep = ""), "", name)

name
```

```
## [1] "2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_from_beginning_to_end"
```

Tan solo queda una variable por crear, `points`, que se corresponde con los puntos para cada categoría (`keyPoints`), pero esta variable se genera a la vez que se dividen las observaciones entre 3 para obtener las dos variables espaciales y el grado de confianza. La función utilizada es `split()` que permite dividir un vector en grupos, en este caso las 3 variables (R Core Team 2021).

El patrón `triplet` es creado con el fin de reducir a un tercio el tamaño de las variables `people` y `typePoints` puesto que al formar parte del `dataFrame` sin estructurar (`dfPoints`) se encuentran por triplicado.

```
length(type)
```

```
## [1] 1644
```

```
  triplet<-c(T,F,F)
  length(type[triplet])
```

```
## [1] 548
```

```
 groups <- c("x", "y", "c") # the variables of the final df

 ## the length is the same when `triplet`  or `split()` is applied
 nrow(data.frame(split(dfPoints[,1], f = groups))) ==  length(type[triplet])
```

```
## [1] TRUE
```

La última variable, `points`, se genera directamente dentro del `dataFrame` resultante, teniendo en cuenta que : "*OpenPose extracts 137 keypoints. 25 keypoints represent the body pose, 70 are facial keypoints, and there are 21 keypoints per hand representing the hand pose*" (De Coster, Van Herreweghe, y Dambre 2020). `points` consiste en la repetición del siguiente vector: `c(0:24,0:69,0:20,0:20)`[1].

En añadido,hay que tener en cuenta que se puede dar el caso que en un fotograma concreto no haya ninguna persona detectada generando un `dataFrame` vacío, por ello se introduce una declaración (*if else*), así en caso de ser un fotograma sin personas detectadas el resultado de la función (`frameMaker`) es nulo (`NULL`) pero no produce error.

```
if(!is.null(dfPoints)){
    pointsDF<-data.frame(split(dfPoints[,1], f = groups),
                         people=dfPoints$people[triplet],
```

[1] El primer punto es el 0.

```
                        typePoint=  type[triplet],
                        point= c(0:24,0:69,0:20,0:20),
                        words=words,
                        frame=frame,
                        name=name) # split in 3 columns

  }else{
    pointsDF=NULL
  }
```

```
library(kableExtra)
```

```
## Warning in !is.null(rmarkdown::metadata$output) && rmarkdown::metadata$output
## %in% : 'length(x) = 2 > 1' in coercion to 'logical(1)'
```

```
caption<-"Cuadro resultante al aplicar la función.\\label{resultado}"
landscape(kable(head(pointsDF,n = 40), caption = caption, booktabs=T),margin = "0.1cm")
```

Cuadro 5: Cuadro resultante al aplicar la función.

c	x	y	people	typePoint	point	words	frame	name
0.872267	473.913	142.047	1	pose_keypoints	0	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.819151	493.972	188.946	1	pose_keypoints	1	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.656654	450.026	192.722	1	pose_keypoints	2	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.601400	441.387	258.741	1	pose_keypoints	3	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.345973	417.479	257.786	1	pose_keypoints	4	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.684274	537.047	188.915	1	pose_keypoints	5	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.772094	553.317	263.544	1	pose_keypoints	6	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.702320	507.385	234.833	1	pose_keypoints	7	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.426136	503.580	310.398	1	pose_keypoints	8	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.347794	471.997	311.343	1	pose_keypoints	9	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.000000	0.000	0.000	1	pose_keypoints	10	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.000000	0.000	0.000	1	pose_keypoints	11	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.395752	535.130	309.437	1	pose_keypoints	12	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.000000	0.000	0.000	1	pose_keypoints	13	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.000000	0.000	0.000	1	pose_keypoints	14	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.951639	470.076	134.387	1	pose_keypoints	15	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.939812	486.343	134.393	1	pose_keypoints	16	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.000000	0.000	0.000	1	pose_keypoints	17	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.942140	508.347	141.081	1	pose_keypoints	18	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.000000	0.000	0.000	1	pose_keypoints	19	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.000000	0.000	0.000	1	pose_keypoints	20	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.000000	0.000	0.000	1	pose_keypoints	21	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.000000	0.000	0.000	1	pose_keypoints	22	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.000000	0.000	0.000	1	pose_keypoints	23	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.000000	0.000	0.000	1	pose_keypoints	24	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.762762	462.838	137.648	1	face_keypoints	0	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.801574	462.223	142.566	1	face_keypoints	1	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.729888	462.633	147.279	1	face_keypoints	2	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.772347	463.863	151.992	1	face_keypoints	3	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.759811	465.502	156.501	1	face_keypoints	4	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.780968	468.371	160.599	1	face_keypoints	5	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.707024	471.035	164.902	1	face_keypoints	6	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.775932	473.494	168.591	1	face_keypoints	7	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.683650	476.773	170.640	1	face_keypoints	8	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.643851	483.330	170.640	1	face_keypoints	9	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.552271	490.092	170.025	1	face_keypoints	10	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.530250	496.650	167.361	1	face_keypoints	11	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.564710	501.568	163.058	1	face_keypoints	12	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.600588	503.207	155.886	1	face_keypoints	13	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr
0.684969	503.207	148.919	1	face_keypoints	14	from_beginning_to_end	128	2014-10-10_0600_US_KCAL_Entertainment_Tonight_374-380_ID933_fr

```
summary(frameMaker("/data/home/agora/data/trainningDataVideos/videosJSON//2014-10-10_0600_US_KCAL_Ent
```

```
##     c              x              y             people        typePoint
##  Mode:logical   Mode:logical   Mode:logical   Mode:logical   Mode:logical
##  TRUE:548       TRUE:548       TRUE:548       TRUE:548       TRUE:548
##   point          words          frame          name
##  Mode:logical   Mode:logical   Mode:logical   Mode:logical
##  TRUE:548       TRUE:548       TRUE:548       TRUE:548
```

2.3 Ultimas comprobaciones de la función

Una vez los datos de un fotograma presentan estructura la función puede ser repetida en cualquier otro fotograma hasta crear el vídeo entero pero antes de ello es necesario realizar una serie de comprobaciones finales para garantizar que toda la metodología empleada es correcta y no se produce ningún error a la hora de estructurar los datos.

```
summary(pointsDF)
```

```
##        c                x               y              people
##  Min.   :0.0000   Min.   :  0.0   Min.   :  0.0   Min.   :1.00
##  1st Qu.:0.0000   1st Qu.:  0.0   1st Qu.:  0.0   1st Qu.:1.75
##  Median :0.5852   Median :207.0   Median :137.3   Median :2.50
##  Mean   :0.4641   Mean   :231.0   Mean   :126.5   Mean   :2.50
##  3rd Qu.:0.7873   3rd Qu.:376.8   3rd Qu.:160.6   3rd Qu.:3.25
##  Max.   :1.0212   Max.   :553.3   Max.   :311.3   Max.   :4.00
##   typePoint             point          words               frame
##  Length:548         Min.   : 0.00   Length:548         Min.   :128
##  Class :character   1st Qu.: 8.00   Class :character   1st Qu.:128
##  Mode  :character   Median :17.00   Mode  :character   Median :128
##                     Mean   :22.88                      Mean   :128
##                     3rd Qu.:35.00                      3rd Qu.:128
##                     Max.   :69.00                      Max.   :128
##      name
##  Length:548
##  Class :character
##  Mode  :character
##
##
##
```

```
summary(as.factor(pointsDF$typePoint))/c(70,21,21,25) ## typePoint/Points vector
```

```
##       face_keypoints  hand_left_keypoints hand_right_keypoints
##                    4                    4                    4
##       pose_keypoints
##                    4
```

```
#the result is equal to people's number

summary(as.factor(pointsDF$typePoint))/c(70,21,21,25)== id
```

```
##       face_keypoints  hand_left_keypoints hand_right_keypoints
##                 TRUE                 TRUE                 TRUE
##       pose_keypoints
##                 TRUE
```

- Intervalo de confianza (μ), existe un valor que no está comprendido entre 0–1, pero esto no tiene porque ser un error, hay que conocer la función que calcula μ y ver si esto es posible o no.
- x, y coinciden en longitud y poco más se puede comprobar de ellas en este momento.
- people, typePoint, point, se encuentran definidas por la siguiente función:

$$\sum_{k=1} typePoint_k/points_k = max(person)$$

* El resto de variables (words, frame, name) provienen del nombre y son siempre iguales para un mismo fotograma (Cuadro 4).

3 `videoMaker` del fotograma al vídeo

3.1 Presentación de la función

Un vídeo analizado con `OpenPose` puede contener cientos o miles de fotogramas y es necesario poder ensamblar todos los fotogramas en un solo archivo que se corresponda con el vídeo completo; es por ello que `videoMaker` repite la función para todos los fotogramas de un vídeo y genera un archivo `.csv` correspondiente al mismo.

```
videoMaker<- function(video.folder,output.folder,save.csv,return.empty) {

  files<-list.files(video.folder, pattern="*.json", full.names=TRUE)

  out=NULL
  emptyFrames=NULL



  frameMaker<<-function(file){
    require(jsonlite)

    rawData<-read_json( path = file)

    rawData<-rawData[2]

    rawData<-rawData[[1]]

    dfPoints=NULL
    for (id in 1:length(rawData)) {

    if(length(rawData)!=0) {

    points<-data.frame(unlist(rawData[[id]]),people=id)

    dfPoints=rbind(dfPoints,points) }else{

      empty<-paste(file)


    }}


    pattern<- sample(T, size=137*3, replace= T) # points triplicates
```

```
  pattern<-c(F,pattern)

  if(!is.null(dfPoints)){
    pattern<-rep(pattern, times=max(dfPoints$people))
  }



  dfPoints<-dfPoints[pattern,]
  ##

  type<-rownames(dfPoints)
  type<-gsub("_2d[0-9]*", "", type)

  ###
  words<-gsub("_000000.*", "", file)


  words<-gsub(".*[0-9]_", "", words)


  ###

  frame<-gsub(paste(".*_000", sep = ""), "", file)

  frame<-as.numeric(gsub("_.*","", frame))
  ###

  name<-gsub(paste(".*//", sep = ""), "", file)

  name<-gsub(paste("/.*", sep = ""), "", name)

  ###

  triplet<-c(T,F,F)
  ###
  groups <- c("x", "y", "c") # the variables of the final df
  if(!is.null(dfPoints)){
    pointsDF<-data.frame(split(dfPoints[,1], f = groups),
                         people=dfPoints$people[triplet],
                         typePoint=  type[triplet],
                         point= c(0:24,0:69,0:20,0:20),
                         words=words,
                         frame=frame,
                         name=name) # split in 3 columns
    return(pointsDF)
  }else{
   return(empty)
  }


}


for (i in 1:length(files)){
```

```
    c <- data.frame(frameMaker(files[i]))

    if (length(c)!=1) {
       out=rbind(out,c)
    }else{

      emptyFrames=rbind(emptyFrames,c)
    }



  }

  if (save.csv==T) {
    folder<-output.folder
    dir.create(folder,recursive = T)
    write.csv(x = out, paste(folder,"/",unique(out$name),".csv",sep = ""),row.names = F)

  }

  if (return.empty==T) {
    video<-list(out,emptyFrames)
    return(video)
  }else{
    return(out)
  }




}
```

La función `videoMaker` aplica `frameMaker` a todo archivo con extensión `.json` dentro de una carpeta, por ello es importante que todos los archivos de un vídeo se encuentren en misma carpeta y no aparezcan fotogramas de otro vídeo en dicha carpeta. De darse el caso, se produciría un error si `save.csv=T` pues a la hora de crear el archivo `.csv` no existiría un solo nombre que es lo que requiere (`unique(out$name)`); pero si se diera el caso contrario (`save.csv=F`) no se produciría error aunque el `dataFrame` resultante sería erróneo . Por lo demás la función ensambla fotogramas sin importar la cantidad de los mismos.

Eso si es imprescindible tener en cuenta que la carpeta donde se encuentran los archivos tiene que ser introducida de un modo especifico. Es muy importante que la carpeta del vídeo con respecto a las carpetas superiores (*upper folder*) sea separada con `//` y que no se introduzca nada detrás del nombre de dicha carpeta (Figura 3).

$$WorkingDirectory/(n)subfolders//videoFolder$$

Figura 3: Introducción del nombre de la carpeta correspondiente al vídeo

```
videoMaker(video.folder = "/data/home/agora/data/trainningDataVideos/videosJSON//2007-01-05_0100_US_K
```

```
videoMaker(video.folder = "/data/home/agora/data/trainningDataVideos/videosJSON//2007-01-05_0100_US_K
```

```
## Warning in dir.create(folder, recursive = T): 'prueba' already exists
```

El atributo `out.folder` se corresponde a la carpeta donde se guardan los archivos `.csv` creados. Este atributo solo tiene sentido en caso de establecer `save.csv = T`, el caso opuesto (`save.csv = F`) genera el `dataFrame` dentro del `enviromental` pero no guarda el archivo `csv`. Hay que tener en cuenta lo siguiente,el atributo `out.folder` tiene como objetivo obtener una carpeta con archivos `.csv`, en la cual cada archivo `.csv` se corresponde con una carpeta de archivos `.json` o vídeo y solo es útil si se desea guardar el archivo `.csv` para su uso futuro.

Al usar el atributo `out.folder` de existir ya la carpeta indicada se producirá un `warning()` indicando que el objeto (carpeta) ya existe, de lo contrario creará la carpeta y guardará dentro los archivos `.csv`. Igualmente, a pesar de que un `warning()` suele preceder a un error, no siempre es así; en este caso el `warning()` solo indica que la carpeta donde se guardan los archivos ya existe. Por si acaso mejor asegurarse bien que la ruta (*path*) es correcta, de otro modo los `.csv` pueden guardarse en cualquier sitio del *working directory* o carpeta del proyecto `R` en el que estamos trabajando (Ver Figura 4). .

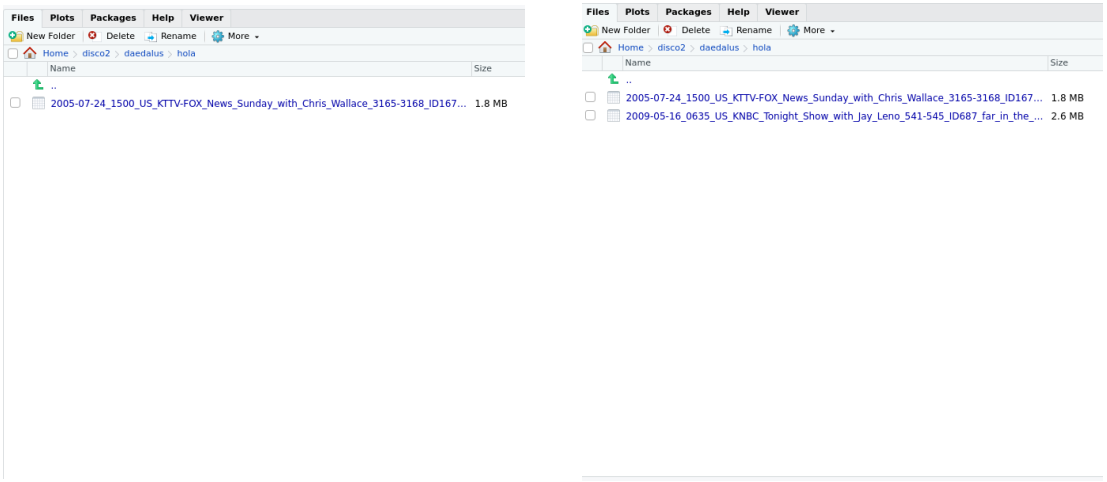

Figura 4: `output.folder` antes (izquierda) y despúes (derecha) de aplicar `videoMaker`.

3.2 Ejemplo

```
videoMaker(video.folder = "/data/home/agora/data/trainningDataVideos/videosJSON//2009-05-16_0635_US_K
         save.csv = F,return.empty = F)->sample
```

Al usar la función el resultado es un `dataFrame` con las variables de `frameMaker` pero que en el caso de `frame` se obtiene un vector de números naturales —además del 0— indicando el fotograma y las variables c, x, y de cada fotograma (`dataFrame`) se ensamblan una detrás de otra, dando como resultado un `dataFrame` con todas las observaciones.

$$nrow(df)/137 = max(frame)$$

De hecho si se comprueba en `R` hay que sumar uno al valor máximo puesto que el primer fotograma se ha etiquetado como `0` pero sigue ocupando la posición `1` del vector (Figura 5).

```
str(sample)
```

```
## 'data.frame':    16440 obs. of  9 variables:
##  $ c         : num  0.874 0.722 0.65 0.536 0.698 ...
##  $ x         : num  271 269 215 179 209 ...
##  $ y         : num  68.9 120.9 113.6 157.4 152.8 ...
##  $ people    : int  1 1 1 1 1 1 1 1 1 ...
##  $ typePoint: chr  "pose_keypoints" "pose_keypoints" "pose_keypoints" "pose_keypoints" ...
```

```
##  $ point    : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ words    : chr  "far_in_the_future" "far_in_the_future" "far_in_the_future" "far_in_the_future"
##  $ frame    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ name     : chr  "2009-05-16_0635_US_KNBC_Tonight_Show_with_Jay_Leno_541-545_ID687_far_in_the_fu
```

```
# Validación
nrow(sample)/137==length(as.numeric(levels(as.factor(sample$frame))))
```

```
## [1] TRUE
```

```r
sample$frame->x


x=x+1 ## Just for the meme

nrow(sample)/137==max(x)
```

```
## [1] TRUE
```

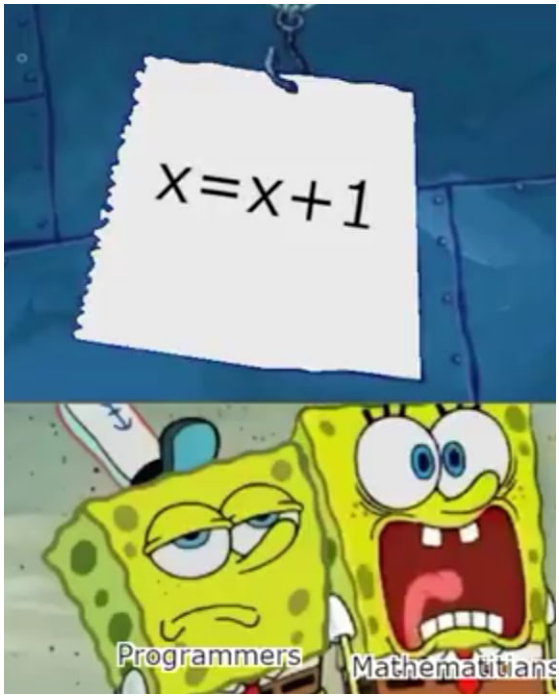

Figura 5: El número `0` no pertenece a los naturales, por ello se suma 1 a cada valor del vector

Por lo demás esta función se puede usar a través de un *loop* que genere un `dataFrame` — y los `.csv` en caso de indicarse— compuesto de varios vídeos a analizar, aunque `dfMaker` se encarga de ello automáticamente (Figura 1).

4 dfMaker cuando un solo vídeo no es suficinete

```
dfMaker <- function(video.folders,save.csv=F, output.folder,return.empty=F) {


 video.folders<-list.dirs(video.folders, full.names=TRUE,recursive = F)


videoMaker<<- function(video.folders,output.folder,save.csv,return.empty) {



  files<-list.files(video.folders, pattern="*.json", full.names=TRUE)

  out=NULL
  emptyFrames=NULL



  frameMaker<<-function(file){
    require(jsonlite)

    rawData<-read_json( path = file)

    rawData<-rawData[2]

    rawData<-rawData[[1]]

    dfPoints=NULL
    for (id in 1:length(rawData)) {

    if(length(rawData)!=0) {

    points<-data.frame(unlist(rawData[[id]]),people=id)

    dfPoints=rbind(dfPoints,points) }else{

      empty<-paste(file)


    }}


    pattern<- sample(T, size=137*3, replace= T) # points triplicates


    pattern<-c(F,pattern)

    if(!is.null(dfPoints)){
      pattern<-rep(pattern, times=max(dfPoints$people))
    }



    dfPoints<-dfPoints[pattern,]
    ##
```

```
  type<-rownames(dfPoints)
  type<-gsub("_2d[0-9]*", "", type)

  ###
  words<-gsub("_0000.*", "", file)


  words<-gsub(".*[0-9]_", "", words)


  ###

  frame<-gsub(paste(".*_000", sep = ""), "", file)

  frame<-as.numeric(gsub("_.*","", frame))
  ###

  name<-gsub(paste("_",words,".*", sep = ""), "", file)
  name<-gsub(paste(".*/",sep = ""), "", name)

  ###

  triplet<-c(T,F,F)
  ###
  groups <- c("x", "y", "c") # the variables of the final df
  if(!is.null(dfPoints)){
    pointsDF<-data.frame(split(dfPoints[,1], f = groups),
                         people=dfPoints$people[triplet],
                         typePoint=  type[triplet],
                         point= c(0:24,0:69,0:20,0:20),
                         words=words,
                         frame=frame,
                         name=name) # split in 3 columns
    return(pointsDF)
  }else{
   return(empty)
  }


}


for (i in 1:length(files)){

  c <- data.frame(frameMaker(files[i]))

  if (length(c)!=1) {
     out=rbind(out,c)
  }else{

    emptyFrames=rbind(emptyFrames,c)
  }
```

```
  }

  if (save.csv==T) {
    folder<-output.folder
    dir.create(folder,recursive = T)
    write.csv(x = out, paste(folder,"/",unique(out$name),".csv",sep = ""),row.names = F)

  }

  if (return.empty==T) {
    video<-list(out,emptyFrames)
    return(video)
  }else{
    return(out)
  }




}



 result=NULL
 dFinal<-NULL
 totalEmpty<-NULL

 for (i in 1:(length(video.folders))) {


  dfVideo<-videoMaker( video.folders[i],save.csv = save.csv,output.folder = output.folder,return.empt

  if (return.empty==F) {

    result=rbind(result,dfVideo)
  }else{

            dFinal=rbind(dFinal,dfVideo[[1]])
            totalEmpty=rbind(totalEmpty,dfVideo[[2]])
            result<-list(dFinal,totalEmpty)

  }


}

 return(result)
}
```

`dfMaker` presenta los mismo atributos excepto que ahora en vez de ser `video.folder` el nombre de la ruta es en plural (`video.folders`), por lo demás la función aplica `videoMaker` sobre todas las carpetas o vídeos en un directorio o carpeta superior (*upper folder*). La única regla a la hora de definir la ruta de las carpetas deriva de `videoMaker` y consiste en terminar la ruta con `/` (Figura 6).

```
sample3<-dfMaker(video.folders = "/data/home/agora/data/trainningDataVideos/videosJSON/",
                 save.csv = F,return.empty = F,)
```

WorkingDirectory/(n)subfolders/

Figura 6: Definición general del atributo `video.folders` en la función `dfMaker` .

```
  head(sample3)
```

```
##           c        x       y people      typePoint point     words frame
## 1 0.8145960 244.9780 114.554      1 pose_keypoints     0 back_then     0
## 2 0.4568650 214.8850 221.260      1 pose_keypoints     1 back_then     0
## 3 0.3252010  91.7276 216.700      1 pose_keypoints     2 back_then     0
## 4 0.0586367  69.8167 295.159      1 pose_keypoints     3 back_then     0
## 5 0.0000000   0.0000   0.000      1 pose_keypoints     4 back_then     0
## 6 0.2942320 338.9420 217.615      1 pose_keypoints     5 back_then     0
##                                                                    name
## 1 2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
## 2 2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
## 3 2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
## 4 2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
## 5 2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
## 6 2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
```

```
landscape(kable(sample3[c(100:102,745:747,10000:10002,50154:50156,1000032:1000034),], booktabs=T,capt
  kable_styling(latex_options="scale_down")
```

Cuadro 6: Cuadro resultante al aplicar dfMaker

	c	x	y	people	typePoint	point	words	frame	name
100	0.000000	0.000	0.0000	1	hand_left_keypoints	4	back_then	0	2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
101	0.000000	0.000	0.0000	1	hand_left_keypoints	5	back_then	0	2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
102	0.000000	0.000	0.0000	1	hand_left_keypoints	6	back_then	0	2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
745	0.883901	242.342	130.1320	1	face_keypoints	34	back_then	5	2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
746	0.876275	247.614	128.2140	1	face_keypoints	35	back_then	5	2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
747	0.926324	205.917	99.4574	1	face_keypoints	36	back_then	5	2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
10000	0.000000	0.000	0.0000	1	hand_right_keypoints	19	back_then	72	2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
10001	0.000000	0.000	0.0000	1	hand_right_keypoints	20	back_then	72	2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
10002	0.779719	241.334	115.4370	1	pose_keypoints	0	back_then	73	2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167
50154	0.000000	0.000	0.0000	1	pose_keypoints	11	back_then	69	2006-10-12_0600_US_KCET_Tavis_Smiley_172-177_ID166
50155	0.000000	0.000	0.0000	1	pose_keypoints	12	back_then	69	2006-10-12_0600_US_KCET_Tavis_Smiley_172-177_ID166
50156	0.000000	0.000	0.0000	1	pose_keypoints	13	back_then	69	2006-10-12_0600_US_KCET_Tavis_Smiley_172-177_ID166
1000032	0.854043	404.853	140.2300	1	face_keypoints	43	earlier_than	30	2015-04-22_1530_FR_KCET_France_24_604-608_ID460
1000033	0.829732	407.444	139.8840	1	face_keypoints	44	earlier_than	30	2015-04-22_1530_FR_KCET_France_24_604-608_ID460
1000034	0.754289	409.863	140.0570	1	face_keypoints	45	earlier_than	30	2015-04-22_1530_FR_KCET_France_24_604-608_ID460

4.1 Últimas comprobaciones

Al ensamblarse los `dataFrames` generados con `dfMaker` el número resultante de observaciones ha de ser divisible en todos los casos entre `137`. Más allá de ahí es difícil establecer generalizaciones a todos los `dataFrames`producidos por `dfMaker`, no todos los fotogramas presentan el mismo número de personas (`people`) y no todos los vídeos tienen el mismo número de fotogramas.

A pesar de no poder definir matemáticamente otras comprobaciones si es posible detectar si existe algún error en en la variable people; porque los valores más bajos tiene que ser los que más veces se repitan, dicho de otra forma para detectar a una segunda persona en un fotograma hay que detectar a otra antes (Ley de Benford).

La regla anterior no es aplicable a frame debido a que uno o varios fotogramas —incluso los primeros— pueden no presentar a ninguna persona detectada y al tratarse de conjuntos vacíos no se encuentran incluidos en el `dataFrame` final. Por último, destacar el número de observaciones resultantes que en solo 52 vídeos es de: 1083670.

```
str(sample3)
```

```
## 'data.frame':    1083670 obs. of  9 variables:
##  $ c        : num  0.8146 0.4569 0.3252 0.0586 0 ...
##  $ x        : num  245 214.9 91.7 69.8 0 ...
##  $ y        : num  115 221 217 295 0 ...
##  $ people   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ typePoint: chr  "pose_keypoints" "pose_keypoints" "pose_keypoints" "pose_keypoints" ...
##  $ point    : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ words    : chr  "back_then" "back_then" "back_then" "back_then" ...
##  $ frame    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ name     : chr  "2005-07-24_1500_US_KTTV-FOX_News_Sunday_with_Chris_Wallace_3165-3168_ID167" "2
```

```
nrow(sample3)/137 ## nrow is multiple of 137
```

```
## [1] 7910
```

```
length(unique(sample3$name)) ## number of total videos in dataFrame
```

```
## [1] 379
```

```
table(as.factor(sample3$people))/137 ## people by frame
```

```
##
##    1    2    3    4    5    6    7    8    9   10
## 6186 1138  350  136   54   20   14    9    2    1
```

```
table(as.factor(sample3$frame))/137 ## frames by video
```

```
##
##   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
##  62  60  58  59  60  60  59  58  58  59  59  60  59  59  59  61  61  64  62  62
##  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39
##  64  62  63  62  62  64  63  63  62  63  61  62  62  62  60  64  65  63  65  67
##  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59
##  65  67  66  67  68  66  67  67  67  65  67  66  67  67  69  69  71  69  71  68
##  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79
##  68  66  66  64  63  64  65  63  62  64  63  61  60  62  63  62  62  61  63  62
##  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
##  61  61  62  61  62  62  61  60  63  63  43  43  42  44  44  44  44  43  44  43
## 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
##  42  41  42  41  44  45  46  45  45  45  45  45  45  45  45  46  45  45  46  45
## 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
##  16  16  15  16  16  15  16  19  19  19  19  19  19  19  19  19  19  18  19  19
## 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
```

```
##  18  19  18  18  18  18  18  18  19  18   8   8   9   9   8   9   9   9  10  10
## 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
##  13  13  12  13  11  17  14  13  11  12  13  13  13  12  10  11  10  11  11  11
## 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## 200 201 202 203 204 205 206 207 208 209
##   1   1   1   1   1   1   1   1   1   1
```

```
sort(table(as.factor(sample3$frame))/137,decreasing = T)[1] #most repeated frame
```

```
## 56
## 71
```

5 conclusión

En resumen, estas tres funciones permiten ordenar los datos generados con `openPose` según las necesidad, desde un solo fotograma hasta todos los vídeos que sean necesarios (Figura 1). Por lo demás, los datos están todavía crudos (*raw data*) y serán necearais trasformaciones para obtener datos modelizables (normalización, eliminación de observaciones vacías...), pero dependerá de la aplicación y uso que se pretenda hacer de los datos.

En principio, las funciones han de servir para cualquier archivo `.json` pero hay que tener en cuenta que las variables asociadas al nombre del archivo dependen de que se respete el sistema de nombrado de la UCLA (**Revisar**) (Figura 2). Si otro sistema de nombrado es utilizado las variables asociadas al nombre pueden dar valores `NA` o erróneos. En tal circunstancia es posible revisar los patrones de búsqueda para dichas variables dentro de `dfMaker`.

Todo lo realizado es código abierto y se puede utilizar libremente por el usuario, siempre y cuando no pretenda crear código cerrado a partir del mismo.

Referencias

Cao, Zhe, Gines Hidalgo, Tomas Simon, Shih-En Wei, y Yaser Sheikh. 2019. «OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields». *IEEE transactions on pattern analysis and machine intelligence* 43 (1): 172-86.

De Coster, Mathieu, Mieke Van Herreweghe, y Joni Dambre. 2020. «Sign language recognition with transformer networks». En *12th International Conference on Language Resources and Evaluation*, 6018-24. European Language Resources Association (ELRA).

Ooms, Jeroen. 2014. «The jsonlite Package: A Practical and Consistent Mapping Between JSON Data and R Objects». *arXiv:1403.2805 [stat.CO]*. https://arxiv.org/abs/1403.2805.

R Core Team. 2021. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.