



SISTEMAS INTELIGENTES PARA LA  
GESTIÓN EN LA EMPRESA  
MASTER PROFESIONAL EN INGENIERÍA EN INFORMÁTICA

## Práctica 2: Multclasificación

---

**Autores**

José Ángel Díaz García y Pablo Martin-Moreno

**Equipo**

José Ángel & Pablo

**Puntuación y Posición**

0.82297



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

---

Granada, Junio de 2017

# Índice general

<b>1. Introducción</b>	<b>5</b>
1.1. Problema y Dataset . . . . .	5
1.1.1. Evaluación . . . . .	6
1.2. Herramientas y objetivos . . . . .	6
1.2.1. Hardware . . . . .	6
1.2.2. Software . . . . .	7
1.2.3. Objetivos . . . . .	7
1.3. Organización del trabajo . . . . .	8
<b>2. Preprocesado</b>	<b>9</b>
2.1. Resize data . . . . .	9
2.2. Cut data . . . . .	11
2.3. Data Aumentation . . . . .	13
2.4. Undersampling . . . . .	15
<b>3. Clasificación con NN</b>	<b>17</b>
3.1. Intel Deeplearning SDK . . . . .	17
3.2. From Scratch . . . . .	18
3.2.1. Mxnet . . . . .	18
3.2.2. TensorFlow+Keras . . . . .	26
3.3. Fine Tunning . . . . .	30
Multclasificación: Cervical Cancer	1

<b>4. Multclasificación y mapas de características</b>	<b>31</b>
4.1. Multclasificacion . . . . .	31
4.2. Mapas de características . . . . .	31
<b>5. Conclusiones y vías futuras</b>	<b>32</b>
5.1. Conclusiones finales . . . . .	32
5.2. Vías futuras . . . . .	32
<b>6. ANEXO I</b>	<b>33</b>

# Índice de figuras

2.1. Deformación de la imagen al redimensionar. . . . .	11
2.2. Cortado de la imagen. . . . .	11

# Índice de tablas

1.1. Especificaciones técnicas de la máquina 1. . . . .	6
1.2. Especificaciones técnicas de la máquina 2. . . . .	7
6.1. Tabla de resultados . . . . .	34

# Capítulo 1

## Introducción

Esta última práctica está enmarcada dentro de la asignatura **Sistemas Inteligentes para La Gestión en la Empresa** del Master Profesional en Ingeniería Informática de la UGR y aborda un problema real de predicción multiclase en la plataforma Kaggle [2].

Este problema, es de un nivel avanzado, y a lo largo de los siguientes capítulos intentaremos aportar una solución aceptable en la plataforma Kaggle, así como estudiar y asentar los diferentes conceptos teóricos vistos en la asignatura.

### 1.1. Problema y Dataset

El problema en última instancia es un problema de clasificación multiclase real el cual deberá ser resuelto mediante técnicas de *deeplearning*. El problema, en concreto es Intel & MobileODT Cervical Cancer Screening [3] y trata de clasificar partiendo de imágenes del cervix de distintas pacientes, que tipo de tratamiento para el cancer es más efectivo, aspecto muy relevante sobre todo en puntos del mundo rural donde el acceso a grandes infraestructuras médicas puede estar limitado y donde la prevención en etapas tempranas puede ser decisiva.

El dataset está compuesto de la siguiente manera:

- **test:** 512 Imágenes que deberemos clasificar tras el entrenamiento.

- **train:** Tenemos un total de 1581 muestras para entrenar, compuestas por 350 de tipo 1, 781 de tipo 2 y 450 de tipo 3.
- **train-extra:** El dataset ofrece también una gran conjunto de imagenes extra para el entrenamiento de unos 30GB de espacio en disco.

Podemos ver como el dataset muestra cierto ratio de des balanceo, por lo que en instancias superiores de la práctica, deberemos atacar este punto para obtener mejores resultados. Por otro lado, las restricciones del problema hacen que sean interesantes propuestas como *One Vs One* o *One vs All* que analizaremos en sucesivos puntos.

### 1.1.1. Evaluación

La evaluación de esta práctica tendrá como evaluador del modelo la función *logloss*, donde los falsos negativos tendrán una gran penalización. En caso de obtener una evaluación perfecta, el *logloss* del clasificador sería **0**.

## 1.2. Herramientas y objetivos

En esta sección veremos una breve introducción a las herramientas usadas para el desarrollo de la práctica así como de los principales objetivos que se buscan conseguir con el desarrollo de la misma.

### 1.2.1. Hardware

Elemento	Características
Procesador	2,6 GHz Intel Core i5
GPU	-
Memoria Ram	8 GB 1600 MHz DDR3
Disco duro	SATA SSD de 120 GB

Tabla 1.1: Especificaciones técnicas de la máquina 1.

Elemento	Características
Procesador	Intel Core i7 6700HQ
GPU	2,6 GHz Intel Core i5
Memoria Ram	16 GB SDRAM
Disco duro	128GB SSD

Tabla 1.2: Especificaciones técnicas de la máquina 2.

### 1.2.2. Software

El software utilizado es en su práctica totalidad software libre, siendo el restante software propietario cuyas licencias vienen incluidas en el sistema operativo de la máquina 1.1 siendo este OS X "Sierra", o el Windows 10 de la máquina 1.2. El software usado es:

- **RStudio**: Entorno de trabajo para R.
- **Tensorflow**: Entorno de deeplearning sobre Python.
- **Keras**: Capa de abstracción sobre Tensorflow.
- **MXNet**: Librería de Deeplearning sobre R.
- **Atom**: Editor de texto plano para la programación de los scripts.
- **TeXShop**: procesador de textos basado en *Latex* usado para elaborar la documentación del presente proyecto.

### 1.2.3. Objetivos

Los objetivos de este trabajo podrían resumirse en los siguientes:

- Obtener un modelo predictivo fiable que dado una nueva imagen pueda predecir el tipo de tratamiento contra el cancer a aplicar.
- Obtener un valor de *LogLoss* aceptable para escalar posiciones en la competición de Kaggle.



- Comprender y estudiar las distintas técnicas de minería de datos vistas en la asignatura.
- Ahondar en el proceso de la multclasificación y las vertientes de estudio dentro de la misma.
- Estudiar distintos métodos de clasificación sobre el mapa de características proveniente del entrenamiento de las redes neuronales.

### 1.3. Organización del trabajo

La organización del presente documento, se centra en detallar cada uno de los pasos seguidos durante el estudio y resolución del problema planteado en esta introducción. En el capítulo 2 veremos los scripts y explicaciones asociadas al preprocesado de datos, más concretamente al data augmentation y cierto enfoque de *imbalance learning*. En el capítulo 3 tendremos el grueso del trabajo en el que entraremos en detalle en una primera aproximación *from scratch* con *mxnet* y el grueso del trabajo realizando *fine tuning* en Tensorflow. Finalizaremos con el capítulo 4 dedicado al estudio de multclasificación y uso de otros clasificadores y por último las conclusiones y vías futuras que quedan relegadas al capítulo 5.

Los resultados obtenidos en la competición de kaggle pueden encontrarse en el Anexo de la sección 6.

# Capítulo 2

## Preprocesado

En este capítulo enunciaremos el proceso de preprocesado llevado a cabo durante la realización de la práctica. Cabe destacar, que por facilitar la comprensión todo el proceso se enuncia en este capítulo, pero hay ciertos puntos como el del proceso de undersampling (sección 2.4) y el de corte de las imágenes (sección 2.2) que surgen fruto de cambios e ideas en el proceso de aprendizaje y entrenamiento de las redes neuronales.

### 2.1. Resize data

Como veremos en el capítulo de conclusión, uno de los requisitos y mayores problemas que esta práctica a ofrecido han sido las limitaciones de memoria tanto de principal para computo como de memoria secundaria para almacenar todas las imágenes y sus diferentes versiones. Con el fin de poder manejar estas más eficientemente se ha realizado un resize de las mismas a tamaño de 256x256px, para ello hemos seguido el siguiente script en R.

---

```
1 #-----
2
3 # Clear workspace
4 rm(list=ls())
5
6 #-----
7 # Load and pre-process images
8 #-----
```

```
9
10 # Set run parameters parameters
11 test_img_path <- "."
12 width <- 256
13 height <- 256
14
15
16 #-----
17 # Load and pre-process train images
18 #-----
19
20 # Load EImage library
21 library(EImage)
22
23 # Load images into a dataframe
24 library(gsubfn)
25 img_file_list <- list.files(path = test_img_path, pattern = "*.jpg", full.names =
    TRUE, recursive = TRUE)
26
27 train_df <- data.frame()
28
29 for(i in 1:length(img_file_list)) {
30   img_file_name <- img_file_list[i]
31   #img_class <- strapplyc(img_file_list[i], ".*\\/Type_(.*)\\/")[1]
32   img <- readImage(img_file_name)
33   img_resized <- resize(img, w=width, h=height)
34
35   name <- paste("reducidas",img_file_name, sep="/")
36   dir.create("reducidas", showWarnings = FALSE)
37   writeImage(img_resized, name)
38
39 }
```

---

Con este simple script podemos preprocesar las imágenes originales y reducirías de tamaño de manera que son mas fácilmente manejables, pero esto nos da un problema asociado y es que estamos modificando los datos originales.

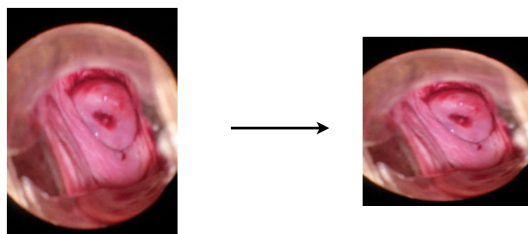


Figura 2.1: Deformación de la imagen al redimensionar.

Tal y como podemos apreciar en la figura 2.1 si una imagen rectangular la ‘obligamos’ a ser cuadrada, estamos deformando el contenido por lo que ya tendremos otra imagen distinta de la original, a este problema la estudiaremos en la siguiente sección.

## 2.2. Cut data

Siguiendo con el ejemplo de la sección anterior, la solución a la deformación de la imagen pasa por cortar al ratio de aspecto 1:1 y posteriormente redimensionar para tener imágenes de menos peso. Este paso, puede explicarse con la imagen 2.2.

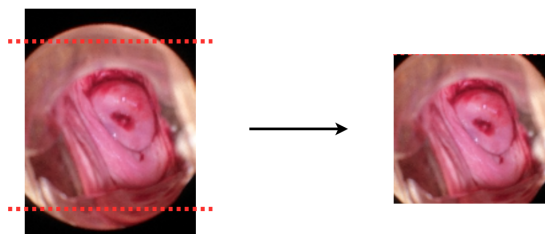


Figura 2.2: Cortado de la imagen.

La altura de la imagen debe ser por tanto igual al ancho de la misma y para ello, debemos quitar los espacios que vemos en la imagen anterior, tras lo cual tendremos una imagen cuadrada pero sin deformar el contenido, además de quitar partes de la imagen que en más del 90 % de los casos no tienen importancia. Para calcular el corte hemos usado la ecuación 2.1, que

nos da el valor que deberemos substraer de la imagen original tanto por la parte superior como por la inferior.

$$corte = \frac{height - width}{2} \quad (2.1)$$

El script que usamos para este objetivo es el siguiente:

---

```

1  #-----
2  # This script cut the images to prevent original data deformation
3  #-----
4
5
6  #setwd("D:/Facultad/Master/Segundo cuatrimestre/SIGE/PracticaFinal")
7  # Clear workspace
8  rm(list=ls())
9
10 # Set run parameters parameters
11 test_img_path <- "."
12
13 # Load EBIImage library
14 library(EBIImage)
15 # Load images into a dataframe
16 library(gsubfn)
17 img_file_list <- list.files(path = test_img_path, pattern = "*.jpg", full.names =
    TRUE, recursive = FALSE)
18
19 img_file_list
20 dir.create("crop", showWarnings = FALSE)
21 for(i in 1:length(img_file_list)) {
22
23   img_file_name <- img_file_list[i]
24   name <- paste("crop",img_file_name, sep="/")
25   print(name)
26   if(!file.exists(name)){
27     img <- readImage(img_file_name)
28     width <- dim(img)[1]
29     heigth <- dim(img)[2]
30     if (width < heigth){
31       cut <- (heigth-width)/2
32       p <- heigth-cut
33       img2 <- img[0:width,cut:p,]
34     }else{
35       cut <- (width-heigth)/2
36       p <- width-cut

```

```
37     img2 <- img[cut:p,0:height,]
38   }
39
40   writeImage(img2, name)
41 }
42
43 }
44 print("Finalizado")
```

---

## 2.3. Data Aumentation

Uno de los principales problemas en *deep learning* es la falta de datos. Para ello, pueden usarse técnicas de *data augmentation* que consisten en aplicar ligeras transformaciones a las imágenes para conseguir un conjunto de entrenamiento mayor. Para ello, hemos usado dos vertientes, una con R (siguiente script) con la que obtenemos imágenes modificadas en ficheros separados para poder ir combinándolas como deseemos, y por último, en el código Python que veremos en puntos siguientes, sobre estas volvíamos a aplicar una serie de cambios aumentando por tanto el data set bastante.

---

```
1
2 #-----
3
4 # Clear workspace
5 rm(list=ls())
6
7 #-----
8 # Load and pre-process images
9 #-----
10
11 # Set run parameters parameters
12 train_img_1_path <- "./Type_1"
13 train_img_2_path <- "./Type_2"
14 train_img_3_path <- "./Type_3"
15
16
17 #-----
18 # Load and pre-process train images
19 #-----
20
21 # Load EBImage library
```

```
22 library(EBImage)
23
24 # Load images into a dataframe
25 library(gsubfn)
26 img_file_1_list <- list.files(path = train_img_1_path, pattern = "*.jpg",
    full.names = TRUE, recursive = TRUE)
27 img_file_2_list <- list.files(path = train_img_2_path, pattern = "*.jpg",
    full.names = TRUE, recursive = TRUE)
28 img_file_3_list <- list.files(path = train_img_3_path, pattern = "*.jpg",
    full.names = TRUE, recursive = TRUE)
29
30
31 #train_df <- data.frame()
32
33 #Hacemos 3 bucles porque tienen tamaños distintos cada clase.
34
35 Imagenes<-function(image,type) {
36   img_file_name <- image
37
38   img <- readImage(img_file_name)
39
40   if(type=="flip"){
41     img <- flip(img)
42   }
43
44   if(type=="flop"){
45     img <- flop(img)
46   }
47
48   if(type=="rot"){
49     img = rotate(img, 90, bg.col = "white")
50   }
51   if(type=="trans"){
52     img_t = transpose(img)
53   }
54
55   #print("IMG_FILE_NAME")
56   #print(img_file_name)
57
58   salida <- substr(img_file_name, 0, 8)
59   name <- paste(salida,type, sep="/")
60   salida <- substr(img_file_name, 10,nchar(img_file_name) )
61   #print("salida")
62   #print(salida)
63   salida <- paste(type,salida, sep="-")
```

```
64
65  name2 <- paste(name,salida, sep="/")
66  #print("NAME2")
67  #print(name2)
68
69  dir.create(name, showWarnings = FALSE)
70  writeImage(img, name2)
71 }
72
73
74
75 tipos<-function(imagen){
76   Imagenes(imagen,"flip")
77   Imagenes(imagen,"flop")
78   Imagenes(imagen,"rot")
79   Imagenes(imagen,"trans")
80 }
81
82 for(i in 1:length(img_file_1_list)) {
83   tipos(img_file_1_list[i])
84 }
85
86
87 for(i in 1:length(img_file_2_list)) {
88   tipos(img_file_2_list[i])
89 }
90
91 for(i in 1:length(img_file_3_list)) {
92
93   tipos(img_file_3_list[i])
94 }
95 }
96
97 print("FINALIZADO")
```

---

## 2.4. Undersampling

Por último, en nuestro proceso de pre-procesado, descubrimos que las clases mostraban cierto desequilibrio. Concretamente, la clase1 frente a clase 3 de 1,66, y de clase 1 frente a clase 2 de 2,35. Aunque no es mucho, están por



encima del 1,5, valor a partir del cual se considera un problema de clase desequilibradas.

Para intentar solventar este problema, quitamos algunas imágenes de las clases mayoritarias intentando siempre que las imágenes originales estuvieran presentes entre las seleccionadas, de manera que se suprimieron algunas de las transformaciones obteniendo ratios de balanceo por debajo del 1.5. Esta solución, al evaluar en test, lejos de mejorar empeoraba el modelo ya que como hemos visto en 2.3 el principal problema a solventar en esta competición es la falta de información en training y con esta técnica, aunque igualamos clases, perdemos información que puede ser decisiva.

# Capítulo 3

## Clasificación con NN

En este capítulo veremos el proceso seguido y las distintas vertientes de entrenamiento usadas a lo largo de la realización de la práctica. Concretamente veremos entrenamiento *from scratch* con dos tecnologías distintas y por último *fine tuning*.

### 3.1. Intel Deeplearning SDK

Como el software se ve en la práctica y en virtud de probar todos y cada uno de los elementos estudiados, instalamos el software de Intel en la máquina 1.1. El primero de los problemas vino con el tiempo de instalación y la cantidad de memoria necesaria por este, por un lado para desplegar los 4 contenedores docker que montan la app, como para alojar los dataset, ya que crea una copia dentro de los contenedores de los mismos. Por otro lado, una vez instalado, constatamos la imposibilidad de validar el conjunto de test de la competición sin realizar modificaciones en los scripts de Caffe o Tensorflow o creando clases ficticias para que clasifique con las cuales el proceso de entrenamiento ofrecería resultados muy malos y la interfaz gráfica del mismo serviría mas bien de poco. La cantidad de problemas y la pérdida de tiempo con el software, hizo que migráramos directamente hacia otras tecnologías como las que veremos a continuación.

## 3.2. From Scratch

La estrategia seguida en una competición en la que el tiempo de computo era un factor claramente privativo, ha pasado por afinar el proceso de pre-procesado de datos con scripts que ofrecían resultados 'aceptables' en poco tiempo de computo. Por ello, nos hemos basado en redes neuronales muy sencillas, sobre las cuales podríamos ir probando distintas combinaciones de preprocesado, como las vistas en el capítulo 2 y ver cuales son las que mejor se adaptan a nuestro problema, con la premisa de si funcionan bien en una red sencilla también lo harán en redes mas complejas. Las redes elaboradas provienen de kernels de Kaggle, concretamente la de MXNET proviene del siguiente tutorial [4] y la de TensorFlow de este otro [5], aunque si bien el código ha sido modificado para adaptarlo a nuestros requisitos.

### 3.2.1. Mxnet

El siguiente script fue

---

```
1      #-----
2  # Kaggle Intel Cervix Cancer Challenge
3  #
4  #
5  # Image loading and basic pre-processing with EBImage
6  # Color images
7  # Submission to Kaggle is generated
8  #-----
9
10
11 #-----
12 #Load librarys
13 #-----
14
15
16 library(dplyr)
17 library(EBImage)
18 library(mxnet)
19 library(nnet)
20
21
22 #-----
23 # Load images using EBImage
```

```

24 #
25 # This loop resize all images, we dont have to do data-augmentation
26 # because, it is done with the script data-augmentation.R.
27 #-----
28
29
30 paths <- c("/Users/joseadiazg/Desktop/Temporales
           UGR/SIGE/all_data_resized/Type_1",
31           "/Users/joseadiazg/Desktop/Temporales
           UGR/SIGE/all_data_resized/Type_2",
32           "/Users/joseadiazg/Desktop/Temporales
           UGR/SIGE/all_data_resized/Type_3")
33
34
35 # Uncomment this if you want to load all training + extra images
36
37
38 #-----
39 #paths <- c("/Users/joseadiazg/Desktop/Temporales
           UGR/SIGE/all_data_resized/Type_1",
40 #           "/Users/joseadiazg/Desktop/Temporales
           UGR/SIGE/all_data_resized/Type_2",
41 #           "/Users/joseadiazg/Desktop/Temporales
           UGR/SIGE/all_data_resized/Type_3",
42 #           "/Users/joseadiazg/Desktop/Temporales
           UGR/SIGE/train-extra-unidas/Type_1",
43 #           "/Users/joseadiazg/Desktop/Temporales
           UGR/SIGE/train-extra-unidas/Type_2",
44 #           "/Users/joseadiazg/Desktop/Temporales
           UGR/SIGE/train-extra-unidas/Type_3",
45 #           )
46 #-----
47
48 in_type_counter <- 1
49
50 for (t in paths){
51
52   patients <- dir(t)
53
54   #For this simple example We rescale photos to 128*128 (= 16384)
55
56
57   n_columnas <- 1 + 1 + 49152
58
59

```

```

60 ordered_images <- data.frame(matrix(nrow = length(patients), ncol =
    n_columnas))
61 colnames(ordered_images) <- c("paciente", "Type", paste("R", (1:16384), sep =
    ""), paste("G", (1:16384), sep = ""), paste("B", (1:16384), sep = ""))
62
63 contador <- 1
64
65 if(in_type_counter == 1){
66     ordered_images$Type <- 1 #(*) ojo , asignación manual segun el folder que
        esté procesando
67 }
68
69 if(in_type_counter == 2){
70     ordered_images$Type <- 2 #(*) ojo , asignación manual segun el folder que
        esté procesando
71 }
72
73 if(in_type_counter == 3){
74     ordered_images$Type <- 3 #(*) ojo , asignación manual segun el folder que
        esté procesando
75 }
76
77 #####
78 mom_inicio <- Sys.time()
79 print("beginning calculation: ")
80 print(mom_inicio)
81 #####
82
83 for (p in patients){
84
85     cat("contador:", contador, " paciente:", p, "\n")
86
87     ordered_images$paciente[contador] <- p
88
89     imagen_paciente <- readImage(paste(t, p, sep = "/")) #abre imagen de cada
        paciente...
90
91     #TODO: We have to change this resize to a new version in witch we use the same
92     # proportion of tam but smaller.
93
94     imagen_paciente <- resize(imagen_paciente, w = 128, h = 128)
95
96
97     ordered_images[contador, c(3:16386)] <- imagen_paciente[, 1]
98     ordered_images[contador, c(16387:32770)] <- imagen_paciente[, 2]

```

```
99     ordered_images[contador, c(32771:49154)] <- imagen_paciente[, , 3]
100
101     contador <- contador + 1
102
103 }
104
105
106 ###
107 print("calculation time: ")
108 print(Sys.time() - mom_inicio)
109 ###
110
111
112 if(in_type_counter == 1){
113     ordered_images_Type_1 <- ordered_images
114 }
115
116 if(in_type_counter == 2){
117     ordered_images_Type_2 <- ordered_images
118 }
119
120 if(in_type_counter == 3){
121     ordered_images_Type_3 <- ordered_images
122 }
123
124
125 in_type_counter <- in_type_counter + 1
126 }
127
128
129 ordered_images_all <- bind_rows(ordered_images_Type_1,
130                                ordered_images_Type_2, ordered_images_Type_3)
131
132 #-----
133 # Subset for validation
134 #-----
135
136 table(ordered_images_all$Type)
137
138
139 ordered_images_all$Type <- ordered_images_all$Type - 1
140
141
142 set.seed(9)
```

```

143 set_validacion <- ordered_images_all %>%
144   group_by(Type) %>%
145   sample_n(25) %>%
146   ungroup()
147
148
149 #-----
150 # Undersampling
151 #-----
152
153
154 set_train_unbalanced <- ordered_images_all[ordered_images_all$paciente %in%
155   setdiff(ordered_images_all$paciente, set_validacion$paciente), ]
156
157 #primera muestra train undersampled:
158 set.seed(9)
159 undersample_1_set_train <- set_train_unbalanced %>%
160   group_by(Type) %>%
161   #sample_n(225) %>%
162   ungroup() %>%
163   sample_frac(1) %>%
164   sample_frac(1) #WE SHUFFLE TWICE
165
166 # dim(undersample_1_set_train)
167 # #[1] 675 12290
168 # dim(set_validacion)
169 # #[1] 75 12288
170
171 #_____train and validación labels_____
172 target_undersample_1 <- undersample_1_set_train$Type
173 label_set_validacion <- set_validacion$Type
174 #_____
175
176
177 #_____train set1(undersampled) y and validation set, both balanced_____
178 undersample_1_set_train <-undersample_1_set_train[, c(3:49154)]
179 set_validacion <-set_validacion[, c(3:49154)]
180 #_____target y label de
181   validación_____
182
183 undersample_1_set_train <- t(undersample_1_set_train)
184 dim(undersample_1_set_train) <- c(128, 128, 3, 675)
185

```

```

186 set_validation <- t(set_validation)
187 dim(set_validation) <- c(128, 128, 3, 75)
188
189
190
191
192 #-----
193 # Tunning parametters for the nn
194 #-----
195
196
197 n_output <- 3
198 num_filters_conv2 = 14
199 num.round = 10
200 learning.rate = 0.1
201 momentum = 0.0056
202 weight_decay = 0.0046
203 initializer = 0.0667
204
205
206 # AND SOME HELPER FUNCTIONS:
207 mLogLoss.normalize = function(p, min_eta=1e-15, max_eta = 1.0){
208   #min_eta
209   for(ix in 1:dim(p)[2]) {
210     p[,ix] = ifelse(p[,ix]<=min_eta,min_eta,p[,ix]);
211     p[,ix] = ifelse(p[,ix]>=max_eta,max_eta,p[,ix]);
212   }
213   #normalize
214   for(ix in 1:dim(p)[1]) {
215     p[ix,] = p[ix,] / sum(p[ix,]);
216   }
217   return(p);
218 }
219
220 # helper function
221 #calculates logloss
222 mlogloss = function(y, p, min_eta=1e-15,max_eta = 1.0){
223   class_loss = c(dim(p)[2]);
224   loss = 0;
225   p = mLogLoss.normalize(p,min_eta, max_eta);
226   for(ix in 1:dim(y)[2]) {
227     p[,ix] = ifelse(p[,ix]>1,1,p[,ix]);
228     class_loss[ix] = sum(y[,ix]*log(p[,ix]));
229     loss = loss + class_loss[ix];
230   }

```



```

231  #return loss
232  return ( list ( "loss"=-1*loss/dim(p)[1], "class_loss"=class_loss));
233  }
234
235  # mxnet specific logloss metric
236  mx.metric.mlogloss <- mx.metric.custom("mlogloss", function(label, pred){
237    p = t(pred);
238    m = mlogloss(class.ind(label),p);
239    gc();
240    return(m$loss);
241  })
242
243
244
245  #-----
246  # Train the nn
247  #-----
248
249
250
251  train.x <- undersample_1_set_train
252  train.y <-target_undersample_1
253
254
255  # -----
256  data = mx.symbol.Variable('data')
257  #FIRST CONVOLUTIONAL LAYER + POOLING
258  conv1 = mx.symbol.Convolution(data=data, kernel=c(3, 3), num_filter = 3)
259  relu1 = mx.symbol.Activation(data=conv1, act_type="relu")
260  pool1 = mx.symbol.Pooling(data=relu1, pool_type="max", kernel=c(2,2),
    stride=c(2,2))
261
262
263  #SECOND CONVOLUTIONAL LAYER + POOLING
264  conv2 = mx.symbol.Convolution(data=pool1, kernel=c(3,3), num_filter =
    num_filters_conv2)
265  relu2 = mx.symbol.Activation(data=conv2, act_type="relu")
266  pool2 = mx.symbol.Pooling(data=relu2, pool_type="max", kernel=c(2,2),
    stride=c(2,2))
267
268  #FLATTEN THE OUTPUT
269  flatten = mx.symbol.Flatten(data=pool2)
270

```

```

271 #FEED FULLY CONNECTED LAYER, NUMBER OF HIDDEN NODES JUST
    GEOMETRIC MEAN OF INPUT(14.5 * 14.5 * 13 = 2733.25) AND OUTPUT
    (3), sqrt(2733.25*3) = 91
272 input_previo_a_filtroconv2 <- 14.5*14.5
273 n_input <- input_previo_a_filtroconv2 * num_filters_conv2
274 num_hidden_fc1 <- round(sqrt(n_input*n_output))
275
276 fc1 = mx.symbol.FullyConnected(data=flatten, num_hidden=84)
277 relu4 = mx.symbol.Activation(data=fc1, act_type="relu")
278
279 # _ _ _ _ _
280
281
282 fc2 = mx.symbol.FullyConnected(data=relu4, num_hidden=3) #ESTA PARA
    CLASIFICACION
283
284 mi_softmax = mx.symbol.SoftmaxOutput(data=fc2)
285
286
287 devices <- mx.cpu()
288
289 mx.set.seed(0)
290
291 # _ _ _
292 tic <- proc.time()
293 # _ _ _
294 model <- mx.model.FeedForward.create( mi_softmax #for clasification
295                                     , X=train.x
296                                     , y=train.y
297                                     , eval.data = list("data" =
298                                     set_validation,"label" =
299                                     label_set_validation)
300                                     , ctx=devices
301                                     , num.round=num.round
302                                     , array.batch.size = 75
303                                     , learning.rate = learning.rate
304                                     , momentum = momentum
305                                     , wd=weight_decay
306                                     , eval.metric = mx.metric.mlogloss
307                                     , initializer =mx.init.uniform( initializer )
308                                     #, epoch.end.callback =
309                                     mx.callback.save_checkpoint("modelo_guardado_ccs")
310                                     #(TO SAVE MODEL AT EVERY
311                                     ITERATION)

```

---

```

307                                     , batch.end.callback =
308                                     mx.callback.log.train.metric(10)#, log)
309                                     , array.layout="columnmajor"
310 )
311 #___
312 print(proc.time() - tic)
313 #___
314
315
316 #-----
317 # Validation
318 #-----
319
320
321 preds <- predict(model, set_validation,
322                  ctx = NULL,
323                  array.layout = "auto")
324
325 #WE CAN INSPECT OUR LIMITED VALIDATION SET, PROBABILITIES:
326 predicciones <- t(preds)
327 predicciones <- cbind(label_set_validation, predicciones)
328 head(predicciones)

```

---

### 3.2.2. TensorFlow+Keras

---

```

1      # This Python 3 environment comes with many helpful analytics libraries
      installed
2      # It is defined by the kaggle/python docker image:
      https://github.com/kaggle/docker-python
3      # For example, here's several helpful packages to load in
4
5      import numpy as np # linear algebra
6      import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
7      import glob
8      import os
9      import matplotlib.pyplot as plt
10     import seaborn as sns
11     import cv2
12
13     from multiprocessing import Pool, cpu_count
14     from subprocess import check_output
15     from subprocess import check_output

```

```

16 from PIL import ImageFilter, ImageStat, Image, ImageDraw
17 from sklearn.model_selection import GridSearchCV, StratifiedKFold
18 from sklearn.preprocessing import LabelEncoder
19 from sklearn.metrics import log_loss
20 from keras.wrappers.scikit_learn import KerasClassifier
21 from keras.models import Sequential
22 from keras.layers.core import Dense, Dropout, Flatten
23 from keras.layers.convolutional import Convolution2D, ZeroPadding2D,
    MaxPooling2D
24 from keras import optimizers
25 from keras.wrappers.scikit_learn import KerasClassifier
26 from keras.models import Sequential
27 from keras.layers.core import Dense, Dropout, Flatten, Activation
28 from keras.layers.convolutional import Convolution2D, ZeroPadding2D,
    MaxPooling2D
29 from keras import optimizers
30 from keras.preprocessing.image import ImageDataGenerator
31 from sklearn.model_selection import train_test_split
32 from keras.optimizers import SGD
33 from keras import backend as K
34 K.set_image_dim_ordering('th')
35 K.set_floatx('float32')
36
37
38 def roi(pathtrain):
39     for typ in types:
40         for img in os.listdir(pathtrain + '/' + typ):
41             image = pathtrain + '/' + typ + '/' + img
42             os.chdir(pathtrain + '/' + typ + '/')
43             ii = cv2.imread(image)
44             #cv.imshow('image',ii[:, :, 1])
45             #cv.waitKey(0)
46             b,g,r = cv2.split(ii)
47             rgb_img = cv2.merge([r,g,b])
48             rgb_img1 = pc.rgb_to_hsv(rgb_img)
49             indices = np.where(rgb_img1[:, :, 0] < 0.7)
50             rgb_img1[:, :, 0][indices] = 0
51             rgb_img1[:, :, 1][indices] = 0
52             rgb_img1[:, :, 2][indices] = 0
53             rgb_img1 = pc.hsv_to_rgb(rgb_img1).astype(np.uint8)
54             pp.imsave(fname = img.split('.')[0] + '_trans.jpg', arr = rgb_img1)
55     return fname
56
57 def im_multi(path):
58     try:

```

```

59         im_stats_im_ = Image.open(path)
60         return [path, {'size': im_stats_im_.size}]
61     except:
62         print(path)
63         return [path, {'size': [0,0]}]
64
65 def im_stats(im_stats_df):
66     im_stats_d = {}
67     p = Pool(cpu_count())
68     ret = p.map(im_multi, im_stats_df['path'])
69     for i in range(len(ret)):
70         im_stats_d[ret[i][0]] = ret[i][1]
71     im_stats_df['size'] = im_stats_df['path'].map(lambda x: ' '.join(str(s) for s
72         in im_stats_d[x]['size']))
73     return im_stats_df
74
75 def get_im_cv2(path):
76     img = cv2.imread(path)
77     resized = cv2.resize(img, (64, 64), cv2.INTER_LINEAR)
78     return [path, resized]
79
80 def normalize_image_features(paths):
81     imf_d = {}
82     p = Pool(cpu_count())
83     ret = p.map(get_im_cv2, paths)
84     for i in range(len(ret)):
85         imf_d[ret[i][0]] = ret[i][1]
86     ret = []
87     fdata = [imf_d[f] for f in paths]
88     fdata = np.array(fdata, dtype=np.uint8)
89     fdata = fdata.transpose((0, 3, 1, 2))
90     fdata = fdata.astype('float32')
91     fdata = fdata / 255
92     return fdata
93
94 def create_model(opt_='adamax'):
95     model = Sequential()
96     model.add(Convolution2D(4, 3, 3, activation='relu', dim_ordering='th',
97         input_shape=(3, 64, 64))) #use input_shape=(3, 64, 64)
98     model.add(MaxPooling2D(pool_size=(3, 3), strides=(3, 3),
99         dim_ordering='th'))
100    model.add(Convolution2D(8, 3, 3, activation='relu', dim_ordering='th'))
101    model.add(MaxPooling2D(pool_size=(3, 3), strides=(3, 3),
102        dim_ordering='th'))

```

```

100     model.add(Dropout(0.2))
101
102     model.add(Flatten())
103     model.add(Dense(12, activation='tanh'))
104     model.add(Dropout(0.1))
105     model.add(Dense(3, activation='softmax'))
106
107     model.compile(optimizer=opt_, loss='sparse_categorical_crossentropy',
108                   metrics=['accuracy'])
109     return model
110
111
112 def main():
113     os.chdir('D:\Facultad\Master\Segundo cuatrimestre\SIGE\PracticaFinal')
114     test = glob.glob("pruebaTest/*.jpg")
115     test = pd.DataFrame([p[11:len(p)],p] for p in test], columns =
116         ['image','path'])
117     train= glob.glob("prueba1/**/*.png")+glob.glob("prueba2/**/*.jpg")
118     train = pd.DataFrame([p[0][8:14],p[0][15: len(p)],p] for p in train], columns
119         = ['type','image','path'])
120
121     train = im_stats(train)
122     train = train[train['size'] != '0 0'].reset_index(drop=True) #remove bad
123     images
124     train_data = normalize_image_features(train['path'])
125
126     np.save('train.npy', train_data, allow_pickle=True, fix_imports=True)
127
128     print(len(train))
129
130     le = LabelEncoder()
131     train_target = le.fit_transform(train['type'].values)
132     print(le.classes_) #in case not 1 to 3 order
133     np.save('train_target.npy', train_target, allow_pickle=True,
134           fix_imports=True)
135     test_data = normalize_image_features(test['path'])
136
137     np.save('test.npy', test_data, allow_pickle=True, fix_imports=True)
138
139     test_id = test.image.values
140     np.save('test_id.npy', test_id, allow_pickle=True, fix_imports=True)
141
142     train_data = np.load('train.npy')
143     train_target = np.load('train_target.npy')

```

---

```

140
141     x_train,x_val_train,y_train,y_val_train =
        train_test_split(train_data,train_target,test_size=0.4,
            random_state=17)
142
143
144     datagen = ImageDataGenerator(rotation_range=0.3, zoom_range=0.3)
145     datagen.fit(train_data)
146
147
148     model = create_model()
149     print(x_train.shape)
150     print(y_train.shape)
151     model.fit_generator(datagen.flow(x_train,y_train, batch_size=15,
        shuffle=True), nb_epoch=35, samples_per_epoch=len(x_train),
        verbose=20, validation_data=(x_val_train, y_val_train))
152
153     test_data = np.load('test.npy')
154     test_id = np.load('test_id.npy')
155
156     pred = model.predict_proba(test_data)
157     df = pd.DataFrame(pred, columns=['Type_1','Type_2','Type_3'])
158     df['image_name'] = test_id
159     df.to_csv('submission0009.csv', index=False)
160
161     if __name__ == '__main__':
162         #freeze_support() # Optional under circumstances described in docs
163         main()

```

---

### 3.3. Fine Tunning

## Capítulo 4

# Multclasificación y mapas de características

### 4.1. Multclasificacion

### 4.2. Mapas de características



# Capítulo 5

## Conclusiones y vías futuras

En este capítulo final se estudian los resultados obtenidos a lo largo del trabajo y vías futuras para aumentar aún más el accuracy. También se complementan las conclusiones que se han ido obteniendo a lo largo del trabajo. Por tanto, podríamos resumir las conclusiones finales del trabajo en las siguientes:

### 5.1. Conclusiones finales

### 5.2. Vías futuras

## Capítulo 6

### ANEXO I

En este Anexo, podemos encontrar la tabla con los resultados obtenidos en las distintas entregas a Kaggle, el número de resultados no coincide al 100 % con los subidos a Kaggle, dado que a petición del profesor de teoría, Francisco Herrera, se realizó una batería de experimentos sobre el problema con el algoritmo XGBoost. Los resultados de estos experimentos fueron obviados del problema ya que no ofrecían mejora alguna sobre los resultados.

<i>Sol</i>	<i>Preprocesado</i>	<i>Algoritmos</i>	<i>Acc Test</i>	<i>Acc Training</i>	<i>Pos</i>
1	Imputados valores perdidos	Asumimos todos mueren	0.61	0.616	6429
2	Imputados valores perdidos	Asumimos que solo viven las mujeres	0.7655	0.7867	4781
3	Imputados valores perdidos	Todos los hombres y las mujeres de tercera clase con >20 de Fare mueren, las demás viven.	0.77990	0.8080	3259
4	Imputados valores perdidos Creada variable isChild	Ademas de lo anterior, niños de 2 clase viven.	0.78240	0.8181	2877
5	Imputados valores perdidos	Random Forest nativo de R	0.78469	0.8316	2543
6	Imputados valores perdidos añadidas variable Title, isMother, isChild	Random Forest nativo de R	0.78469	0.8316	2543
7	Imputados valores perdidos, añadida variable Title	Random Forest rparty	0.79904	0.838	1732
8	Imputados valores perdidos, añadida variable Title, eliminados outliers	Random Forest rparty	0.80383	0.8432	936
9	Imputados valores perdidos, añadidas title, familysize y familyID,	Random Forest rparty	0.8134	0.8356	473
10	Añadidas nuevas características Eliminado outliers, imputados valores perdidos	Random Forest muy ajustado y probado con distintos parámetros quedándonos con el mejor en training	0.82297	0.88431	218

Tabla 6.1: Tabla de resultados



# Bibliografía

- [1] Repositorio del proyecto. <https://github.com/joseangeldiazg/>
- [2] Website de Kaggle <https://www.kaggle.com>
- [3] Competicion en Kaggle. <https://www.kaggle.com/c/intel-mobileodt-cervical-cancer-screening>
- [4] Tutorial MXNET. <https://www.kaggle.com/miguelpm/r-mxnet-simple-tutorial>
- [5] Script Tensorflow. <https://www.kaggle.com/marek3000/test-num-001/code>