Sistemas Inteligentes para la Gestión de la Empresa

UNIVERSIDAD DE GRANADA

Práctica 2:
Deep Learning para clasificación

Autores
Juan Manuel Castillo Nievas
Guillermo Bueno Vargas

Máster Profesional en Ingeniería Informática 2020-2021

—

Granada, 5 de junio de 2021

Índice

1. Introducción

La idea fundamental de esta práctica consiste en desarrollar y proponer una solución a un problema de clasificación con un determinado conjunto de datos. El conjunto de datos se llama **Fakeddit** [1] y está formado por un conjunto de noticias falsas.

El problema de clasificación consiste en predecir si una determinada publicación es una noticia falsa o no. En un principio se trata de una clasificación binaria (es o no es una notifica falsa), pero este problema se puede ampliar a una clasificación con seis clases, distinguiendo distintos tipos de noticias falsas.

2. Entorno de trabajo

El desarrollo y proceso de esta práctica se ha realizado usando **RStudio**, un entorno de desarrollo integrado para el lenguaje de programación **R**, tal y como se ha ido aprendiendo a lo largo de esta asignatura.

Se ha utilizado **Keras** [2] como API para el desarrollo y experimentación de distintas redes neuronales para conseguir el objetivo de la práctica. Realmente **Keras** es una API de alto nivel de **Tensorflow** [3]. De acuerdo a la definición en su propia web, **Tensorflow** es "*la principal biblioteca de código abierto para enseñarte a desarrollar y entrenar modelos de aprendizaje automático.*" Sin embargo, esta biblioteca está diseñada para construir modelos de aprendizaje automático y resulta más compleja y menos *user-friendly*, mientras que **Keras** está diseñada para redes neuronales específicamente [4]. De acuerdo a las ventajas de **Keras** proporcionadas en la misma web de **Tensorflow** [5]: "*Keras tiene una interfaz simple y consistente optimizada para casos de uso comun. Proporciona informacion clara y procesable sobre los errores del usuario.*"

3. Organización de los datos

Se cuenta con tres muestras de datos: *mini*, *medium* y *all*, que contienen 50, 10.000 y todas las imágenes de **Fakeddit**, respectivamente. Para cada muestra de datos, se ha construido una versión binaria del problema, *twoClasses*, y una versión con seis clases, llamada *sixClasses*.

Para el desarrollo de esta práctica se ha usado el conjunto de 10.000 imágenes (*medium*). Inicialmente se ha usado la versión de clasificación binaria.

La organización del directorio de trabajo se puede ver en la Figura 1. Dentro de cada muestra de datos hay tres carpetas: *test*, *train* y *val*; y dentro de cada una de estas carpetas hay otras n carpetas (siendo n el número de clases) y un fichero *.tsv* que contiene los metadatos de todas las imágenes de la carpeta.

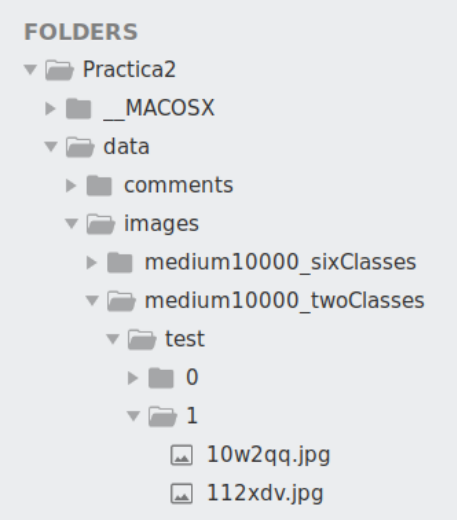

Figura 1: Organización del directorio de trabajo

4. Análisis exploratorio de datos

4.1. Lectura y visualización de los datos

Para visualizar una simple imagen (ver Figura 2) se puede utilizar el código que se encuentra a continuación:

```
img_sample <- image_load(path = './data/images/medium10000_twoClasses/test/1/2euhha.jpg',
                         target_size = c(150, 150))
img_sample_array <- array_reshape(image_to_array(img_sample), c(1, 150, 150, 3))
plot(as.raster(img_sample_array[1,,,] / 255))
```

Figura 2: Visualización de '*./data/images/medium10000_twoClasses/test/1/2euhha.jpg*'

4.2. Carga de directorios y metadatos

Se procede a cargar el directorio *medium10000_twoClasses* para poder trabajar con todas las imágenes que se encuentran en él. También se carga el archivo que contiene todos los comentarios de las noticias y se procede a **crear la columna *class*** a partir del valor de la columna ***2_way_label***, indicando si la fila es una noticia falsa o no (ver Figura 3).

```
dataset_dir           <- './data/images/medium10000_twoClasses'
train_images_dir      <- paste0(dataset_dir, '/train')
val_images_dir        <- paste0(dataset_dir, '/val')
test_images_dir       <- paste0(dataset_dir, '/test')
comments_file         <- './data/comments/all_comments.tsv'

metadata_train <- read_tsv(paste0(train_images_dir, "/multimodal_train.tsv"))
metadata_train <- metadata_train %>%
  mutate(created_at = as.POSIXct(created_utc, origin="1970-01-01")) %>%
  select(-one_of('created_utc')) %>%
  mutate(class = ifelse(`2_way_label` == 0, 'Disinformation', 'Other'))
```

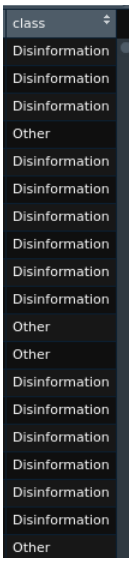

Figura 3: Creación de la columna **class**

4.3. Exploración de comentarios

Se han explorado los comentarios para ver las estadísticas más básicas y se ha observado un alto número de **valores NA**, con un total de **1.094.552**. La **media** es de **0.0137** (ver Figura 4).

```
comments <- read_tsv(comments_file)

summary(comments)
sum(is.na(comments))
mean(is.na(comments))
```

```
> summary(comments)
       X1                id              author             body          isTopLevel      parent_id         submission_id          ups          
 Min.   :       0   Length:10002177    Length:10002177    Length:10002177    Mode :logical   Length:10002177    Length:10002177    Min.   :-9432.00  
 1st Qu.: 2625314   Class :character   Class :character   Class :character   FALSE:6134918   Class :character   Class :character   1st Qu.:    1.00  
 Median : 5256822   Mode  :character   Mode  :character   Mode  :character   TRUE :3866201   Mode  :character   Mode  :character   Median :    2.00  
 Mean   : 5290277                                                            NA's :1058                                            Mean   :   23.81  
 3rd Qu.: 7973879                                                                                                              3rd Qu.:    7.00  
 Max.   :10601592                                                                                                              Max.   :42573.00  
 NA's   :639                                                                                                                   NA's   :1058      
> sum(is.na(comments))
[1] 1094552
> mean(is.na(comments))
[1] 0.01367892
```

Figura 4: Resumen estadístico de comentarios

Se ha decidido **eliminar los valores perdidos**, siguiendo el mismo procedimiento que se propuso en clase. También se ha **combinado la tabla de metadatos con la tabla de comentarios** usando la operación **left_join**.

```
comments <- comments %>%
  drop_na()
sum(is.na(comments))

metadata_train_comments <- left_join(x = metadata_train, y = comments,
                                     by = c("id" = "submission_id"),
                                     keep = FALSE,
                                     suffix = c('.publication', '.comment'))
```

4.4. Distribución de clases

En la Figura 5 se muestra la distribución de clases, siendo una clasificación binaria (desinformación o no). En el conjunto de datos mediano, se encuentran un total de **6.129 valores pertenecientes a desinformación** y **3.871 valores pertenecientes a otros**.

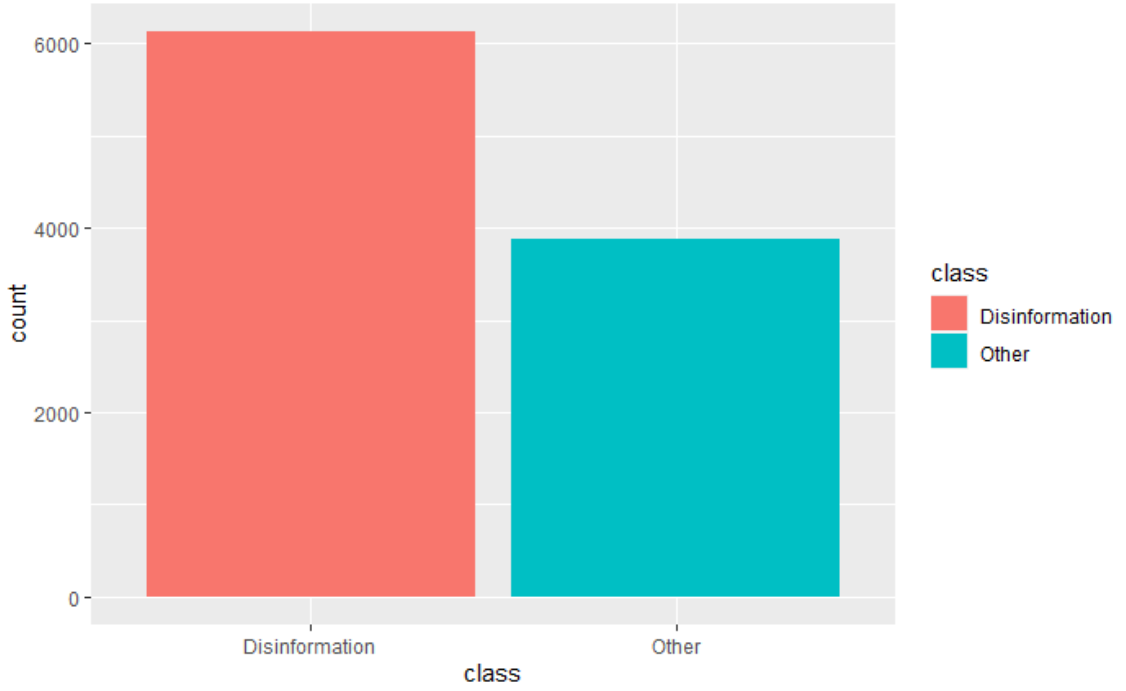

Figura 5: Distribución de clases (2 clases)

Se encuentra un conjunto de datos **desbalanceado**, pues hay muchos más ejemplos pertenecientes a la primera clase que a la otra. Para solucionar esto se aplica la técnica vista en clase de **downsampling**, la cual consiste en reducir el número de ejemplos de la clase mayoritaria al número de ejemplos de la clase minoritaria (elegidos al azar para evitar cualquier tipo de sesgo).

El resultado se puede ver en la Figura 6, en la cual se puede ver que ahora hay **3.871 ejemplos de ambas clases**. Esto quiere decir que se ha reducido el número del conjunto de datos de 10.000 ejemplos a **7.742 ejemplos**.

Figura 6: Distribución de clases con **downsampling**

4.5. Análisis de información de desinformación

A lo largo de esta sección se van a mostrar las gráficas de datos más interesantes con respecto al conjunto de datos. Aparte de ser información del conjunto de datos, también llegan a ser **curiosidades** que permiten analizar la situación actual y el rumbo que están tomando las noticias falsas en nuestra sociedad.

4.5.1. Crecimiento de desinformación

En la Figura 7 se muestra una gráfica que indica la tendencia en los últimos años de desinformación. Se puede apreciar que todo esto apareció aproximadamente a principios de 2010, posiblemente con la influencia del auge de nuevas redes sociales tales como **Twitter**, **Instagram** y **Whatsapp**.

También hay que indicar que en el año 2017 se inició lo que se conoce como **Deep Fakes**, que son técnicas de manipulación de imágenes y vídeo que se utilizan mucho para la propagación de nuevas noticias falsas.

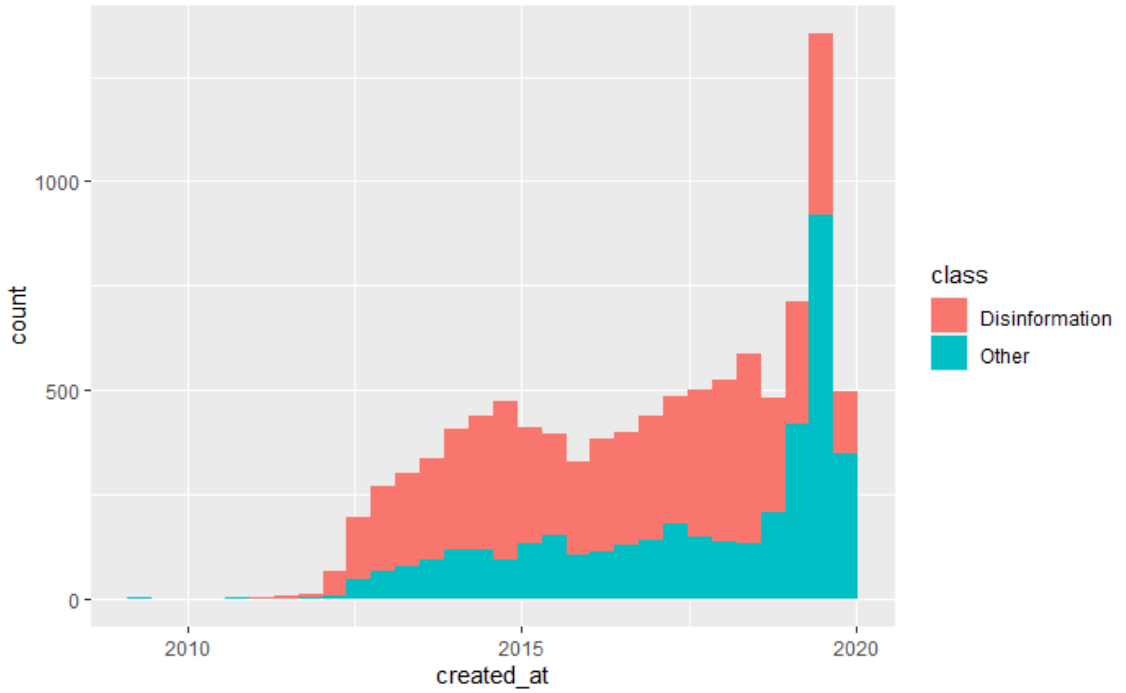

Figura 7: Crecimiento de desinformación a lo largo del tiempo

4.5.2. Número de comentarios por post con más desinformación

En la Figura 8 se muestra el número de comentarios por post con más desinformación. Se puede apreciar que hay una mayoría de publicaciones con **0 comentarios**, pero hay muchísimas otras que presentan un gran número de comentarios. Se puede ver que el número de comentarios oscila entre aproximadamente **1 y 15 comentarios**.

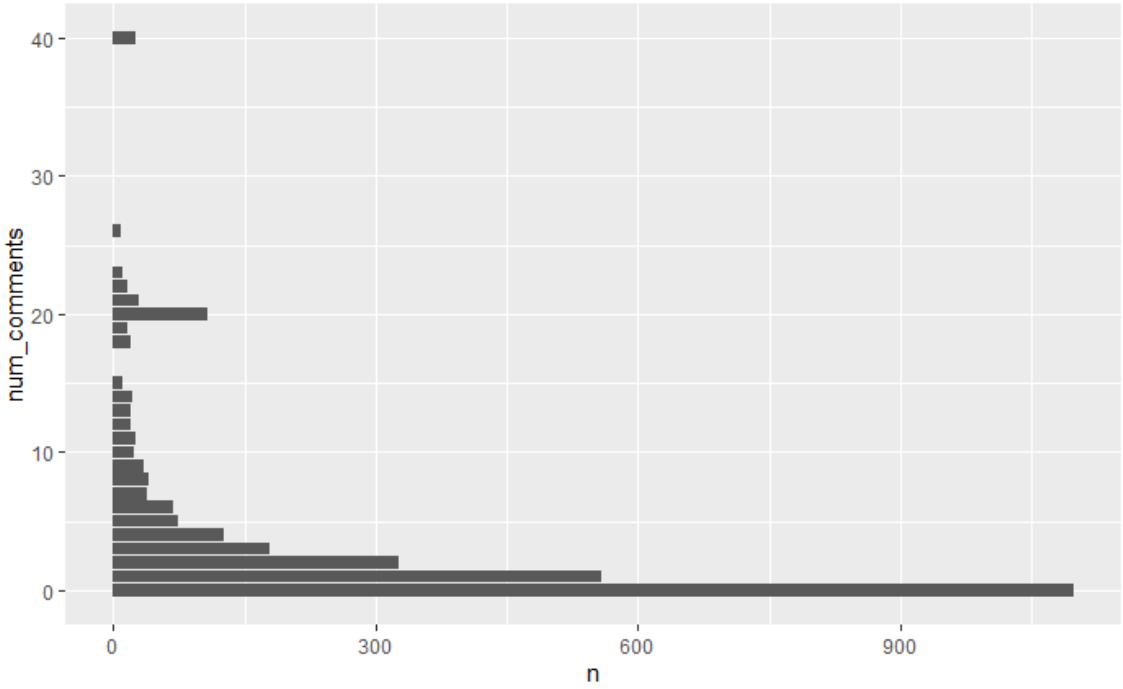

Figura 8: Número de comentarios por post con más desinformación

4.5.3. Subrredits con más desinformación

En la Figura 9 se muestran los subrredits con más desinformación. Con una gran diferencia, la categoría con más desinformaión es la de **psbattle_artwork**.

Se hace mención especial a la categoría **fakehistoryporn** debido a que el término anteriormente mencionado de **Deep Fakes** (2017) se inició debido a que un usuario de la red social **Redit** utilizó técnicas de manipulación de vídeo e imágenes para suplantar la identidad de *celebrities* en vídeos pornográficos, y desde entonces esta categoría está tendiendo a subir.

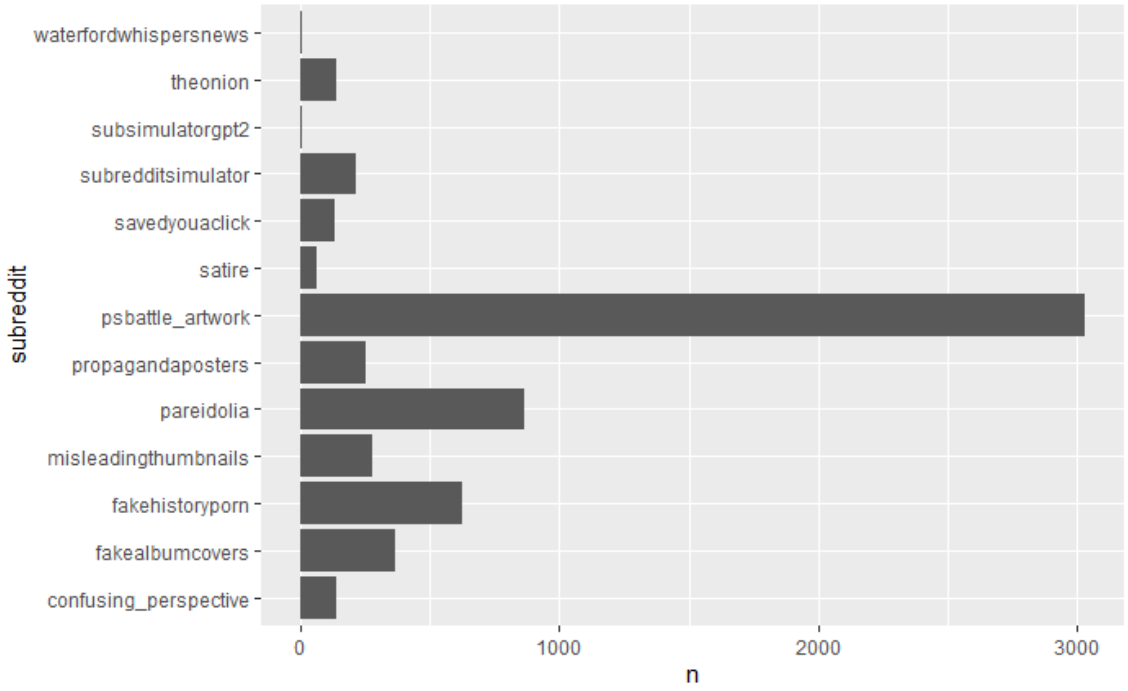

Figura 9: Subrredits con más desinformación

4.6. Análisis de información en los títulos

En esta sección se va a extraer la información de los títulos de las publicaciones. Se ha obtenido la cantidad de exclamaciones, mayúsculas, dígitos y emojis existentes en los títulos de las distintas publicaciones.

```
data_binary_extended <- data_binary %>%
  mutate(title_text_exclamations = str_count(title, "!")) %>%
  mutate(title_text_caps = str_count(title, "[A-Z]")) %>%
  mutate(title_text_digits = str_count(title, "[0-9]")) %>%
  mutate(title_text_emojis = str_count(title, '[\U{1F300}-\U{1F6FF}]')) %>%
  mutate(title_text_emoji_flag = str_count(title, '\U{1F1FA}|\U{1F1F8}]'))
```

4.6.1. Exclamaciones

En la Figura 10 se muestra la distribución de densidad del **número de exclamaciones** usadas en los títulos de las publicaciones. Es curioso apreciar que las noticias con desinformación tienden a usar un **número mayor de exclamaciones**.

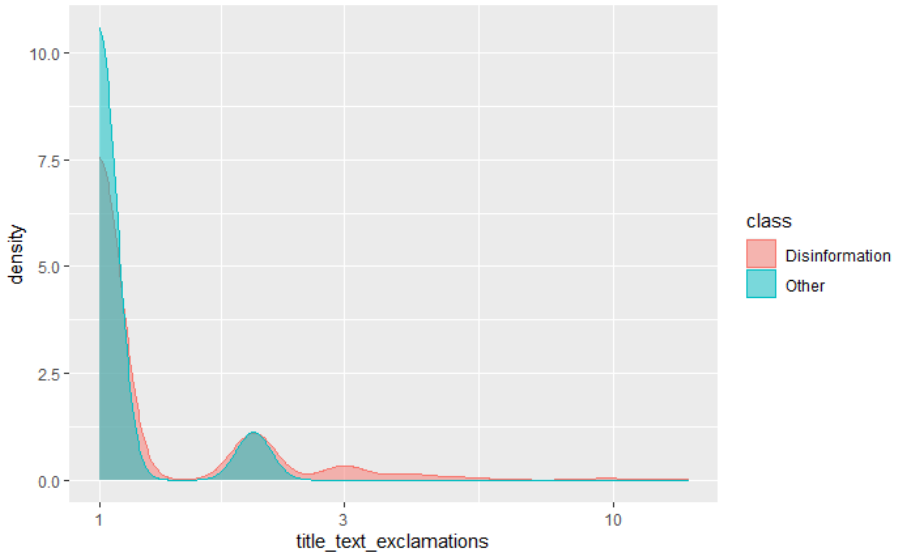

Figura 10: Distribución de densidad de exclamaciones en títulos

4.6.2. Mayúsculas

En la Figura 11 se muestra la distribución de densidad del **número de mayúsculas** usadas en los títulos de las publicaciones. Esta distribución parece estar un poco más compensada entre ambas clases.

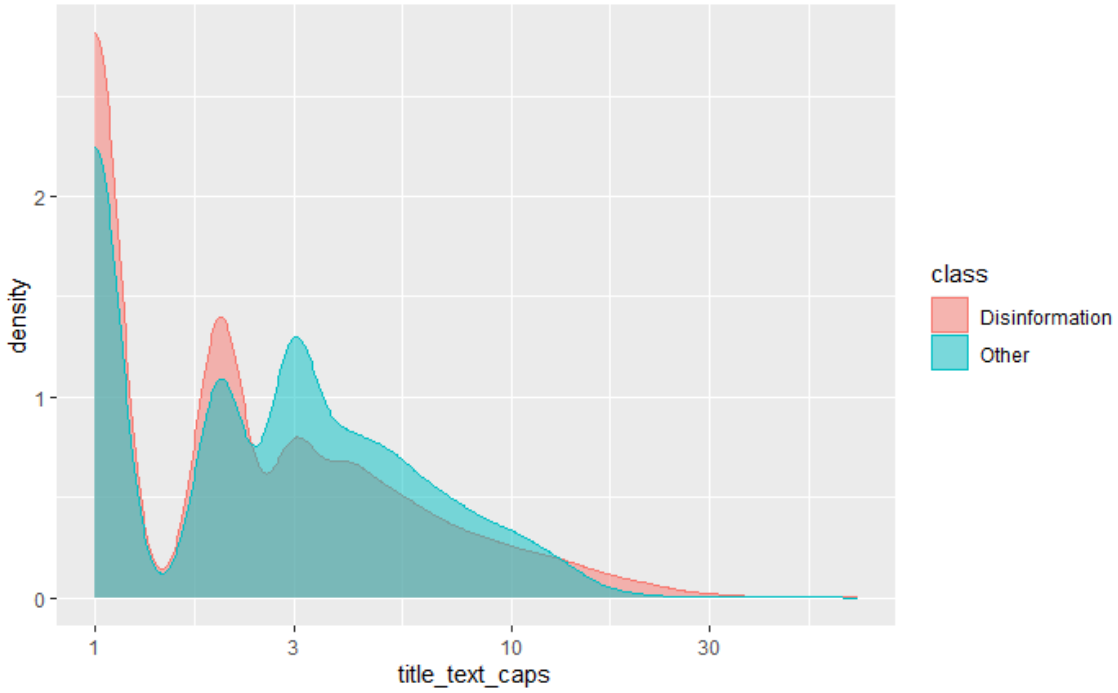

Figura 11: Distribución de densidad de mayúsculas en títulos

4.6.3. Dígitos

En la Figura 12 se muestra la distribución de densidad del **número de dígitos** usadas en los títulos de las publicaciones. Las noticias con desinformación tienden a usar un **número mayor de dígitos** en su títulos.

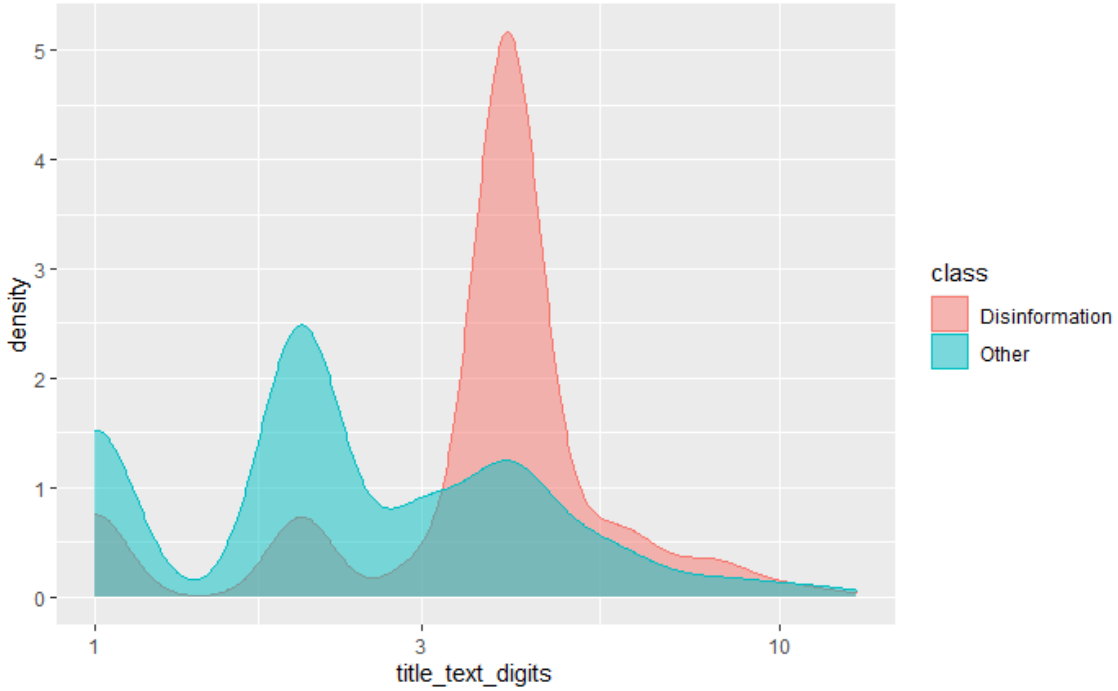

Figura 12: Distribución de densidad de dígitos en títulos

4.6.4. Emojis

En la Figura 13 se muestra la distribución de densidad del **número de emojis** usadas en los títulos de las publicaciones. El número de emojis no aporta demasiado ya que no suelen usarse emojis en los títulos de las noticias.

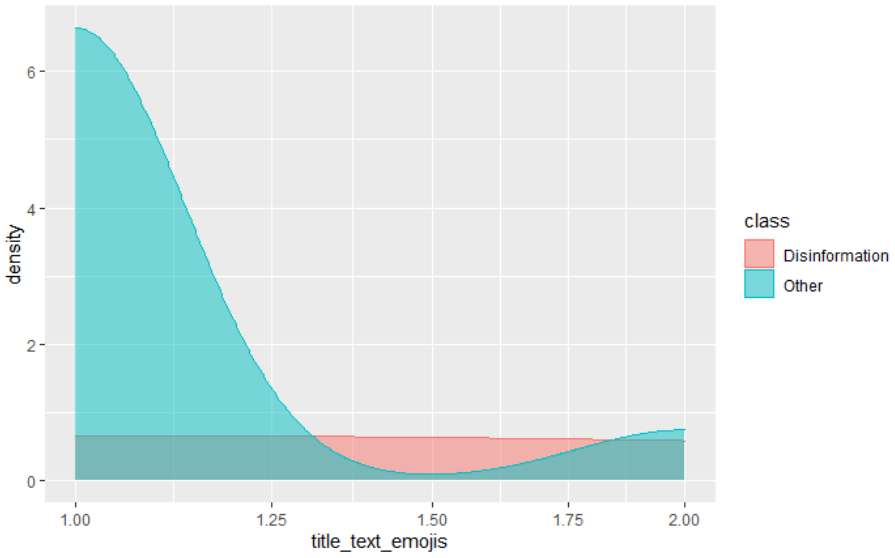

Figura 13: Distribución de densidad de emojis en títulos

4.7. Análisis de información en el cuerpo de las noticias

En esta sección se ha extraído la información que hay en el cuerpo de las noticias. Se ha obtenido la cantidad de exclamaciones, mayúsculas, signos de puntuación, dígitos y emojis.

```
data_binary_comments_extended <- data_binary_comments %>%
  mutate(body_text_exclamations = str_count(body, "!")) %>%
  mutate(body_text_caps = str_count(body, "[A-Z]")) %>%
  mutate(body_text_has_punctuation = str_count(body, "[.,:;]")) %>%
  mutate(body_text_digits = str_count(body, "[0-9]")) %>%
  mutate(body_text_emojis = str_count(body, '[\U{1F300}-\U{1F6FF}]')) %>%
  mutate(body_text_emoji_flag = str_count(body, '\U{1F1FA}|\U{1F1F8}]'))
```

4.7.1. Exclamaciones

En la Figura 14 se muestra la distribución de densidad del **número de exclamaciones** usadas en el cuerpo de las publicaciones. Esta distribución no sirve mucho para diferenciar entre una noticia falsa o no, ya que las distribuciones son bastante parecidas.

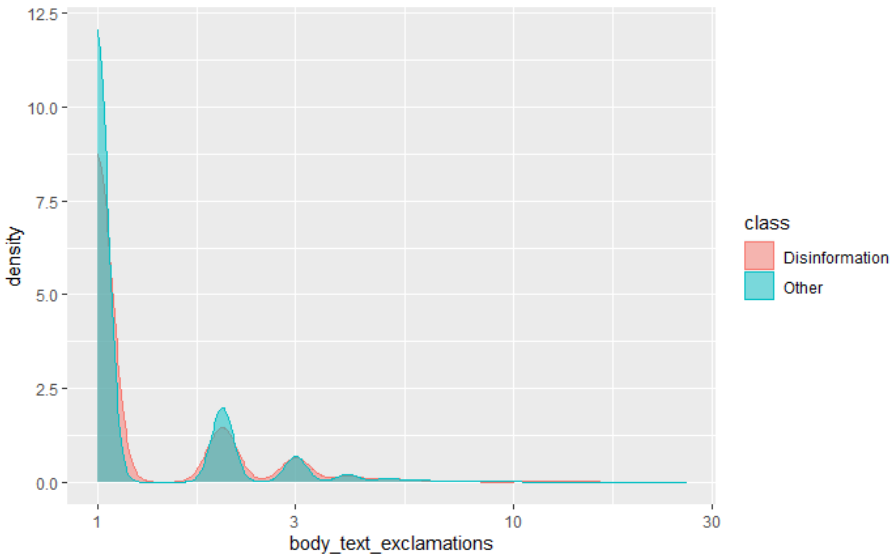

Figura 14: Distribución de densidad de exclamaciones en el cuerpo de las noticias

4.7.2. Mayúsculas

En la Figura 15 se muestra la distribución de densidad del **número de mayúsculas** usadas en el cuerpo de las publicaciones. Al igual que la distribución de exclamaciones, esta distribución no sirve mucho para diferenciar entre una noticia falsa o no, ya que las distribuciones son bastante parecidas.

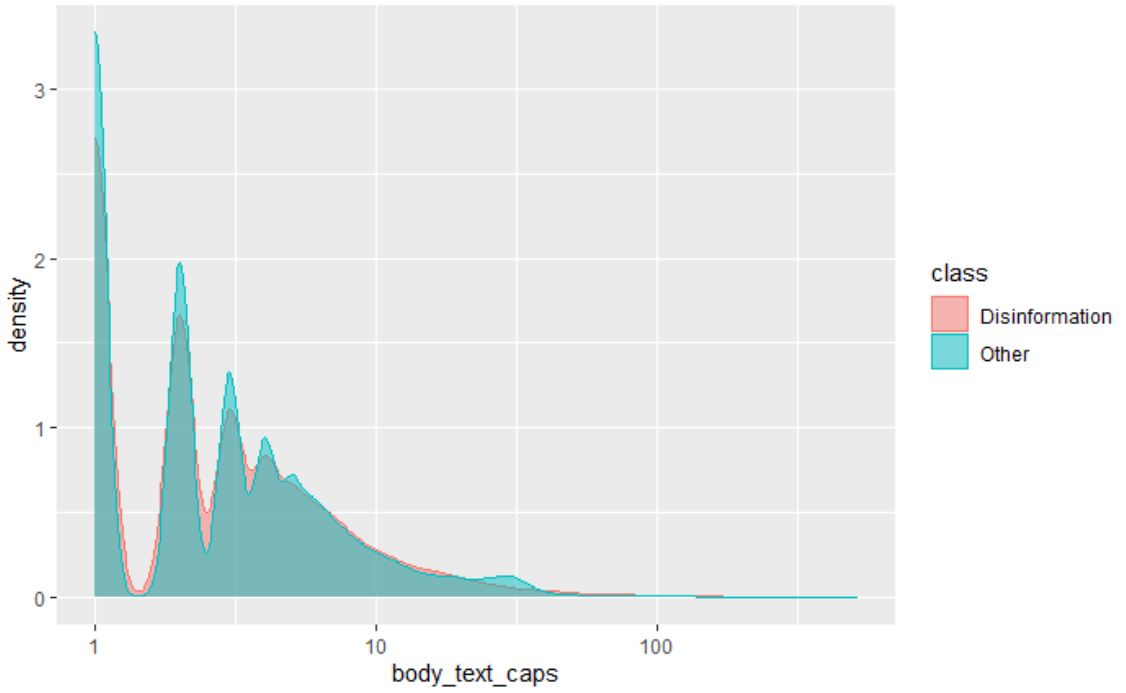

Figura 15: Distribución de densidad de mayúsculas en el cuerpo de las noticias

4.7.3. Signos de puntuación

En la Figura 16 se muestra la distribución de densidad del **número de signos de puntuación** usadas en el cuerpo de las publicaciones. También son distribuciones bastante parecidas.

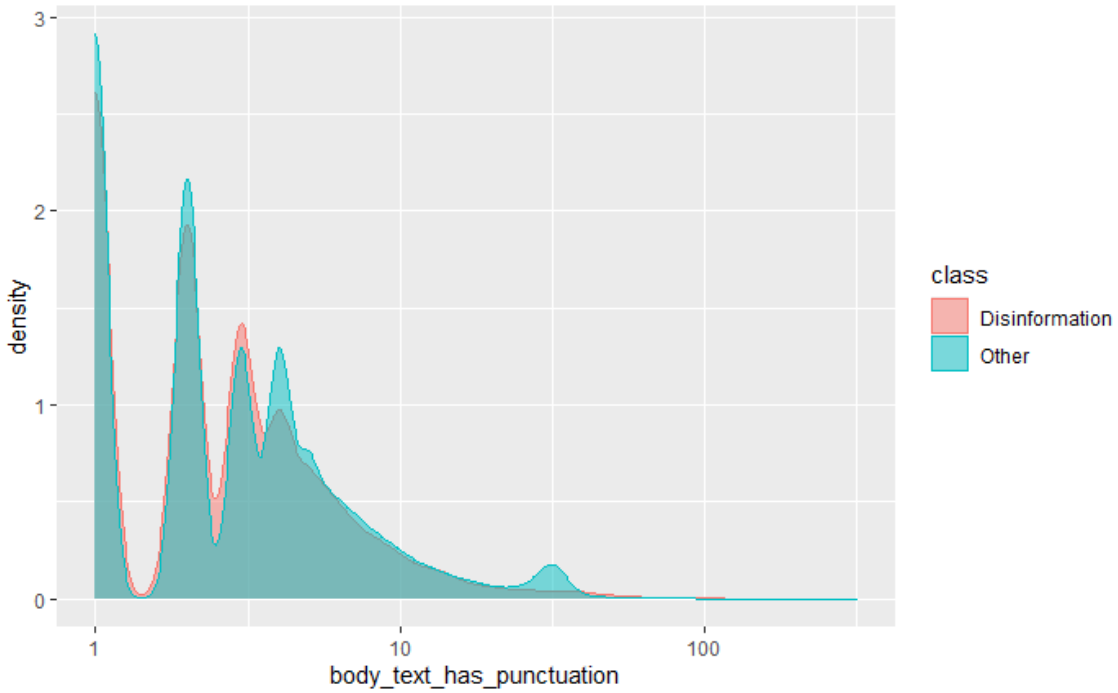

Figura 16: Distribución de densidad de signos de puntuación en el cuerpo de las noticias

4.7.4. Números

En la Figura 17 se muestra la distribución de densidad del **número de números** usadas en el cuerpo de las publicaciones. Permite dislumbrar cierta aumento en las noticias que desinforman, pero no parece bastante relevante para determinar si una notica es falsa o no.

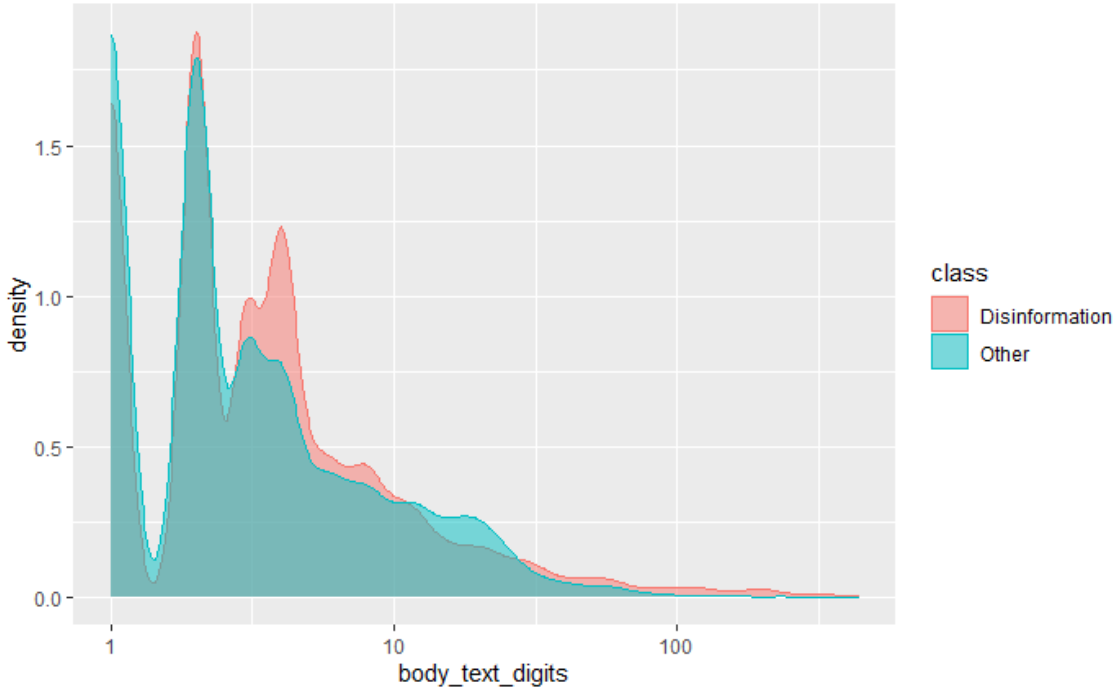

Figura 17: Distribución de densidad de números en el cuerpo de las noticias

4.7.5. Emojis

En la Figura 18 se muestra la distribución de densidad del **número de emojis** usadas en el cuerpo de las publicaciones. Se puede apreciar que las **noticias falsas** poseen un número de emojis en los cuerpos relativamente mayor a una noticia que no es falsa.

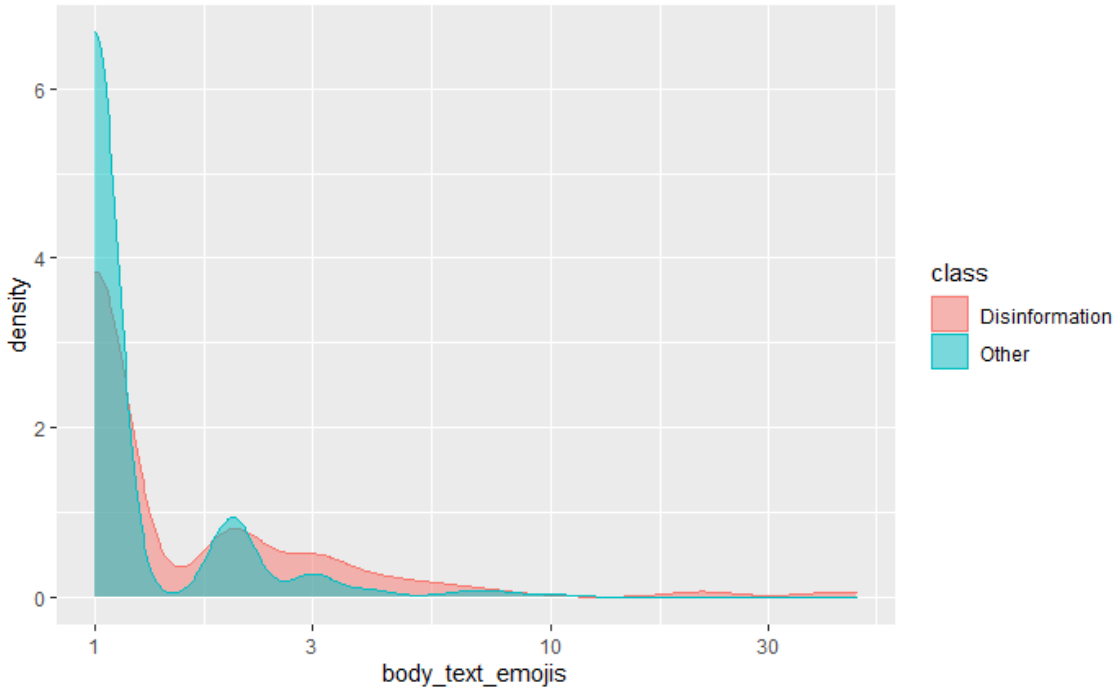

Figura 18: Distribución de densidad de emojis en el cuerpo de las noticias

5. Modelos de clasificación

En esta sección se presentan los distintos modelos clasificación empleados para la resolución del problema de clasificación binaria con imágenes.

Tal y como se ha propuesto en clase, se han utilizado generadores de flujo para evitar los problemas que puedan surgir debido a recursos limitados de memoria. Los flujos de datos para alimentar el modelo son los siguientes:

```
train_generator_flow_1 <- flow_images_from_directory(
  directory = train_images_dir,
  generator = train_images_generator,
  class_mode = 'categorical',
  batch_size = 128,
  target_size = c(64, 64)         # (w x h) --> (64 x 64)
)

validation_generator_flow_1 <- flow_images_from_directory(
  directory = val_images_dir,
  generator = val_images_generator,
  class_mode = 'categorical',
  batch_size = 128,
  target_size = c(64, 64)         # (w x h) --> (64 x 64)
)

test_generator_flow_1 <- flow_images_from_directory(
  directory = test_images_dir,
  generator = test_images_generator,
  class_mode = 'categorical',
  batch_size = 128,
  target_size = c(64, 64)         # (w x h) --> (64 x 64)
```

Lo único que se ha cambiado con respecto al script inicial del profesor ha sido el **batch_size**, que se ha aumentado a **128** para obtener una mejoría en las imágenes.

5.1. Modelo del profesor

En la Figura 19 se muestra el modelo propuesto por el profesor (*modeloProfesor*).

Este modelo utiliza **categorical_crossentropy** para el cálculo de *loss*. También se aplica el algoritmo **RMSPROP** para el cálculo del gradiente, y se encarga de dividir el gradiente por la raíz de la media.

```
modeloProfesor %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')

history <- modeloProfesor %>%
  fit_generator(
    generator = train_generator_flow,
    validation_data = validation_generator_flow,
    steps_per_epoch = 10,
    epochs = 10
  )
```

El resultado de las distintas iteraciones se puede ver en la Figura 20. El resultado obtenido es el siguiente:

- **Tiempo de ejecución:** 4.81 minutos
- **Precisión:** 61 %
- **Pérdida:** 64 %

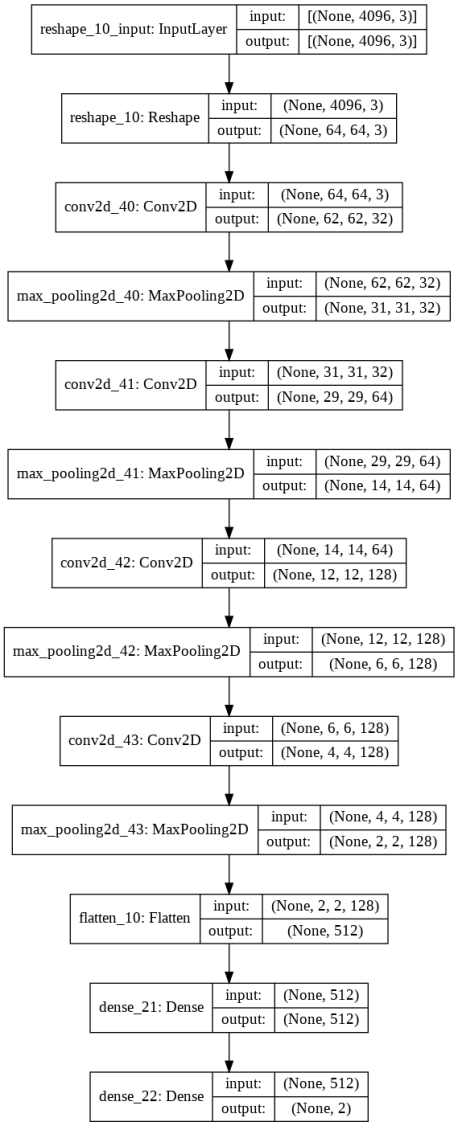

Figura 19: Modelo del profesor

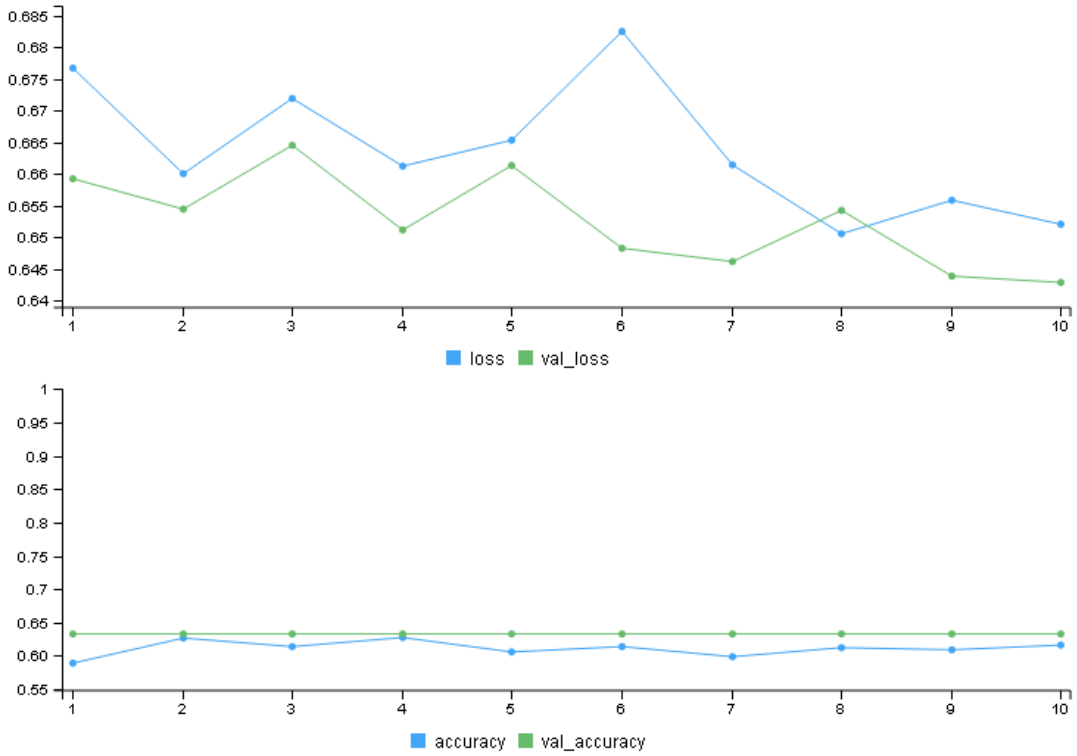

Figura 20: Resultados del modelo del profesor

5.2. Modelo propuesto

En la Figura 21 se muestra el primer modelo propuesto por nosotros (*modeloPropuesto*).

Al igual que en el modelo anterior, este modelo utiliza **categorical_crossentropy** para el cálculo de *loss*. También se aplica el algoritmo **RMSPROP** para el cálculo del gradiente, y se encarga de dividir el gradiente por la raíz de la media.

Nuestro modelo incluye las siguientes mejoras:

- **Cambio en los filtros:** 4 capas convolutivas, empezando la primera capa con 64 filtros y bajando en potencias de 2 hasta 8.
- **Capa densa de 1024 unidades** con activación relu
- **Capa densa de 256 unidades** con activación relu
- **Capa densa de 128 unidades** con activación sigmoid

La última capa se deja igual, con una activación softmax, que funciona muy bien en modelos de clasificación binarios.

```
modeloPropuesto %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')

history <- modeloPropuesto %>%
  fit_generator(
    generator = train_generator_flow,
    validation_data = validation_generator_flow,
    steps_per_epoch = 10,
    epochs = 10
  )
```

El resultado de las distintas iteraciones se puede ver en la Figura 22. El resultado obtenido es el siguiente:

- **Tiempo de ejecución:** 5.64 minutos
- **Precisión:** 60 %
- **Pérdida:** 66 %

Este primer modelo propuesto no mejora la solución propuesta por el profesor, pero debido a que aún se deben implementar nuevas mejoras, se ha decidido dejar este modelo para enfatizar aún más los porcentajes mejorados obtenidos en los siguientes modelos.

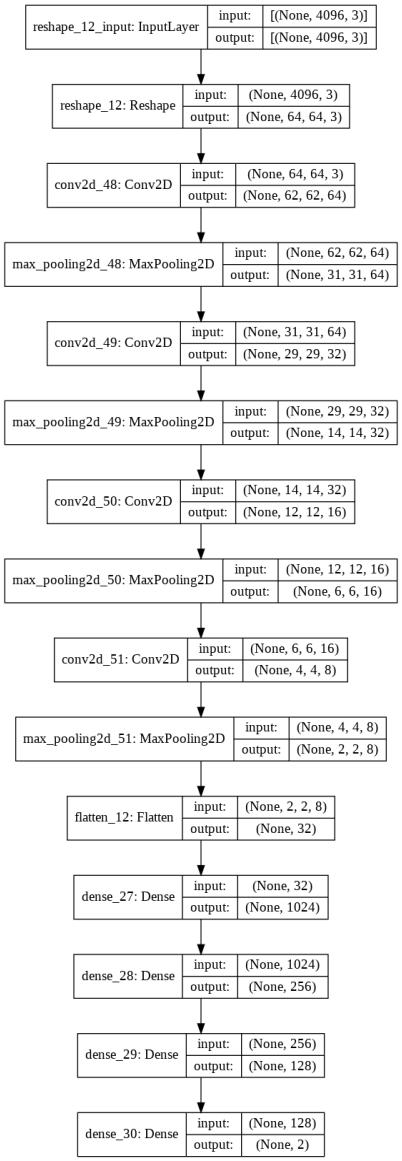

Figura 21: Modelo propuesto

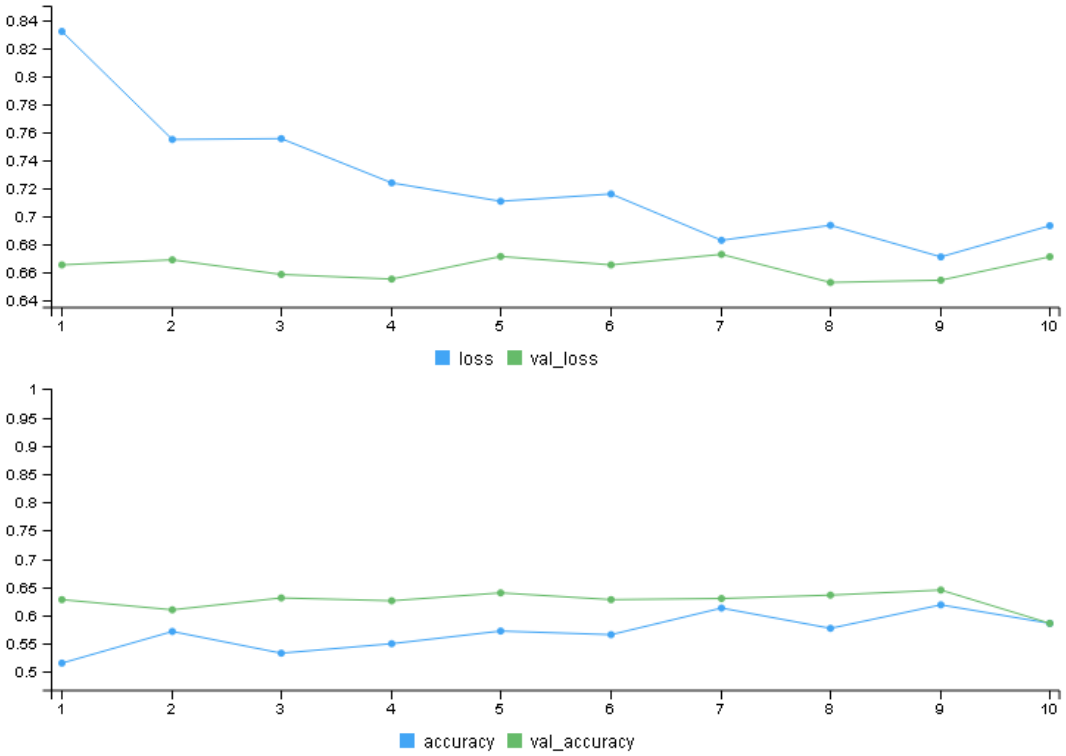

Figura 22: Resultados del modelo del propuesto

5.3. Modelo propuesto mejorado (cambio de relu a sigmoide y viceversa)

En la Figura 23 se muestra el primer modelo propuesto por nosotros (*modeloPropuestoMejorado*).

La mejora implementada en este modelo es el cambio en la función de activación de las capas relu y sigmoide. Aquellas que tenían una función de activación *relu* se ha cambiado a *sigmoide* y viceversa.

```
modeloPropuestoMejorado %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')

history <- modeloPropuestoMejorado %>%
  fit_generator(
    generator = train_generator_flow,
    validation_data = validation_generator_flow,
    steps_per_epoch = 10,
    epochs = 10
  )
```

El resultado de las distintas iteraciones se puede ver en la Figura 24. El resultado obtenido es el siguiente:

- **Tiempo de ejecución:** 5.04 minutos
- **Precisión:** 62 %
- **Pérdida:** 67 %

En este segundo modelo se ha incrementado la precisión conseguida en un 2 %. No es una mejora sorprendente, pero teniendo en cuenta la dificultad del proceso de clasificación de este problema, sí que es una buena mejora.

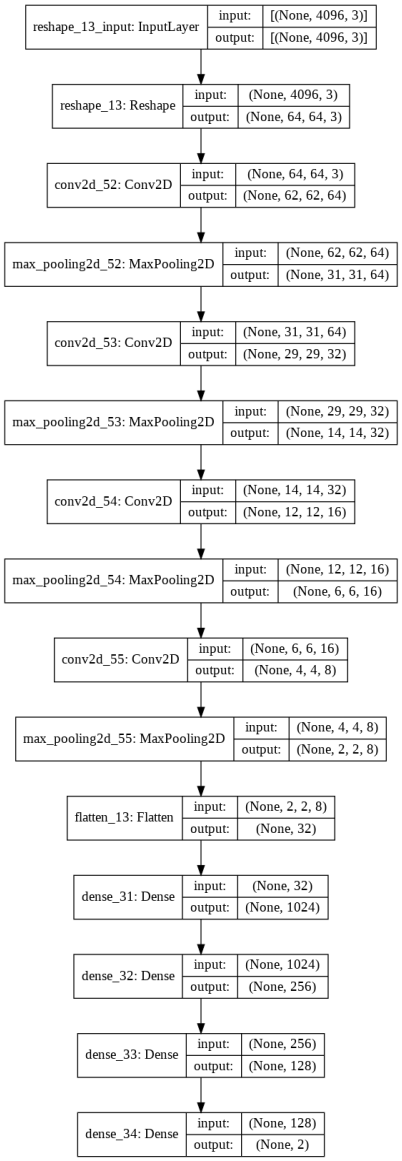

Figura 23: Modelo propuesto mejorado 1

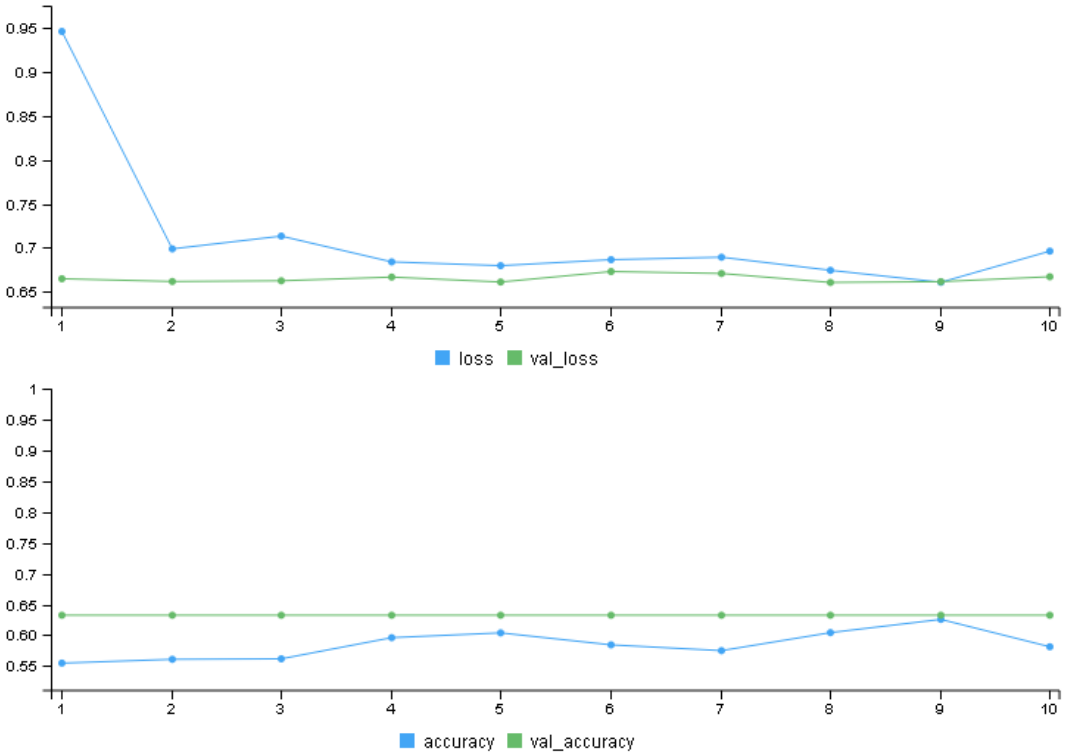

Figura 24: Resultados del modelo propuesto mejorado 1

5.4. Modelo final mejorado (normalización y dropout)

En la Figura 25 se muestra el segundo y último modelo propuesto por nosotros (*modeloPropuestoMejorado*).

Las mejoras implementadas que se han implementado son las siguientes:

- **Layer batch normalization:** la idea de esta técnica es normalizar las activaciones de las capas anteriores en cada lote, de forma que la transformación mantenga la media de activación cercana a 0 y la desviación estándar de activación cercana a 1.
- **Dropout:** con un valor del 50 %, elimina la mitad de las neuronas al azar. El objetivo es reducir el sobreajuste que puede ocasionarse y funciona especialmente bien si se hace en las últimas capas.

```
modeloFinal %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')

history <- modeloFinal %>%
  fit_generator(
    generator = train_generator_flow,
    validation_data = validation_generator_flow,
    steps_per_epoch = 10,
    epochs = 10
  )
```

El resultado de las distintas iteraciones se puede ver en la Figura 26. El resultado obtenido es el siguiente:

- **Tiempo de ejecución:** 5.71 minutos
- **Precisión:** 63 %
- **Pérdida:** 69 %

Este último modelo es el que mejor precisión obtiene. Sí que es cierto que no es una mejora drástica, pero ya se ha comentado que este problema es bastante complicado de mejorar. La precisión obtenida es de un **63 %**.

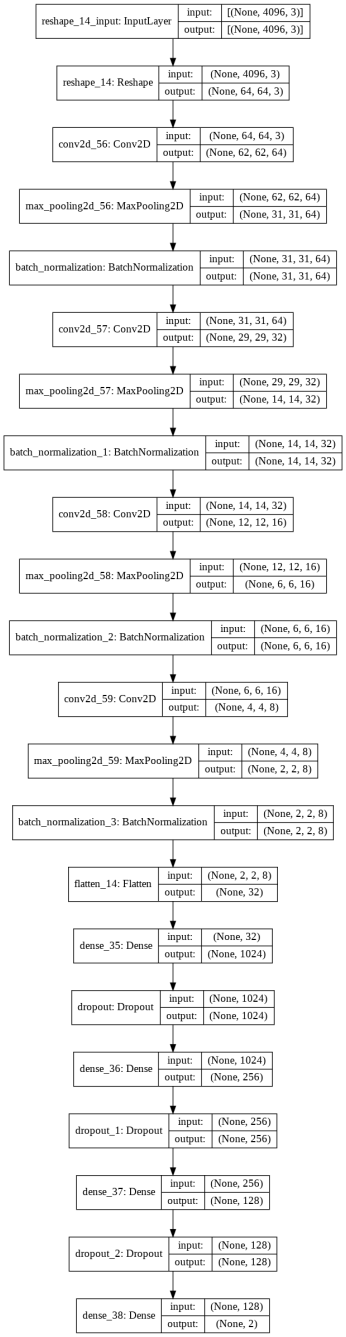

Figura 25: Modelo final mejorado 2

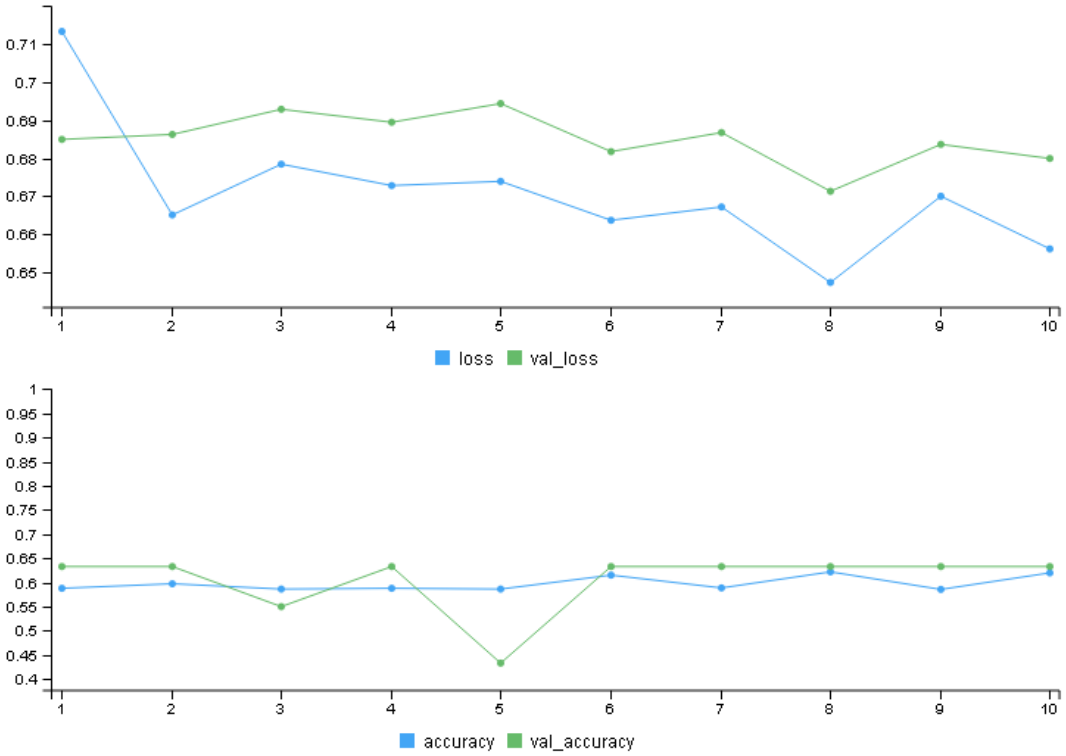

Figura 26: Resultados del modelo final mejorado 2

6. Resultados y conclusiones

En la Tabla 1 se mustran los resultados obtenidos a lo largo de esta práctica.

	Modelo 1	**Modelo 2**	**Modelo 3**	**Modelo 4**
Tiempo de ejecución	4.81 minutos	5.64 minutos	5.04 minutos	5.71 minutos
Precisión	61 %	60 %	62 %	63 %
Pérdida	64 %	66 %	67 %	69 %

Tabla 1: Tabla comparativa de resultados

Se pueden obtener las siguientes conclusiones:

- Debido a la cantidad de imágenes que se ofrecen en el conjunto de datos, este problema requiere de recursos bastante potentes para su resolución.
- Es muy importante el preprocesamiento de datos. En este caso, el único preprocesamiento hecho ha sido la aplicación de la técnica **downsampling**, y esto ha podido influir en la dificultad de mejora de modelos.
- La sociedad está tendiendo a la propagación de noticias falsas, cada día más presente. Es por ello que han surgido organizaciones como **Maldita Bulo**, organización ecargada de desmentir noticias falsas que se propagan a lo largo de las redes sociales y que mucha gente tiende a creerse.
- **No hay teoría exacta para el ajuste de hiperparámetros**. Los hiperparámetros deben acomodarse de acuerdo a cada problema, y sus valores pueden variar dependiendo de cada uno. A pesar de que hay recomendaciones (por ejemplo, utilizar **Dropout** en las capas finales), no hay hiperparámetros fijos.
- Hay una **enorme cantidad de información sobre redes convolutivas** a lo largo de Internet que ha influido en la resolución propuesta en esta práctica. Las mejoras en las redes neuronales son **infinitas**. Un problema de la inteligencia artificial es que **nunca se va a alcanzar un 100 % de precisión** y siempre se va a poder mejorar algo, por lo que se concluye que podría tratarse de algo como un **aprendizaje infinito**.

7. Bibliografía

[1] entitize. Fakeddit. https://github.com/entitize/Fakeddit, 2019.

[2] Keras. https://keras.io/.

[3] Tensorflow. https://www.tensorflow.org/.

[4] Ambika Choudhury. Tensorflow vs keras: Which one should you choose. https://analyticsindiamag.com/tensorflow-vs-keras-which-one-should-you-choose/, 2019.

[5] Tensorflow guide: Keras. https://www.tensorflow.org/guide/keras?hl=es.