



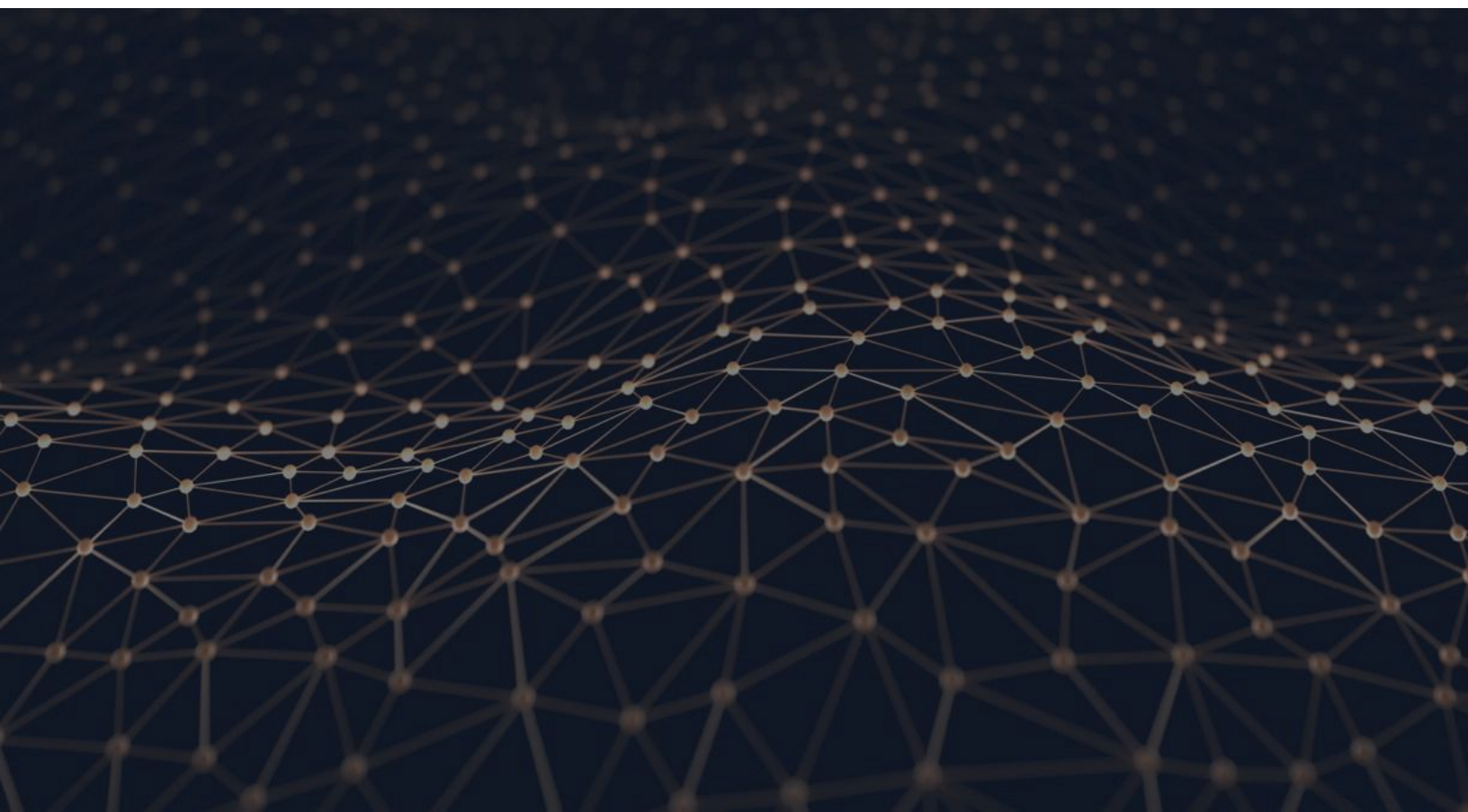
UNIVERSIDAD
DE GRANADA



Deep Learning

Procesamiento de texto

Gema Correa Fernández Andrea Morales Garzón



SISTEMAS INTELIGENTES PARA LA GESTIÓN DE LA EMPRESA, CURSO 2018-2019

MÁSTER PROFESIONAL EN INGENIERÍA INFORMÁTICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE GRANADA

gecorrea@correo.ugr.es y andreamgmg@correo.ugr.es

9 de junio de 2019

Índice de figuras

1.1	Parte del contenido del CSV en training	9
1.2	Parte de las descripciones de las mascotas en training	9
2.1	Contenido del dataset I	11
2.2	Contenido del dataset II	12
2.3	Contenido del dataset III	12
2.4	12
2.5	Nube de palabras sin preprocesar	13
2.6	Contenido del corpus	15
2.7	Frecuencia de los términos	15
2.8	Contenido del corpus sin mayúsculas	16
2.9	Contenido del corpus sin signos de puntuación	16
2.10	Contenido del corpus sin stopwords	17
2.11	Palabras son las que presentan mayor frecuencia en nuestro texto	17
2.12	Palabras son las que presentan mayor frecuencia en nuestro texto	18
3.1	De texto a tokens a vectores	22
3.2	<i>Word embedding</i>	22
3.3	Funcionamiento CNN 1D	26
3.4	Funcionamiento RNN	28
4.1	Ejemplo de Visualización con Convnets	33
4.2	Visualización en Convnets I	34
4.3	Visualización en Convnets II	34
4.4	Visualización en Convnets III	34
4.5	Visualización con HeatMap I	35
4.6	Visualización con HeatMap II	36
4.7	Visualización con HeatMap III	36

4.8	Modelo a crear para el ensemble datos textuales + datos numéricos	40
-----	---	----

Índice general

1	Introducción	7
1.1	Descripción del trabajo de investigación	7
1.1.1	Comprender el problema a resolver	7
1.1.2	Consideraciones a tener en cuenta	9
1.1.3	Estructura de la práctica	9
2	Análisis exploratorio	11
2.1	Análisis exploratorio	11
2.1.1	Representación gráfica de los datos	12
2.1.2	Nube de palabras	13
2.2	Primeros pasos con Text Mining	14
2.2.1	Creación del corpus	14
2.2.2	Representación gráfica de las frecuencias del corpus	15
2.2.3	Conversión a minúsculas	16
2.2.4	Eliminación de los signos de puntuación	16
2.2.5	Eliminación de Stopwords	16
2.2.6	Agrupación de sinónimos	17
3	Usando pretrained word embeddings	21
3.1	Clasificación de textos con Redes Neuronales	21
3.2	Word Embedding	22
3.2.1	Configuración del modelo	23
3.3	Redes neuronales convolucionales	26

3.4	Redes neuronales recurrentes	28
3.5	Redes neuronales recurrentes + CNN	29
3.6	Análisis de los resultados	31
4	Modelos Ensemble	33
4.1	Redes Neuronales Convolucionales	33
4.1.1	Visualización en convnets	33
4.1.2	Ensemble con Convnets	35
4.1.3	Weight Average Ensemble con Convnets	37
4.2	Ensemble con Procesamiento de Textos	38
4.2.1	Weight Average Ensemble con datos del CSV	38
4.2.2	Modelos RRNN de entrada y salida múltiple	38
5	Conclusiones	43
5.1	Conclusiones	43
5.2	Otros trabajos que se podrían realizar	44
6	Referencias	47

1. Introducción

1.1 Descripción del trabajo de investigación

La evaluación para la parte teórica de la asignatura va a consistir en realizar un trabajo de investigación, que extienda la parte técnica correspondiente a la implementación de modelos basados en Redes Neuronales para resolver problemas de clasificación.

Con este trabajo, se pretende entrar más en profundidad en alguna técnica, En concreto nos vamos a centrar en técnicas basadas en **procesamiento de texto**, con el fin de obtener una predicción en base al anuncio que acompaña la foto de las mascotas. Para su realización, se va hacer uso de datos aportados en la *Práctica 2* los cuáles no se utilizaron para el desarrollo de dicha tarea. **Por tanto se podría decir, que este trabajo es una continuación directa de la Práctica 2, y que continúa básicamente en el punto en el que se dejó.**

Durante dicha práctica, se manejaron imágenes de mascotas, con el fin de predecir las etiquetas a partir de las características de las mismas. Esta vez vamos a incorporar los datos aportados en el fichero con extensión .csv, para intentar obtener resultados para dicho problema, que posteriormente puedan, o bien juntarse, o bien utilizarse de alguna manera para mejorar los resultados de la predicción. El objetivo del problema se mantiene: tendremos que predecir el tiempo de adopción de una mascota.

1.1.1 Comprender el problema a resolver

De primeras, partimos de la idea de centrarnos únicamente en la descripción asociada a cada mascota para la obtención de diversos modelos¹.

Como idea inicial, vamos a realizar un **análisis de sentimientos** a partir de las descripciones de las mascotas a adoptar. Dicho análisis nos va a servir como exploración de los datos, para permitir comprenderlos mejor y seleccionar el camino que mejor se adapte a nuestro problema en el alcance

¹Sin embargo, a lo largo del documento se observarán cómo es necesario incorporar distintos datos y diversas técnicas, como pueden ser las redes neuronales

de nuestro objetivo. Se ha llevado a cabo esta primera tarea, porque nos parece esencial comprender las características del texto con las que vamos a trabajar, además de ayudarnos a ver, de manera global, con qué tipo de contenido estamos tratando

Como ya se comentó en la práctica anterior, el problema con el que estamos tratando es claramente subjetivo, y en el que las emociones que se transmiten guardan un papel muy importante. Por ello, es de esperar, que el anuncio de la mascota también tenga una relación con esta subjetividad. El uso de unas palabras u otras (así como lo que se transmite en dichas palabras) suele tener una fuerte dependencia respecto la adopción de la mascota, ya que va ligado con sentimientos y otras emociones:

- Si en la descripción nos encontramos términos preferentemente *tristes*, como por ejemplo que dicha mascota fue encontrada en la calle con signos de desnutrición, esto puede causar que algunas personas empaticen y la adopten.
- Pero por otro lado, haciendo uso de un lenguaje completamente distinto, es posible obtener el mismo resultado.

Esto nos determina que seguimos manteniendo la componente subjetiva de la que hablamos en la *Práctica 2*, con las consecuencias que ya hemos comentado que eso conlleva en nuestros resultados. Por eso mismo, el lenguaje es un gran aliado para la realización del análisis de sentimientos. Asimismo, este tema es muy subjetivo y requiere un estudio y análisis adecuado para sus conclusiones. Adicionalmente, como objetivo secundario es interesante obtener las dependencias y patrones existentes entre las distintas mascotas, para conocer aspectos sobre ellas o incluso realizar agrupaciones.

El dataset aportado en la *Práctica 2*, contiene una exhaustiva base de datos de mascotas proporcionada en CSV, en la cual se puede apreciar información muy diversa desglosada a través del nombre de la mascota, la edad, el color de pelo, si está vacunado, si está esterilizado, la salud que tiene, el identificador o la descripción de la mascota (ver figura 1.1). Contiene una dimensión de 14993 filas por 24 columnas. Por otro lado, se dispone de un CSV con el conjunto de datos test sobre el que se debería de probar la efectividad de la red que hemos entrenado previamente, con una dimensión de 3972 filas y 23 columnas (se omite la columna con las etiquetas). Asimismo, como se hizo con las imágenes en la práctica, es necesario disponer de la clase asociada a cada mascota, es por eso que se hace la partición de los datos solo con el conjunto training.

De igual forma a como se ha realizado en la *Práctica 2*, lo primero es llevar a cabo un particionamiento de los datos, con los que podamos trabajar en el futuro a la hora de llevar a cabo las predicciones correspondientes. El código asociado a este particionamiento se encuentra a continuación, y, tal y como se puede ver, se realiza de manera análoga a la práctica anterior (el 80 % de los datos será para el conjunto Train, y un 20 % restante para test).

```
1 # Particionamiento de los datos
2 trainIndex <- createDataPartition(data.train$AdoptionSpeed, p = .8, list =
  FALSE, times = 1)
3 train <- data.train[trainIndex, ]
4 test <- data.train[-trainIndex, ]
```

Como se ha comentado, nos vamos a centrar principalmente en la descripción asociada a la mascota. En ese campo campo nos podemos encontrar información muy diversa de la mascota como las condiciones en las que se la encontraron o como es (ver figura 1.2). Por tanto, a partir de estos

	Type	Name	Age	Breed1	Breed2	Gender	Color1	Color2	Color3	MaturitySize	FurLength	Vaccinated	Dewormed
1	2	Nibble	3	299	0	1	1	7	0	1	1	2	
2	2	No Name Yet	1	265	0	1	1	2	0	2	2	3	
3	1	Brisco	1	307	0	1	2	7	0	2	2	1	
4	1	Miko	4	307	0	2	1	2	0	2	1	1	

Figura 1.1: Parte del contenido del CSV en training

Description
Nibble is a 3+ month old ball of cuteness. He is energetic and playful. I rescued a couple of cats...
I just found it alone yesterday near my apartment. It was shaking so I had to bring it home to pr...
Their pregnant mother was dumped by her irresponsible owner at the roadside near some shop...
Good guard dog, very alert, active, obedience waiting for her good master, plz call or sms for m...
This handsome yet cute boy is up for adoption. He is the most playful pal we've seen in our pup...
This is a stray kitten that came to my house. Have been feeding it, but cannot keep it.
anyone within the area of ipoh or taiping who interested to adopt my cat can contact my father ...
Siu Pak just give birth on 13/6/10 to 6puppies. Interested pls call or sms me. Left 2female pupp...
healthy and active, feisty kitten found in neighbours' garden. Not sure of sex.
Very manja and gentle stray cat found, we would really like to find a home for it because we can...
For serious adopter, please do sms or call for more details, thanks!
Kali is a super playful kitten who is on the go the minute she wakes up. She is very quiet and on...
Peanut was an abused puppy until he was rescued. He was very scared of people but now he is a...
Hi Pet Lovers! This is my first posting and I need help! 3 months ago we befriended a mother str...
Lost Dog Found (Bandar Menjalara, Kepong / Taman Bukit Maluri or Desa Park City) 迷途犬被尋獲...
We moved out of our apartment to a landed home and there were many friendly strays in the nei...
to be spayed on /12 adorable & friendly
shes active... she can obey wht command that u told her.. example shakeshand . sleep and eat ...

Figura 1.2: Parte de las descripciones de las mascotas en training

datos nuestro objetivo inicial es realizar un pequeño análisis exploratorio de los datos, con el que determinar el comportamiento y tratamiento a seguir con ellos.

1.1.2 Consideraciones a tener en cuenta

Para la lectura del presente trabajo de investigación, es muy aconsejable leer previamente la documentación aportada en la *Práctica 2*. Además, para la implementación se va hacer uso de las mismas herramientas: **R + Keras**.

1.1.3 Estructura de la práctica

En este primer apartado, se ha explicado el problema que vamos a abarcar a lo largo del trabajo. A continuación, se detallan de forma resumida el contenidos del resto de apartados del documento:

- En el *apartado 2*, se detalla el análisis exploratorio de los datos a partir del análisis de sentimientos realizado a la parte textual.
- En el *apartado 3*, se detalla el uso de *Pretrained Word Embedding* como posible mejora para

la predicción del tiempo de adopción de la mascota.

- En el *apartado 4*, se detalla el uso de *ensembles* como posible mejora para la predicción del tiempo de adopción de la mascota.
- En el *apartado 5*, se desarrollan las conclusiones de los resultados obtenidos en el trabajo.



Para la realización de la práctica se usarán de manera complementaria referencias de Internet, la documentación proporcionado por Keras (<https://keras.rstudio.com>), como el libro de *Deep Learning with R* (<https://www.manning.com/books/deep-learning-with-r>).

2. Análisis exploratorio

2.1 Análisis exploratorio

Previamente a la aplicación de cualquier técnica, es necesario comprender el comportamiento de los datos, para establecer así el camino a seguir, y poder preveer distintas dificultades que podamos encontrarnos. Por ello mismo y a partir de nuestros conocimientos en análisis de sentimientos, se ha optado por realizar un previo procesamiento al conjunto de datos training, el cual se particionará para la obtención del conjunto para test y para validación. A continuación, una vez el leído el CSV para training, se realizan las siguientes operaciones:

1. Representación gráfica de los datos
2. Nube de palabras
3. Creación del corpus para la columna *Description*
4. Representación gráfica de las frecuencias del corpus
5. Conversión a minúsculas
6. Eliminación de los signos de puntuación
7. Eliminación de Stopwords

Como hemos comentado, este procesamiento nos sirve como análisis exploratorio de los datos. En las figuras siguientes podemos apreciar parte del contenido del dataset.

	Type <int>	Name <text>	Age <int>	Breed1 <int>	Breed2 <int>	Gender <int>	Color1 <int>	Color2 <int>	Color3 <int>
1	2	Nibble	3	299	0	1	1	7	0
2	2	No Name Yet	1	265	0	1	1	2	0
3	1	Brisco	1	307	0	1	2	7	0
4	1	Miko	4	307	0	2	1	2	0
5	1	Hunter	1	307	0	1	1	0	0

5 rows | 1-10 of 24 columns

Figura 2.1: Contenido del dataset I

MaturitySize	FurLength	Vaccinated	Dewormed	Sterilized	Health	Quantity	Fee	State	RescuerID
1	1	2	2	2	1	1	100	41326	8480853f516546f6cf33aa88cd76c379
2	2	3	3	3	1	1	0	41401	3082c7125d8fb66f7dd4bff4192c8b14
2	2	1	1	2	1	1	0	41326	fa90fa5b1ee11c86938398b60abc32cb
2	1	1	1	2	1	1	150	41401	9238e4f44c71a75282e62f7136c6b240
2	1	2	2	2	1	1	0	41326	95481e953f8aed9ec3d16fc4509537e8

5 rows | 11-20 of 24 columns

Figura 2.2: Contenido del dataset II

Description
Nibble is a 3+ month old ball of cuteness. He is energetic and playful. I rescued a couple of cats a few months ago but could not get them neutered in time as the ...
I just found it alone yesterday near my apartment. It was shaking so I had to bring it home to provide temporary care.
Their pregnant mother was dumped by her irresponsible owner at the roadside near some shops in Subang Jaya. Gave birth to them at the roadside. They are all h...
Good guard dog, very alert, active, obedience waiting for her good master, plz call or sms for more details if you really get interested, thanks!!
This handsome yet cute boy is up for adoption. He is the most playful pal we've seen in our puppies. He loves to nibble on shoelaces , Chase you at such a young a...

5 rows | 22-22 of 24 columns

Figura 2.3: Contenido del dataset III

2.1.1 Representación gráfica de los datos

Antes de comenzar con el análisis exploratorio, hemos realizado un diagrama de barras vamos con el fin de comprender mejor los datos. En el siguiente diagrama, podemos ver la cantidad de mascotas adoptadas en función del tiempo que tardarán en adoptarlas.

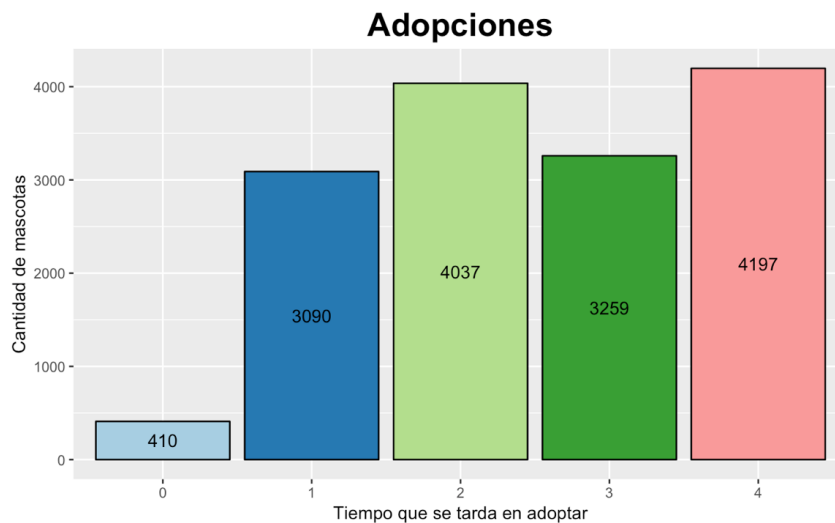


Figura 2.4:

Las variables categóricas que podemos encontrarnos están determinadas de la misma manera que antes, es decir, en un rango entre 0 y 4, en donde el **0** hace referencia al **mínimo tiempo** que pasa la mascota desde que se recoge hasta que se adopta y el **4**, hace referencia al **máximo tiempo** que pasa la mascota desde que se recoge hasta que se adopta.

Por otro lado, se podría hacer un estudio independiente a partir de las conclusiones obtenidas aquí, es decir, ¿es rentable dejar la etiqueta 0 sola? ¿no sería mejor juntarlo con la clase 1? Por otro lado, estamos ante un problema de multclasificación, pero ¿qué pasaría si lo convertimos a clasificación binaria? Es decir, predecir el tiempo que tarda un mascota en ser adoptada puede ser muy subjetivo, por lo que se podría pensar en realizar una clasificación teniendo en cuenta dos cosas:

1. Si la mascota se adopta en un tiempo corto.
2. Si la mascota se adopta en un tiempo largo.

Estos conceptos que acabamos de obtener, pueden ser temas a tratar en un trabajo futuro. Sin embargo, debido a haber querido realizar una continuación de la *Práctica 2*, nos vamos a centrar en usar las etiquetas tal como vienen. Teniendo en cuenta esta clasificación, se observa como el 50% de las mascotas dispone de un tiempo prolongado para su adopción.

2.1.2 Nube de palabras

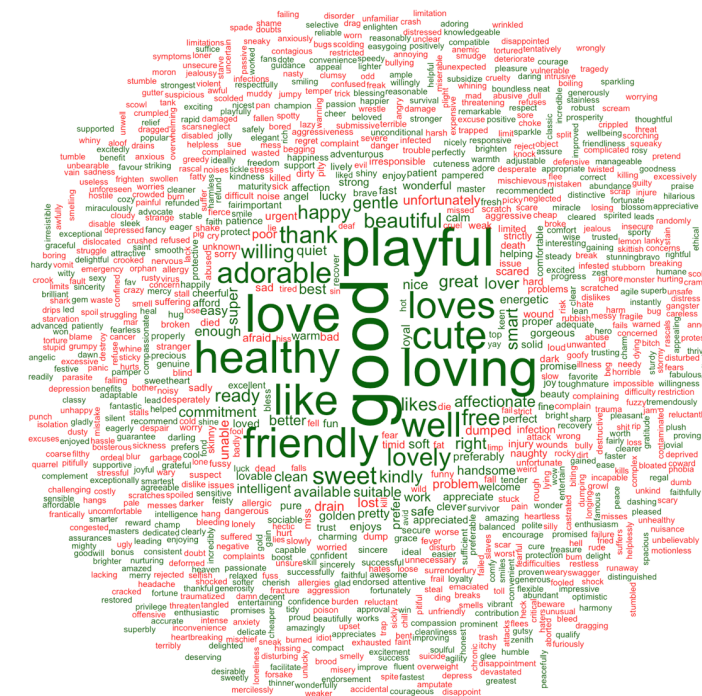


Figura 2.5: Nube de palabras sin preprocesar

Como comentamos al principio del presente trabajo, hacer uso del lenguaje conlleva una componente subjetiva. Así que para obtener las palabras usadas en el mismo, se ha pensado en visualizar los datos textuales haciendo uso de una **nube de palabras**. Una nube de palabras, no es más que una representación visual de las palabras que conforman un texto, en donde el tamaño es mayor para las palabras que aparecen con más frecuencia. Asimismo, se ha establecido un color **rojo** para las palabras que tienen connotación negativa y color **verde** para las palabras con connotación positiva. A continuación, se visualiza la nube de palabras con los datos textuales que aparecen en las descripciones.

nes sobre las mascotas para adoptar.

En la imagen de antes, se observa como existe un predominio del color negativo por los bordes. No obstante, las palabras más importantes hacen referencia a palabras positivas, como *good*, *love*, *like*, *friendly*, *adorable*, *love*, *cute*, *happy* o *playful*. Ya que el uso de estos términos determina que las mascotas son *buenas*, para así adoptarlas lo antes posible. Por lo que no es de extrañar, que por el tema que estamos tratando, las palabras con más importancia dentro de las descripciones sean positivas.

Por tanto, una vez visualizado y entendido el conjunto de datos, en mayor medida, podemos empezar con el procesamiento a los mismos. Para ello, lo primero tenemos que hacer es cargar la librería que procesa los datos de tipo texto en R, para la construcción y manipulación del corpus. La librería más conocida se llama *tm*. Para este análisis inicial, solo se va hacer uso del conjunto de training.

2.2 Primeros pasos con Text Mining

Una vez habíamos visualizado los datos, partimos de la idea de realizar nuestra propia tokenización para las palabras del texto, mediante la aplicación de diversos métodos de análisis de sentimientos, a partir de la creación del corpus para la variable *Description*. Sin embargo, cuando llegamos a la agrupación de sinónimos, se nos complicó la tarea de seguir por este camino, ya que no nos era posible aplicar dichos métodos a todo el conjunto del dataset. Por lo que se obtuvieron que explorar otras alternativas.

2.2.1 Creación del corpus

Para poder obtener la estructura con la que vamos a procesar nuestra información, debemos obtener un vector con documentos. En nuestro caso, cada uno de los documentos se corresponde con una descripción sobre la mascota (*Description*). Para ello, primero debemos de construir un vector con todas las descripciones del dataframe y convertir cada elemento del vector al formato de documento. Podemos usar la función *VectorSource* para hacer esta conversión.

```

1 # Nos quedamos con la columna del dataset que nos interesa.
2 # Necesitamos obtenerla en forma de vector, y no como un dataframe
3 # por lo que usamos as.vector para hacer la conversion
4 data.train.description <- data.train$Description
5 data.train.description <- as.vector(data.train.description)
6
7 # Lo convertimos en la estructura de documento,
8 # y lo guardamos ya en el corpus que lo vamos a utilizar
9 data.train.description.corpus <- (VectorSource(data.train.description))
10
11 # Creamos el propio corpus
12 data.train.description.corpus <- Corpus(data.train.description.corpus)

```

Podemos ver que funciona accediendo a uno cualquiera:

```

1 # Mostrar el contenido por pantalla
2 data.train.description.corpus[[4]]$content
3 data.train.description.corpus[[7]]$content

```

Así que ya tenemos los datos en el formato adecuado, para comenzar a realizar el análisis.

```
[1] "Good guard dog, very alert, active, obedience waiting for her good master, plz call or sms for more details if you really get
interested, thanks!!"
[1] "anyone within the area of ipoh or taiping who interested to adopt my cat can contact my father at this number (mazuvi)or can just email
me. currently bulat is at my hometown at perak but anyone outside the area still want to adopt can travel there to my hometown.there is a lot
of cats in my house rite now..i think i should let one of them go to a better owner who can give better attention to him."
```

Figura 2.6: Contenido del corpus

2.2.2 Representación gráfica de las frecuencias del corpus

Una vez creado el corpus, vamos a visualizar las frecuencias para la columna con la que vamos a trabajar, es decir, la variable *Description*. Para ello, debemos calcular la matriz de términos y obtener los términos con mayor frecuencia.

```
1 # Cargamos la libreria y calculamos la matriz de terminos
2 dtm <- DocumentTermMatrix(data.train.description.corpus)
3
4 # Calculamos la frecuencia
5 freq <- sort(colSums(as.matrix(dtm)), decreasing=FALSE)
6 wf <- data.frame(word=names(freq), freq=freq)
7
8 # Dibujamos el histograma
9 subset(wf, freq>3050) %>%
10 ggplot(aes(word, freq)) +
11   geom_bar(stat="identity", fill="darkred", colour="black") +
12   theme(axis.text.x=element_text(angle=45, hjust=1)) +
13   ggtitle("Frecuencias para Description en train")
```

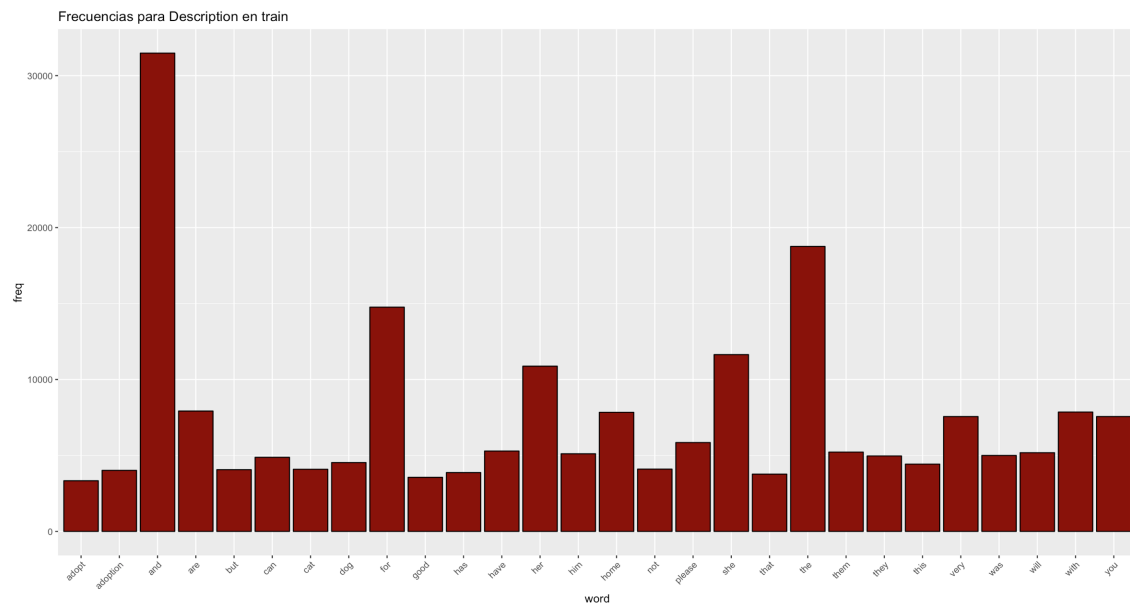


Figura 2.7: Frecuencia de los términos

Como se puede apreciar en la imagen 2.7, algunos de los términos que más se repiten son *the*, *adopt*, *adoption*, *and*, *are*, *but*, *cat*, *dog*... Si tenemos en cuenta que estamos tratando las adopciones de perros y gatos, es lógico que las palabras que más se repiten sean las relacionadas con adopción,

perros y/o gatos. Pero por otro lado, algunas de esas palabras que podemos encontrar, no nos aportan información alguna, como *and*, *the*, *are* o *but*. Es por eso, que se debe de hacer una transformación a dichos términos, como su eliminación de los *Stopwords*, su conversión a minúsculas o la eliminación de signos de puntuación.

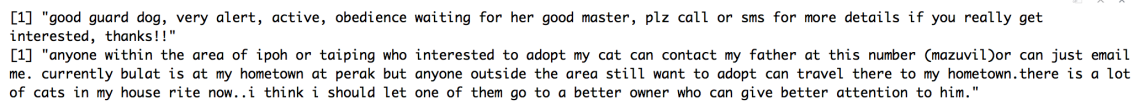
2.2.3 Conversión a minúsculas

Para el tratamiento de los datos textuales es aconsejable hacer uso de los términos por igual, es decir, debemos convertir las mayúsculas en minúsculas. Normalmente se convierte en minúsculas todas las letras para que los comienzos de oración no sean tratados de manera diferente por los algoritmos.

```
1 data.train.description.corpus <- tm_map(data.train.description.corpus ,
2                                         content_transformer(tolower))

1 # Mostrar el contenido por pantalla
2 data.train.description.corpus[[4]]$content
3 data.train.description.corpus[[7]]$content
```

Podemos ver que funciona accediendo a uno cualquiera:



```
[1] "good guard dog, very alert, active, obedience waiting for her good master, plz call or sms for more details if you really get
interested, thanks!!"
[1] "anyone within the area of ipoh or taiping who interested to adopt my cat can contact my father at this number (mazuvil)or can just email
me. currently bulat is at my hometown at perak but anyone outside the area still want to adopt can travel there to my hometown.there is a lot
of cats in my house rite now..i think i should let one of them go to a better owner who can give better attention to him."
```

Figura 2.8: Contenido del corpus sin mayúsculas

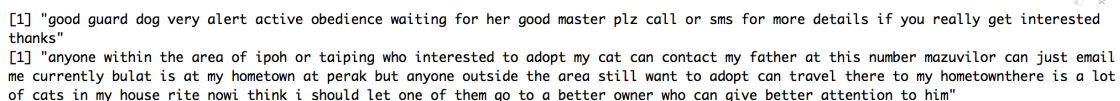
Como se observa, han desaparecido las mayúsculas, lo que facilita el tratamiento de todos los datos por igual, tanto al principio de la frase como al final.

2.2.4 Eliminación de los signos de puntuación

En el subapartado anterior podemos ver como el documento presenta exclamaciones, signos de puntuación y sinónimos. En un principio, no tiene sentido en *Data Mining* contemplar los signos de puntuación, ya que no nos van a aportar información.

```
1 data.train.description.corpus <- tm_map(data.train.description.corpus ,
2                                         content_transformer(removePunctuation))
```

Podemos ver que funciona accediendo a uno cualquiera:



```
[1] "good guard dog very alert active obedience waiting for her good master plz call or sms for more details if you really get interested
thanks"
[1] "anyone within the area of ipoh or taiping who interested to adopt my cat can contact my father at this number mazuvilor can just email
me currently bulat is at my hometown at perak but anyone outside the area still want to adopt can travel there to my hometownthere is a lot
of cats in my house rite nowi think i should let one of them go to a better owner who can give better attention to him"
```

Figura 2.9: Contenido del corpus sin signos de puntuación

2.2.5 Eliminación de Stopwords

En cualquier idioma, hay palabras que son tan comunes o utilizadas que no aportan información relevante, a dichas palabras se las conoce como *stopwords* o palabras *stop*. Por ejemplo, en español, las palabras 'la', 'a', 'en', 'de' son ejemplos de *stopwords*. Este tipo de palabras debemos de

suprimirlas de nuestro corpus. Como, en nuestro caso, el contenido del corpus está en inglés, debemos especificar el idioma correcto para que nos elimine del corpus las palabras adecuadas en dicho idioma, tanto en train como en test.

```
1 data.train.description.corpus <- tm_map(data.train.description.corpus,
2     content_transformer(removeWords), stopwords("english"))
```

Podemos ver que funciona accediendo a uno cualquiera:

```
[1] "good guard dog alert active obedience waiting good master plz call sms details really get interested thanks"
[1] "anyone within area ipoh taiping interested adopt cat can contact father number mazuvilor can just email currently
bulat hometown perak anyone outside area still want adopt can travel hometownthere lot cats house rite now think
let one go better owner can give better attention "
```

Figura 2.10: Contenido del corpus sin stopwords

Se aprecia en la salida anterior como existen más huecos entre algunas palabras, lo que sugiere que ha eliminado aquellas que carecen de significado. No obstante antes de aplicar al eliminación de los espacios en blanco, es preferible agrupar algunas palabras por sus sinónimos.

2.2.6 Agrupación de sinónimos

Con el fin de disminuir la dimensión del espacio a trabajar, se pueden identificar palabras distintas con el mismo significado y reemplazarlas por una única palabra. Para ello, se deben obtener los sinónimos para esa palabra. Dentro de las librerías que podemos usar para agrupar sinónimos, destacamos dos: wordnet y rword2vec. Sin embargo, por su sencillez y facilidad de uso, nos hemos decantado por rword2vec. Lo primero que tenemos que obtener, es saber qué palabras son las que presentan mayor frecuencia en nuestro texto.

home	please	will	dog	can	adoption	cat	good	adopt
7610	5709	5168	4355	4244	3934	3837	3524	3257
give	found	old	contact	interested	looking	playful	love	loving
2909	2811	2782	2773	2749	2722	2651	2640	2601
now	call	kitten	care	one	puppy	cats	healthy	take
2589	2524	2520	2454	2334	2248	2231	2205	2165
owner	rescued	like	active	house	new	kittens	friendly	months
2165	2151	2080	2077	2060	2032	2012	1983	1930
dogs	food	cute	puppies	time	adopter	loves	must	little
1901	1856	1851	1813	1789	1786	1765	1736	1727
family	need	around	female	just	find	keep	well	adopted
1700	1695	1663	1620	1617	1557	1525	1471	1366
also	mother	male	trained	adorable	get	vet	people	long
1327	1317	1306	1301	1259	1249	1236	1197	1195
already	pls	help	pet	hope	forever	shes	back	really
1190	1185	1171	1150	1143	1103	1089	1083	1080
play	someone	litter	lovely	black	still	sweet	dewormed	stray
1054	1052	1044	1007	1000	979	978	970	964

Figura 2.11: Palabras son las que presentan mayor frecuencia en nuestro texto

Una vez que tenemos los términos con mayor frecuencia para nuestros texto y su frecuencia asociada, pasamos a obtener la matriz de dichos datos, con el fin de obtener únicamente las palabras y descartar su frecuencia asociada.

Como ya sabemos las palabras a usar, es decir, los términos que más se repiten, procedemos a la agrupación por sinónimos. En donde, mediante la función `distance(...)` de la librería `rword2vec`, obtendremos todas palabras más similares de nuestro conjunto. A continuación, se muestran los pasos seguidos.

1. Escribir un fichero los datos asociados a dicha columna.
2. Entrenar los datos del fichero, con el fin de obtener los vectores de palabras que nos darán los palabras más similares.
3. Obtener para cada término del documento, la distancia con los términos del ficheros, quedándonos con las de mayor frecuencia.
4. Guardar en un fichero dichas distancias.

```

1  # Escribo en un fichero la columna "description"
2  write.table(data.train$Description[1:4000], "description.txt",
3              sep = "\t", quote = F, row.names = F)
4
5  # Entreno los datos del texto para obtener los vectores de palabras
6  model.train.description = word2vec(train_file = "description.txt",
7                                     output_file = "description.bin", binary=1)
8
9  dist_terms_benefits_train_corpus = c()
10
11 # Obtengo la distancia de las 100 palabras con mayor frecuencia
12 # Calculamos la distancia de la palabra a sus sin nimos
13 for (i in 1:length(terms.train.description.corpus)){
14     dist_terms_benefits_train_corpus[i] = distance(
15                                     file_name = "description.bin",
16                                     search_word = terms.train.description.corpus[i],
17                                     num = 2)
18 }

```

En el código anterior, hemos entrenado con nuestros datos, para obtener para cada palabra las dos palabras más similares a éstas. En la siguiente figura, se aprecian algunos ejemplos, como sinónimo de *cat* hace uso de *kitten* o *dog*, lo que nos hace estar comprender que esas tres palabras se usan en el mismo contexto, al estar hablar de mascotas.

```

[1] "will"
[1] could should
Levels: could should
[1] "cat"
[1] dog kitten
Levels: dog kitten
[1] "give"
[1] find provide
Levels: find provide
[1] "girl"
[1] boy puppy
Levels: boy puppy

```

Figura 2.12: Palabras son las que presentan mayor frecuencia en nuestro texto

Por tanto, el resultado que acabamos de obtener es coherente, por lo que podríamos seguir realizado dicho procesamiento o análisis de sentimientos. Sin embargo, debido a las dimensiones de los textos, es decir, todas las palabras por las que conforman una frase, no es imposible aplicar el entrenamiento haciendo uso de todas las filas, por lo que se debe limitar dicho entrenamiento a 4000 filas del dataset.

Esto supone una desventaja, en cuanto a estudio de nuestros datos, por lo que nos limita seguir esta línea.

Al llegar a este bache, nos encontramos con varias limitaciones, la más importante que no es posible hacer de todo el conjunto del dataset, y que computacionalmente es poco viable. Por lo que después de varias discusiones, entran en juego las redes neuronales, vistas en la *Práctica 2*, con la aplicación de otros modelos asociados a los datos textuales. Por tanto, dejamos esta vía de trabajo, y comenzamos a utilizar Redes Neuronales con el objetivo de suplir las dificultades encontradas en nuestro tratamiento, hasta ahora manual, de los datos textuales de los que disponemos.

3. Usando *pretrained word embeddings*

3.1 Clasificación de textos con Redes Neuronales

Como conclusión del apartado anterior obtuvimos que no es posible partir de un tratamiento manual de nuestros datos con el fin de resolver este problema (por dificultades, como hemos comentado, de recursos principalmente). En la referencia [Bro], nos aconseja que el *modus operandi* para la clasificación de textos implica el uso de una palabra incrustada para representar palabras y una red neural convolucional para aprender a discriminar documentos sobre problemas de clasificación. En este capítulo, seguiremos esta línea de trabajo, y expondremos los pasos así como resultados obtenidos.

A lo largo de la literatura se aconseja hacer uso de *Deep Learning* para el procesamiento del lenguaje natural, ya que permite el reconocimiento de patrones aplicado a palabras, oraciones y párrafos. Podemos aquí hacer una analogía respecto al problema tratado en la práctica 2: mientras que aquí tenemos Deep Learning que nos permite obtener información de los textos, en la práctica 2 teníamos los métodos de Visión por Computador (entran aquí aspectos como la *convolución*) para permitirnos obtener buenos resultados a partir de encontrar patrones en los píxeles. A lo largo de este capítulo veremos distintas formas de abordar el problema de procesamiento de textos.

Sin embargo, tal y como hemos visto hasta ahora, y como era de esperar, no podemos proporcionar, como entrada de una red neuronal, los textos no estructurados. Ya sabemos que las redes neuronales trabajan con los llamados *Numerical Tensors*¹. Es por ello que, debemos realizar una transformación previa a nuestros datos, para poder proporcionarle a la red los datos de la manera correcta. Por tanto, debemos realizar una configuración numérica del contenido de los textos, que pueda ser interpretable por la red. Para ello, tendremos que dividir el texto en el elemento mínimo de representación (palabras, frases, caracteres...), y transformar cada uno de esos elementos en un vector. Las diferentes unidades en las que puede desglosar el texto (palabras, caracteres o n-gramas) se denominan *tokens*,

¹Matemáticamente, cuando hablamos de Numerical Tensors estamos haciendo referencia a una especie de contenedor de datos, el cuál nos permite agrupar o reunir diferentes datos en una dimensión N

y la división del texto en dichos tokens se denomina tokenización (ver figura 3.1).

A partir de la idea que acabamos de comentar, principalmente se harán uso de las siguientes técnicas, expuestas a continuación.

- Word Embedding
- Redes neuronales convolucionales
- Redes neuronales recurrentes
- Redes neuronales recurrentes + CNN

La filosofía, se basará en introducir los datos de entrada (texto) a una red, que, en primer lugar, llevará a cabo un proceso de embedding que permita trabajar con una representación de dichos datos, y, posteriormente, entrenará con una serie de capas.

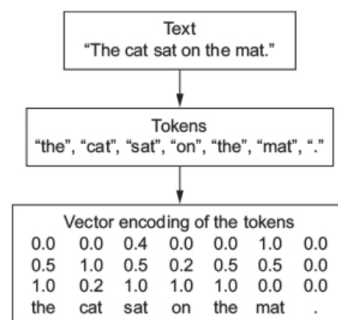


Figura 3.1: De texto a tokens a vectores

3.2 Word Embedding

La primera de las técnicas a usar va a ser *Word Embedding*. Que es otra forma comúnmente de asociar un vector con una palabra, mediante el uso de vectores de palabras densos, también llamados *word embedding* (ver figura 3.4).

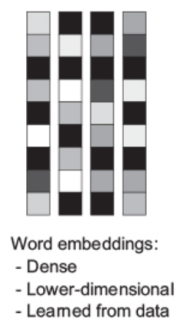


Figura 3.2: *Word embedding*

Para la obtención de estos vectores, existen dos maneras principales de obtenerlos:

1. Aprender *word embedding* conjuntamente con la clasificación de documentos o la predicción de sentimientos. Para esta configuración, se comienza con vectores de palabras al azar y luego aprendes vectores de palabras de la misma manera que aprendes los pesos de una red neuronal.
2. Cargar un modelo de *word embedding* pre-entrenado para una tarea distinta a la que se está tratando de resolver. Esta configuración se llama *pretrained word embeddings*.

Debido a la complejidad que conlleva realizar un *word embedding* desde cero, se ha optado por hacer uso de la segunda vertiente. Además, de que a veces se tiene tan poca información de entrenamiento disponible que no se pueden usar sólo los datos de uno mismo para aprender un *word embedding* adecuado. Es por eso, que en lugar de aprender *word embedding* conjuntamente con nuestro problema, podemos cargar *word embedding* pre-entrenados, los cuales están altamente estructurado y tienen propiedades útiles, que suelen capturar aspectos genéricos de la estructura del lenguaje.

La razón detrás del uso de *pretrained word embedding* en el procesamiento en lenguaje natural es muy parecida a la de usar *convnets* pre-entrenados en la clasificación de imágenes: no se tienen suficientes datos disponibles propios para aprender características realmente poderosas, pero se espera que las características sean necesarias. En este caso, tiene sentido reutilizar las características aprendidas en un problema diferente.

Hay varias bases de datos que tienen *pretrained word embedding*, las cuales se pueden descargar y usar en una capa de *embedding* en Keras. Entre ellas, destacamos *Word2vec* y *GloVe*. Debido a diversas recomendaciones se va a hacer uso de *Global Vectors for Word Representation* (GloVe, <https://nlp.stanford.edu/projects/glove>).

3.2.1 Configuración del modelo

A continuación, se exponen los pasos necesarios para la creación de una red neuronal con *pretrained word embedding*. De todas maneras, en un paso previo, se ha seleccionado el conjunto training y particionado en dos conjuntos uno para train y otro para test. Además, al principio se pensó en realizar el mismo particionamiento que habíamos hecho con las imágenes para futuros estudios, sin embargo, debido a la incoherencia de dimensiones y de disponer en las imágenes de 4 fotografías por mascota y en el CSV de solo una fila para dicha mascota, se ha obviado hacer uso de esta opción.

1. Lo primero de todo, es **obtener los textos y las etiquetas** de nuestro conjunto training.

```
1 labels_train <- train$AdoptionSpeed
2 texts_train <- train$Description
```

2. Seguidamente, como hemos comentado previamente es necesario **tokenizar el texto**, es decir, obtener textos estructurados, requisito indispensable para la red. Para ello se sigue la siguiente estructura:
 - a) Cortar revisiones después de 100 palabras
 - b) Obtener del train 900 muestras
 - c) Validar con 10000 muestras

- d) Considerar solo las 10000 palabras principales o más importantes en el conjunto de datos
- e) Dividir los datos en un conjunto de entrenamiento y en un conjunto de validación.

```

1 maxlen <- 100
2 training_samples <- 900
3 validation_samples <- 10000
4 max_words <- 10000
5
6 tokenizer <- text_tokenizer(num_words = max_words) %>%
7   fit_text_tokenizer(texts_train)
8 sequences <- texts_to_sequences(tokenizer, texts_train)
9 word_index = tokenizer$word_index
10
11 data <- pad_sequences(sequences, maxlen = maxlen)
12 labels <- as.array(labels_train)
13
14 indices <- sample(1:nrow(data))
15 training_indices <- indices[1:training_samples]
16 validation_indices <- indices[(training_samples + 1):
17   (training_samples + validation_samples)]
18
19 x_train <- data[training_indices,]
20 y_train <- labels[training_indices]
21 x_val <- data[validation_indices,]
22 y_val <- labels[validation_indices]

```

3. Una vez que ya tenemos nuestros dos conjuntos de datos estructurados de forma adecuada, pasamos a la descarga del modelo en <https://nlp.stanford.edu/projects/glove> para **Glove**. Dicho archivo contiene las *pretrained word embedding* de Wikipedia en inglés de 2014. Contiene vectores de integración de 100 dimensiones para 400,000 palabras (o tokens sin palabras). Consecutivamente, es necesario preprocesar los *embeddings*, para crear un índice que asigne palabras (como cadenas) a su representación vectorial (como vectores numéricos).
4. Lo siguiente es **definir el modelo**, para la definición de dicho modelo, se ha usado la misma estructura que se comentó en la práctica 2, sin embargo, en este caso se debe de introducir como capa principal la capa de *embedding*.

```

1 model <- keras_model_sequential() %>%
2   layer_embedding(input_dim = max_words,
3     output_dim = embedding_dim, input_length = maxlen) %>%
4   layer_dropout(0.3) %>%
5   layer_flatten() %>%
6   layer_dense(units = 512, activation = "relu") %>%
7   layer_dense(units = 256, activation = "relu") %>%
8   layer_dense(units = 5, activation = "sigmoid")

```

Se debe destacar que la capa de *embedding* tiene una sola matriz de ponderación, es decir, una matriz flotante 2D donde cada entrada i es el vector de palabra destinado a asociarse con el índice i . Y debe ser cargada en la primera capa del modelo.

5. De la misma manera que se hizo en la Práctica 2, es necesario **congelar los pesos** de la capa de *embedding*, siguiendo el mismo razonamiento utilizado para las redes neuronales convolucionales pre-entrenadas: cuando las partes de un modelo están pre-entrenadas (como su capa de incrustación) y las partes se inicializan al azar (como su clasificador), las partes pre-entrenadas no deben actualizarse durante el entrenamiento, para evitar olvidar lo que ya

saben.

```
1 get_layer(model, index = 1) %>%
2   set_weights(list(embedding_matrix)) %>% freeze_weights()
```

6. Por tanto, ya solo nos queda entrenar y evaluar el modelo. Para el **entrenamiento**, se ha vuelto hacer uso del algoritmo de optimización ADAM y de las medidas loss y accuracy, igual que en la Práctica 2. Se han usado 10 épocas para entrenar y tamaño de lotes de 100.

```
1 # Compilamos
2 model %>% compile(
3   optimizer = "adam", # Mejora mas 'adam' que con rmsprop
4   loss = "sparse_categorical_crossentropy",
5   metrics = c("accuracy")
6 )
7
8 # Entrenamos
9 history <- model %>%
10   fit(x_train, y_train,
11     epochs = 10,
12     batch_size = 100,
13     validation_split = 0.2,
14     validation_data = list(x_val, y_val)
15 )
```

7. Una vez que ya tenemos nuestro modelo solo hace falta, **evaluar** dicho modelo sobre los datos del conjunto test, para ello es necesario que previamente dichos datos textuales se conviertan en token, tal y como vimos para los datos training.

```
1 # Tokenizar el conjunto de datos test
2 labels_test <- train$AdoptionSpeed
3 texts_test <- train$Description
4
5 sequences <- texts_to_sequences(tokenizer, texts_test)
6 x_test <- pad_sequences(sequences, maxlen = maxlen)
7 y_test <- as.array(labels_test)
8
9 # Evaluar el modelo
10 model %>% evaluate(x_test, y_test)
```

Salida para el modelo Word Embedding

Tiempo de Entrenamiento: 62 segundos
Train Loss: 0.0559
Train Accuracy: 0.9789
Validation Loss: 3.3735
Validation Accuracy: 0.2647
Test Loss: 3.134192
Test Accuracy: 0.3172155

A partir de los resultados obtenidos en la anterior salida, parece apreciarse un mejor comportamiento de los datos textuales que de las imágenes, si tenemos en cuenta el *accuracy* para dichos datos.

Además, la no linealidad de la red, así como la capacidad de integrar fácilmente *embedding* de palabras pre-entrenadas, puede conducir a una precisión de clasificación superior. Por otro lado, si miramos la componente respecto el tiempo de entrenamiento, se ve una reducción considerable si lo comparamos con las imágenes. Por lo que esto nos lleva a la conclusión de que debemos usar las descripciones asociadas a cada mascota si queremos predecir el tiempo que tardan en adoptar.

En los siguientes apartados se va a seguir haciendo uso de modelos pretrained word embedding más la incorporación de otras diversas capas.

3.3 Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN), son una clase de redes neuronales artificiales de avance profundo en las que las conexiones entre nodos no forman un ciclo. Se utilizan generalmente en la visión por computadora, sin embargo, también han mostrado resultados prometedores cuando se aplican a varias tareas de procesamiento de lenguaje natural.

Para las imágenes era necesario hacer uso de convoluciones en 2D, las cuales extraían parches 2D de los tensores de imagen y aplicaban una transformación idéntica a cada parche. De la misma manera, se pueden usar las convoluciones 1D, extrayendo parches 1D locales (subsecuencias) de secuencias.

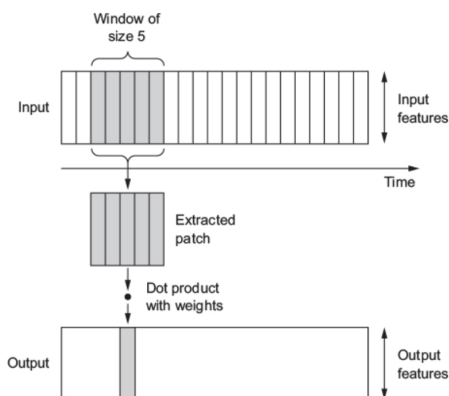


Figura 3.3: Funcionamiento CNN 1D

Estas capas de convolución 1D pueden reconocer patrones locales en una secuencia. Debido a que la misma transformación de entrada se realiza en cada parche y un patrón aprendido en una determinada posición en una oración puede reconocerse posteriormente en una posición diferente. Por ejemplo, una secuencia de caracteres de procesamiento de convnet 1D que utiliza ventanas de convolución de tamaño 5 debería poder aprender palabras o fragmentos de palabras de longitud 5 o menos, y debería poder reconocer estas palabras en cualquier contexto en una secuencia de entrada. Por lo tanto, un convnet 1D de nivel de caracteres puede aprender sobre morfología de palabras.

La arquitectura de las CNN se compone de tres piezas clave:

1. **Incorporación de palabras:** una representación distribuida de palabras donde diferentes palabras que tienen un significado similar también tienen una representación similar.

2. **Modelo convolucional:** un modelo de extracción de características que aprende a extraer características de documentos usando *word embedding*.
3. **Modelo totalmente conectado:** la interpretación de las características extraídas en términos de un resultado predictivos.

En Keras, para usar convnet 1D, es necesario hacer uso de la función `layer_conv_1d`, que tiene una interfaz similar a `layer_conv_2d`.

1. Capa de *embedding*
2. Capa de convolución 1D con 64 filtros y kernel de tamaño (7x7)
3. Capa de activación con función reLu"
4. Capa MaxPooling 1D con `pool_size=5x5`
5. Capa Dropout con 0.3
6. Capa flatten
7. Capa densa con 64 neuronas.
8. Capa de activación con función reLu"
9. Capa densa con 64 neuronas.
10. Capa de activación con función reLu"
11. Capa de salida con función de activación *sigmoid*.

```

1 model <- keras_model_sequential() %>%
2   layer_embedding(input_dim = max_words,
3     output_dim = embedding_dim, input_length = maxlen) %>%
4   layer_conv_1d(filters = 64, kernel_size = 7, activation = "relu") %>%
5   layer_max_pooling_1d(pool_size = 5) %>%
6   layer_dropout(0.3) %>%
7   layer_flatten() %>%
8   layer_dense(units = 64, activation = "relu") %>%
9   layer_dense(units = 64, activation = "relu") %>%
10  layer_dense(units = 5, activation = "sigmoid")

```

Salida para el modelo CNN

Tiempo de Entrenamiento: 62 segundos
Train Loss: 0.9990
Train Accuracy: 0.5700
Validation Loss: 1.6617
Validation Accuracy: 0.2835
Test Loss: 1.656911
Test Accuracy: 0.2845352

Cómo se aprecia dichas redes son más rápidas en tiempo de ejecución. Ya que las convoluciones son una parte central de los gráficos de computadora y se implementan a nivel de hardware en las GPU. Sin embargo, se obtiene peores resultados que con el uso de *word embedding* sólo.

Por otro lado, en la literatura se comenta que dichas redes 1D pueden ser competitivas con las redes

neuronales recurrentes² en ciertos problemas de procesamiento de secuencias, generalmente a un costo computacional considerablemente más bajo.

3.4 Redes neuronales recurrentes

Una característica importante de todas las redes neuronales es que no tienen memoria, ya que cada entrada que se les muestra se procesa de forma independiente, sin que se mantenga ningún estado entre las entradas. Si con dichas redes queremos procesar una secuencia o una serie temporal de puntos de datos, se debe mostrar la secuencia completa a la red a la vez, es decir, convertirla en un solo vector de datos.

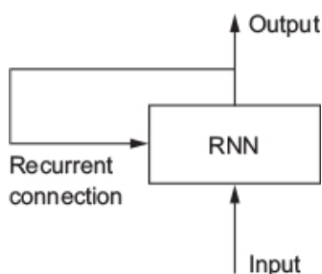


Figura 3.4: Funcionamiento RNN

Una **red neuronal recurrente** (RNN) adopta este mismo principio: procesa secuencias al iterar a través de los elementos de la secuencia y mantiene un estado que contiene información relativa a lo que se ha visto hasta ahora. Por tanto, es una clase de red neuronal donde las conexiones entre nodos forman un grafo dirigido a lo largo de una secuencia, es decir, cada una está pasando un mensaje a un sucesor. Esta arquitectura permite que las RNN muestren un comportamiento temporal y capture datos secuenciales, lo que lo convierte en un enfoque más *natural* cuando se trata de datos textuales, ya que el texto es naturalmente secuencial. Puesto que nos gustaría aprender que ciertas secuencias de palabras son buenos indicadores del tema, y no necesariamente nos importa dónde aparecen en el documento.

Esta manera de procesar la información, está más relacionada con la forma en como los humanos procesan la información de manera incremental, ya que mantiene un estado interno de lo que está procesando. Y se construye un contexto a partir de información pasada y se actualizan constantemente a medida que entra la nueva información.

Por otra parte, se debe destacar que una RNN está capacitada para reconocer patrones a lo largo del tiempo, mientras que una CNN aprende a reconocer patrones en el espacio. Es por eso que las RNN suelen ser buenas para predecir lo que sigue en una secuencia, mientras que las CNN suelen ser buenas para aprender a clasificar una oración o un párrafo.

²Se verá en el siguiente apartado

Dentro de Keras, existen distintas funciones que usan una RNN, debido a su mejora y como recomendación se destaca usar `layer_lstm(...)`. Básicamente lo que hace LSTM es guardar información para más tarde, evitando así que los datos más antiguos se desvanezcan gradualmente durante el proceso. Para la creación del modelo, se debe de hacer uso del mismo proceso explicado previamente, por lo que solo será necesario añadir a nuestro modelo una capa LSTM.

1. Capa de *embedding*
2. Capa LSTM con 32 filtros
3. Capa flatten
4. Capa densa con 512 neuronas.
5. Capa de activación con función reLu"
6. Capa densa con 256 neuronas.
7. Capa de activación con función reLu"
8. Capa de salida con función de activación *sigmoid*.

```

1 model <- keras_model_sequential() %>%
2   layer_embedding(input_dim = max_words,
3                   output_dim = embedding_dim, input_length = maxlen) %>%
4   layer_lstm(units = 32) %>% # Funciona mejor la 32 que la 64
5   layer_flatten() %>%
6   layer_dense(units = 512, activation = "relu") %>%
7   layer_dense(units = 256, activation = "relu") %>%
8   layer_dense(units = 5, activation = "sigmoid")

```

Salida para el Modelo RNN

Tiempo Ejecución: 56 segundos

Train Test: 1.599465

Train Test: 0.3134639

Como se aprecia se obtiene un resultado bastante bueno, ya que como hemos comentado un LSTM procesa secuencias al iterar a través de los elementos de la secuencia y mantiene un estado que contiene información relativa a lo que se ha visto hasta ahora. Por lo que ese estado que mantiene, permite tener una mejor comprensión de los textos. Además, gracias a su comportamiento temporal y captura de datos secuenciales, lo que lo convierte en un enfoque más *natural* cuando se trata de datos textuales. Por lo que se considera, que debe de obtener un mejor resultado que hacer uso de la aplicación de CNN.

3.5 Redes neuronales recurrentes + CNN

Viendo el buen comportamiento que tienen tanto las RNN como las CNN, se decide hacer uso de ambas de forma combinada. Debido a que la RNN está capacitada para reconocer patrones a lo largo del tiempo, mientras que una CNN aprende a reconocer patrones en el espacio. Por lo que las RNN se centrarán más predecir lo que sigue en una secuencia, mientras que las CNN se basarán en aprender a clasificar una oración o un párrafo.

Se observa como ambas extraen características que la otra no tiene en cuenta, por lo que una posible combinación de ambas puede dar lugar a un modelo adecuado.

1. Capa de *embedding*
2. Capa de convolución 1D con 64 filtros y kernel de tamaño (7x7)
3. Capa de activación con función reLu"
4. Capa MaxPooling 1D con pool_size=5x5
5. Capa LSTM con 32 filtros
6. Capa Dropout 0.3
7. Capa densa con 256 neuronas.
8. Capa de activación con función reLu"
9. Capa densa con 128 neuronas.
10. Capa de activación con función reLu"
11. Capa de salida con función de activación *sigmoid*.

```

1 model <- keras_model_sequential() %>%
2   layer_embedding(input_dim = max_words,
3     output_dim = embedding_dim, input_length = maxlen) %>%
4   layer_conv_1d(filters = 64, kernel_size = 7, activation = "relu") %>%
5   layer_max_pooling_1d(pool_size = 5) %>%
6   layer_lstm(units = 32) %>%
7   layer_dropout(0.3) %>%
8   layer_dense(units = 256, activation = "relu") %>%
9   layer_dense(units = 128, activation = "relu") %>%
10  layer_dense(units = 5, activation = "sigmoid")

```

Salida para el Modelo CNN con RNN

Tiempo de Entrenamiento: 62 segundos

Train Loss: 1.4795

Train Accuracy: 0.3044

Validation Loss: 1.4777

Validation Accuracy: 0.2832

Test Loss: 1.478439

Test Accuracy: 0.2848687

Respecto los resultados obtenidos en dicho modelo, podemos destacar que CNN se centra más clasificar una oración y RNN se centra más en predecir lo que sigue a una secuencia. Sin embargo, no es de extrañar que el resultado obtenido empeore, una de las causas puede ser debido a la componente aleatoria que hemos ido comentando a lo largo de estos trabajos. Adicionalmente, se debe tener en cuenta que la combinación de capas que de forma independiente funcionan mejor, no debe de suponer una mejora cuando se combinan. Ya que se debe de considerar que la aplicación de modelos más simples a nuestros datos suele dar resultados mejores.

3.6 Análisis de los resultados

Después de realizar varios experimentos, hemos llegado a la conclusión de que bajar el error del rango (1.4-1.6) es difícil. Además, podemos comprobar que los resultados obtenidos con los datos textuales son parecidos a los obtenidos con las imágenes, cosa que nos lleva a pensar que ambos aportan información importante. No obstante, se obtienen mejoras en cuanto a tiempo de entrenamiento estamos hablando, ya que computacionalmente es menos costoso de procesar un texto que una imagen.

MODELO	TEST LOSS	TEST ACCURACY
Word embedding	1.47964	0.2759242
Word embedding + CNN	1.656911	0.2845352
Word embedding + RNN	1.599465	0.3134639
Word embedding + CNN + RNN	1.478439	0.2848687

Cuadro 3.1: Resumen de los modelos con Word Embedding para el conjunto test

Por otro lado, no es de extrañar que el modelo simple con **Word embedding y LSTM**³ obtenga tan buenos resultados. Ya que una capa hace uso de información pre-entrenada y la otra mantiene un estado que contiene información relativa a lo que se ha visto hasta ahora. Esto permite que ambos enfoques sean más *naturales* cuando se trata de datos textuales.

³Cuando hacemos referencia a cada uno de estos métodos, estamos hablando de redes neuronales, que incluyen dichas técnicas de manera previa a aplicar capas densas, que derivan finalmente en las 5 etiquetas.

4. Modelos Ensemble

4.1 Redes Neuronales Convolucionales

4.1.1 Visualización en convnets

En la práctica 2, hemos hablado de las ventajas y grandes facilidades que nos aportan las Redes Neuronales Convolucionales, principalmente en cuanto al tratamiento de imágenes para así poder extraer sus características más importantes.

Una de las características de las Redes Neuronales, más conocidas y comentadas sobretodo en ámbitos de Inteligencia Artificial, es la dificultad añadida que traen consigo, ya que, a ojos de alguien que haga uso de ellas, son como una especie de caja negra, donde, introduces unos datos de entrada, y, sin saber muy bien lo que ocurre internamente, obtienes un resultado a partir de las mismas. Por ello, es curioso ver, que, con las redes Neuronales, sí que podemos extraer conocimiento interno de dicha red.



Figura 4.1: Ejemplo de Visualización con Convnets

Por ejemplo, podemos ver, los distintos filtros que se aplican en las capas de convolución, a través de la visualización de las activaciones intermedias de una red. De esta manera, dada una determinada entrada, podremos descomponer los filtros que aprende la red neuronal, de la misma forma que ocurre en la imagen 4.1, donde se puede observar, uno de los canales de filtros aplicado en las primeras capas para la imagen *0fd68ca16-1.jpg*. Cabe decir, que, como se observa, principalmente se visualizan, en distintos tonos, los distintos cambios de gradiente en las imágenes, que son lo que, al fin y al cabo, caracteriza a una imagen y permitirá extraer características que la distinguan.

Además, ya que estamos visualizando lo que se observa en estas capas convolucionales, en la imagen 4.2 se puede observar todos los canales para una de las primeras capas convolutivas en la red. En la imagen 4.3 repetimos el proceso para una capa intermedia, y en la imagen 4.4 para una de las últimas. Podemos contrastar que, tal y como se ha estado comentando en la práctica 2, las primeras capas nos permiten obtener información más genérica (podemos distinguir la figura del gato), mientras que en las últimas, ya hablamos de puntos muy específicos y concretos como para visualmente poder interpretar nosotros información.

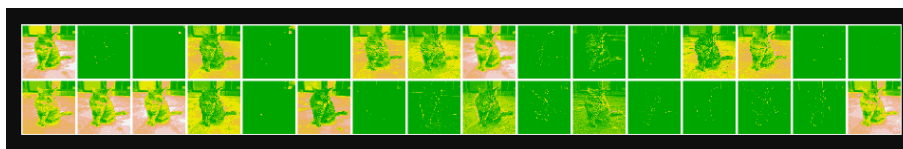


Figura 4.2: Visualización en Convnets I

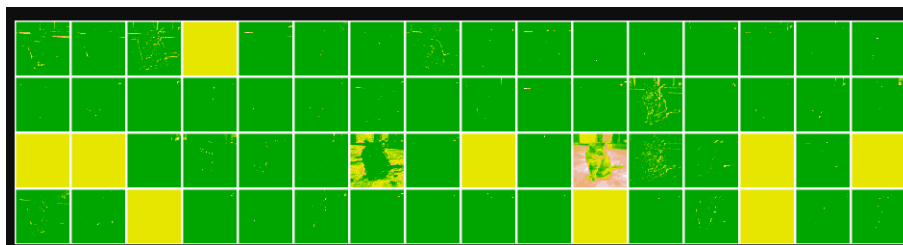


Figura 4.3: Visualización en Convnets II

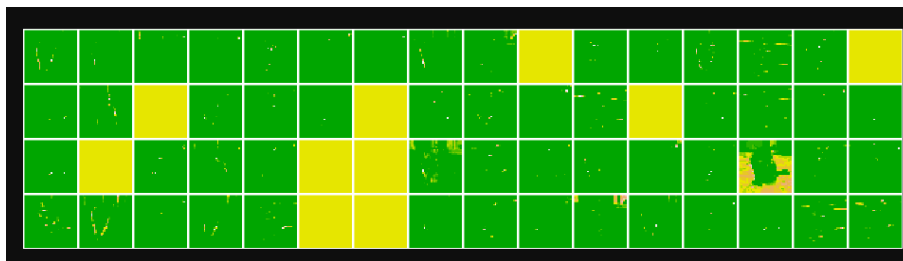


Figura 4.4: Visualización en Convnets III

Además, podemos también visualizar con un mapa de calor, aquellas zonas de la imagen las cuáles nos permiten apreciar de manera visual, cuáles son las zonas en las que se está fijando la red entrenada en nuestras imágenes. Este es el caso que se puede ver en la imagen 4.5.



Figura 4.5: Visualización con HeatMap I

4.1.2 Ensemble con Convnets

Del apartado anterior, hemos visto que podemos visualizar sobre una imagen, aquellas partes en las que se fija o visualiza más una red, en función de la activación que trae intrínseca como resultado del entrenamiento. Es por tanto que podremos ver, para una misma imagen, cómo se comportan distintos modelos sobre la misma.

Ahora imaginemos que queremos realizar un modelo, que es resultado de una serie de modelos que trabajan de forma distinta. Asimismo, hemos podido visualizar, cómo se comporta una red sobre una imagen. Parece obvio que podamos combinar, estos dos aspectos con el fin de dar lugar técnicas potentes. Véase el ejemplo que se puede consultar en la bibliografía principal de este trabajo. Imaginemos que hay tres personas ciegas, que intentan identificar un elefante. Una puede tocar la oreja, otra una pata, y otra alguna parte distinta de las dos anteriores. En equipo, podrían, gracias a la información que facilitan todos, dar con la predicción correcta.

Ese mismo caso, es el que pretendemos simular en este apartado. Gracias a la visualización anterior, podemos intentar dar con modelos que trabajen de forma contraria, y o bien que se fijen en partes distintas de la foto, y combinarlos, para intentar dar lugar a un modelo poderoso. Este es el principio básico a la hora de realizar un ensemble, ya que, si se combinan varios modelos que predicen de forma parecida, no conseguiremos tal efecto que si, por otra parte, intentamos combinar modelos distintos que tienen en cuenta distintos aspectos para poder realizar la predicción.

En nuestro caso, vamos a utilizar tres modelos, que se comportan de manera diferente para los problemas que tenemos. No solo tienen estructuras diferentes, sino que esto, ha sido comprobado también observando qué partes se activan al visualizarlo sobre distintas imágenes del conjunto.

Para ello, hemos cogido distintos modelos convolucionales, creados por nosotras, los cuáles hemos comprobado que generaban resultados diferenciados. Podemos ver un ejemplo de esta diferenciación en el conjunto de imágenes 4.6, donde, se puede ver que mientras que lo que se activa en uno de esos modelos (llamémoslo Modelo I) en contraste a lo que se visualiza en el Modelo II, es distinto.

De igual forma ocurre con el caso del conjunto de imágenes 4.7.



Figura 4.6: Visualización con HeatMap II



Figura 4.7: Visualización con HeatMap III

La estructura utilizada para estos modelos, ha sido la siguiente:

Modelo I

1. Capa de convolución 2D con 32 filtros y kernel de tamaño (11x11)
2. Capa de activación con función reLu"
3. Capa MaxPooling 2D con pool_size=2x2
4. Capa de convolución 2D con 64 filtros y kernel de tamaño (7x7)
5. Capa de activación con función reLu"
6. Capa de convolución 2D con 64 filtros y kernel de tamaño (5x5)
7. Capa de activación con función reLu"
8. Capa MaxPooling 2D con pool_size=2x2
9. Capa flatten
10. Capa densa con 512 neuronas.
11. Capa de activación con función reLu".
12. Capa de salida con función de activación *softmax*.

Modelo II

1. Capa de convolución 2D con 64 filtros y kernel de tamaño (11x11)
 2. Capa de activación con función reLu"
 3. Capa MaxPooling 2D con pool_size=2x2
 4. Capa de convolución 2D con 64 filtros y kernel de tamaño (9x9)
 5. Capa de activación con función reLu"
 6. Capa de convolución 2D con 32 filtros y kernel de tamaño (5x5)
 7. Capa de activación con función reLu"
 8. Capa MaxPooling 2D con pool_size=2x2
 9. Capa flatten
 10. Capa densa con 512 neuronas.
 11. Capa de activación con función reLu".
 12. Capa de salida con función de activación *softmax*.
-

Por último, a estos modelos, se le añadió un modelo más complejo, que permitiera mejorar los resultados y a cuyas predicciones le pudiéramos dar un mayor peso (y a su vez, más representativo). Por ello, se eligió como tercer modelo el que proporciona la mejor configuración en la práctica 2, y se puede ver su estructura a continuación:

Modelo III

1. Output de la Red VGG16 (se congelan las 13 primeras capas).
 2. Capa de media global de pooling 2D.
 3. Capa densa con 512 neuronas.
 4. Capa de activación con función reLu".
 5. Capa Dropout con un valor de 0.2
 6. Capa densa con 256 neuronas.
 7. Capa de activación con función reLu".
 8. Capa de salida con función de activación *softmax*.
-

4.1.3 Weight Average Ensemble con Convnets

Se ha llevado a cabo un ensemble sencillo, conocido como **Weight Average Ensemble**, que nos permite ponderar, en función de lo que nos interese o con un algoritmo de optimización, las predicciones que nos proporciona cada uno de los modelos por separado.

De esta manera, se intenta dar más importancia a aquellos modelos que pensamos que puedan dar mejores resultados, pero también teniendo en consideración la predicción de modelos más distintos. Como las predicciones que obtenemos a partir de los distintos modelos, son probabilidades, hemos ponderado el valor de dicha probabilidad en función de lo que nos interesaba. Posteriormente, hemos asignado, a cada muestra, la etiqueta cuyo valor era el más alto, y hemos calculado el accuracy correspondiente a nuestro procedimiento (de la misma manera que se calcularía internamente en un algoritmo). Se puede ver el código a continuación.

```
1 y_modelo1 <- model1 %>% predict_generator(test_data, steps=5)
2 y_modelo2 <- model2 %>% predict_generator(test_data, steps=5)
```

```
3 y_modelo4 <- model4 %>% predict_generator(test_data, steps=5)
4
5 etiquetas_predichas <- apply(final_preds, 1, which.max)
6 etiquetas_predichas <- etiquetas_predichas - 1
7 accuracy = sum(test_data$labels == etiquetas_predichas)
8             / length(test_data$labels)
```

El valor de Accuracy obtenido con esta combinación ha sido de 0.29 (un poco menor que el máximo conseguido). Sin embargo, se debe tener en cuenta que un ensemble es un método más robusto, pero si los modelos que lo forman no dan buenos resultados, éste tampoco lo hará, pues depende directamente de cada uno de ellos.

Sin embargo, puestos a decidir, es un modelo mucho más fiable. Esto lo hemos podido comprobar gracias a las matrices de confusión de los modelos, ya que, realizando una comparación exhaustiva de nuestros resultados en Test, hemos observado distintos aspectos:

- Teníamos modelos muy malos, que no llegaban a predecir siquiera en algunas de las etiquetas.
- Hemos visto otros, que se equivocaban más que acertaban en la etiqueta en cuestión (recorremos que Accuracy no es una medida que tenga en cuenta cuánto nos hemos equivocado, ni si ha sido en diferencia de 1 o de más).
- Viendo el modelo ensemble, en definitiva, veíamos cómo los resultados en la matriz de confusión se iban viendo representados por todos los modelos, y que incluso el porcentaje de elementos que se equivocaban en clases cercanas, era mayor.

Por ello, aunque no se hayan obtenido mejores resultados, ya que, además de la deficiencia de nuestros modelos, se ha llevado a cabo un modelo muy básico, hemos podido comprobar que usarlos para obtener mejores resultados, y sobretodo, más fiables, es una opción a considerar, y además, de con las que mejores resultados vamos a obtener.

4.2 Ensemble con Procesamiento de Textos

4.2.1 Weight Average Ensemble con datos del CSV

Asimismo, el mismo procedimiento de *Weight Average Ensemble con Convnets* de la sección anterior, se ha llevado a cabo con los modelos generados con los ficheros del CSV. En concreto, se ha llevado a cabo un ensemble con dos modelos (uno de ellos con LSTM y otro uno sencillo con capas densas). Sin embargo, además de no haber dado buenos resultados, no hemos visto conveniente juntarlos con los del apartado anterior, por dos razones:

- El número de datos del CSV es mucho inferior que el de las imágenes, y requería de un procesamiento que, o nos dejaba con muchas muestras con información muy repetida, o pocas muestras (al reducir al tamaño del csv).
- Realmente, esta técnica no es potente, sino que es muy sencilla, y vimos preferible avanzar en algo más complejo y que pudiera darnos mayor potencia y veracidad de los resultados.

4.2.2 Modelos RRNN de entrada y salida múltiple

Hasta este punto, hemos visto la implementación de modelos ensemble muy simples, basados simplemente en realizar una ponderación, en función de la importancia que nosotros queramos darle,

a las distintas predicciones realizadas por los algoritmos. Como se ha comentado, esta técnica es muy poco potente, ya que no solo se ve influenciada por los malos resultados, sino que es demasiado básica como para poder aprovechar bien toda la información que tenemos. Por ejemplo, con este ensemble, no tendríamos forma de combinar, a nivel de entrenamiento, los datos de las imágenes y de los textos juntos. Es decir, para poder disponer de esta información con un *Weight Average Ensemble*, los entrenaríamos de manera separada.

En otras palabras, el objetivo final con este apartado es conseguir, tener como entrada de una red, conjuntos de datos de distinta procedencia: imágenes, texto y características categóricas y numéricas del animal.

Además, con el objetivo de optimizar los resultados, se ha implementado un modelo ensemble con el fin de mejorar dichos resultados mediante la combinación de varios modelos, en este caso, queremos hacer uso de todos los datos aportados en el CSV, sin embargo, es necesario hacer uso indistintamente de los datos numéricos del CSV por un lado, y de los datos textuales por otro, ya que ambos modelos reciben tratamiento de datos de forma diferente, por lo que para los textos seguiremos manteniendo la tokenización de los mismos.

Ensemble datos textuales + datos numéricos

Entonces, vamos aplicar un ensemble mediante la combinación de los datos textuales que vimos en el capítulo 3 junto con los otros datos que nos aporta el CSV como pueden ser la edad, el color del pelo, si está vacunado, la salud que tiene, entre otros. Estos datos pueden ser de vital relevancia en cuanto a la adopción de la mascota (ver figura 4.8).

La entrada principal al modelo será la descripción en sí de la mascota, es decir, como una secuencia de palabras, pero para darle un toque más especial, nuestro modelo también tendrá una entrada auxiliar, que recibirá los restantes datos aportados en el CSV que no sean textos. **En otras palabras, estamos concatenando texto con metadatos y poder entrenarlo todo de manera simultánea.**

La entrada principal recibirá la descripción, como una secuencia de enteros (cada entero codifica una palabra), es decir, se deberá volver a realizar una tokenización del texto como hicimos anteriormente. Y debido a que la técnica para *LSTM* fue la que mejor resultados obtuvo, se decide hacer uso de ella. **Es importante tener en cuenta que, para poder añadir la información de los metadatos del CSV, y concatenarla a nuestros resultados con los textos, no podemos unirlos al mismo nivel, sino que la unión debe realizarse una vez tengamos procesado el texto de la manera adecuada para textos (técnicas de Embedding y LSTM).**

```
1 main_input <- layer_input(shape = c(100), dtype = 'int32',
2                             name = 'main_input')
3 lstm_out <- main_input %>%
4   layer_embedding(input_dim = 10000, output_dim = 512,
5                   input_length = 100) %>%
6   layer_lstm(units = 32)
```

Seguidamente insertaremos la pérdida auxiliar, lo que permite que el LSTM y la capa de incrustación se entrenen sin problemas aunque la pérdida principal sea mucho mayor en el modelo.

```
1 auxiliary_output <- lstm_out %>%
```

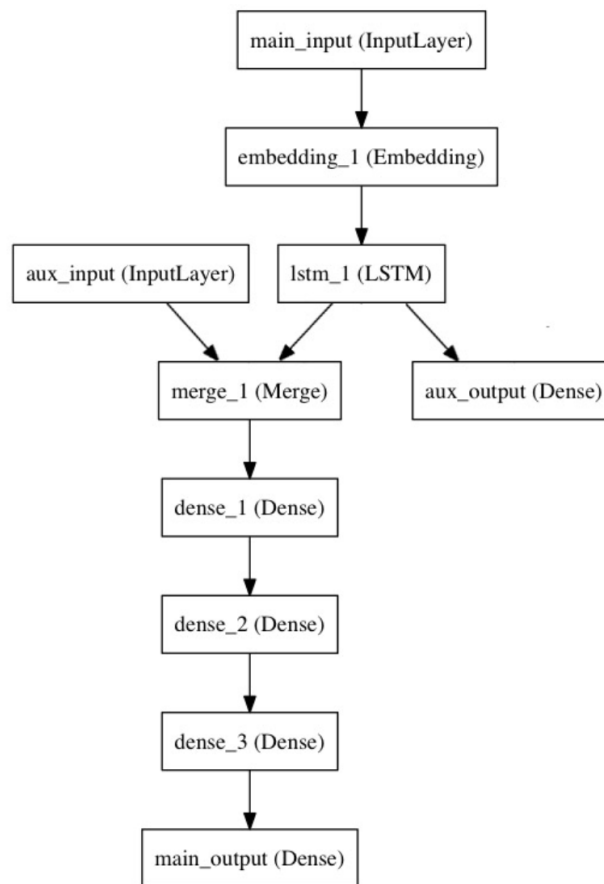


Figura 4.8: Modelo a crear para el ensemble datos textuales + datos numéricos

```
2 layer_dense(units = 5, activation = 'sigmoid', name = 'aux_output')
```

A continuación, insertamos en el modelos nuestros datos de entrada auxiliares (datos procedentes del resto del CSV), al concatenarlos con la salida LSTM, es necesario crear una red con una gran densidad de conexión en la parte superior y agregar una capa de concatenación de ambas.

```
1 auxiliary_input <- layer_input(shape = c(19), name = 'aux_input')
2 main_output <- layer_concatenate(c(lstm_out, auxiliary_input)) %>%
3   layer_dense(units = 64, activation = 'relu') %>%
4   layer_dense(units = 64, activation = 'relu') %>%
5   layer_dense(units = 64, activation = 'relu') %>%
6   layer_dense(units = 5, activation = 'sigmoid', name = 'main_output')
```

Por último, ya solo nos queda compilar, entrenar y evaluar el modelo. Para dicha evaluación, es indispensable que los datos textuales del conjunto test estén tokenizados.

```
1 # Definir el modelo
2 model <- keras_model(
3   inputs = c(main_input, auxiliary_input),
4   outputs = c(main_output, auxiliary_output)
5 )
6
```



```

7  # Compilar el modelo
8  model %>% compile(
9      optimizer = 'adam',
10     loss = 'sparse_categorical_crossentropy',
11     metrics = c("accuracy")
12 )
13
14 # Entrenar el modelo
15 model %>% fit(
16     x = list(x_train_nuevo, as.matrix(train.normal)),
17     y = list(train$AdoptionSpeed, train$AdoptionSpeed),
18     epochs = 2,
19     batch_size = 64
20 )
21
22 # Evaluar
23 model %>% evaluate(list(x_test_nuevo, as.matrix(test.normal)),
24                   list(y_test_nuevo, y_test_nuevo))

```

Salida el ensemble con datos textuales y los restantes del CSV

Tiempo de Entrenamiento: 116 segundos
Train Loss: 10.4170
Train Accuracy: 0.2792
Validation Loss: 8.9511
Validation Accuracy: 0.2686
Test Loss: 8.951117
Test Accuracy: 0.278519

Como se aprecia, la obtención de este ensemble no ha obtenido buenos resultados, esto puede ser debido a que hay datos en el CSV numéricos que penalicen el modelo, al no aportar información alguna. Por lo que no se ha seguido explorando más allá, al no aportar conclusiones claras. Y además, de que muchos de los datos numéricos del CSV no aportan contenido fundamental. **Aquí sería recomendable haber realizado un estudio de correlaciones para ver que variables son realmente significativas**, ya que de por sí, Kaggle tampoco proporciona información de lo que significa cada una de las variables (e incluso haber aplicado técnicas de preprocesamiento como escalado y PCA). Para la implementación de este ensemble se ha hecho uso del siguiente enlace.

Asimismo, la continuación y mejora de esta técnica es natural: añadir otra entrada auxiliar de la red neuronal la extracción de características de nuestros modelos de redes convolucionales que tratan con imágenes. El objetivo sería que, a la vez que añadimos a nuestra red el resto de datos del .csv (como se puede ver en la imagen anterior donde se visualiza la estructura de la red), se añadiese también una matriz, con los vectores de características de las imágenes de cada una de dichas muestras. Posteriormente, realizar el entrenamiento con todo conjunto. **De esta manera, podríamos realizar un entrenamiento que tuviera en cuenta tanto las imágenes, como los datos del csv de manera simultánea.**

Esto último, no ha sido implementado puesto que se salía del área de nuestro trabajo de investigación, pero el procedimiento consistiría en añadir, a nuestra red neuronal de este apartado lo siguiente:

- Realizar un muestreo dentro del conjunto de nuestro problema, para poder tratar con las mismas imágenes que texto (recordemos que los datos del csv se corresponden con un porcentaje pequeño de la totalidad de imágenes que tenemos, por lo que habría que relacionarlas).
- Añadir a la capa `layer_concatenate` comentada anteriormente, un input más, que fuese una capa `auxiliary_input` con las características extraídas de las imágenes tras un proceso potente de convolución.
- Para la obtención de las características de la imagen, se debería realizar algo similar a la función `extract_features` vista en clase.

5. Conclusiones

5.1 Conclusiones

En primer lugar, a lo largo de este trabajo, hemos podido verificar muchas de las conclusiones a las cuáles ya llegamos durante la Práctica 2. Entre ellas, podemos destacar:

- El componente subjetivo existente en el problema es muy fuerte, tal y como hemos podido ver en el análisis exploratorio de nuestro conjunto de textos. Ya es difícil, para una persona, determinar el rango correspondiente a una mascota, porque, no hay una solución objetiva y fundamentada que pueda justificar el adoptar antes o no a una mascota. Depende de muchas circunstancias, tanto de los gustos personales, como de la situación de esa persona, o las razones que puedan llevar a alguien a adoptar una mascota. Al haber tantas variantes no fijas, este problema se convierte en uno de muchísima mayor complejidad que el reconocimiento de imágenes, por ejemplo.
- Por otra parte, mencionar que la decisión de la adopción o no de una mascota, no se basará únicamente en una fotografía, sino que aquí entrarán muchas otras variables, como puede ser la edad o el sexo de la mascota. En el trabajo de investigación, entraremos en detalle en estos aspectos.

Hemos podido comprobar también, las grandes dificultades con las que nos podemos encontrar al trabajar con conjuntos en los que interviene texto. Por un lado, la falta de recursos a la hora de intentar crear el texto de manera individualizada y por nuestra cuenta. Hemos podido ver, la necesidad de utilizar técnicas, que nos permitan trabajar con estos textos de una manera más eficaz.

Por otra parte, mencionar la necesidad de trabajar con técnicas que nos permitan codificar los textos a un vector que sí pueda trabajar con la red neuronal. En nuestro caso, como el conjunto era lo suficientemente pequeño, esto nos ha permitido poder trabajar con conjuntos ya pre-entrenados, y poder obtener así resultados más o menos similares a los obtenidos en otros apartados y entrenamientos realizados en la práctica. De haber tenido un conjunto más grande de características en formato texto,

podríamos (o incluso, deberíamos) habernos planteado realizar nuestro propio embedding, específico y adaptado a nuestro problema, y así poder obtener mejores resultados en este problema.

De igual forma, y centrándonos ya en los resultados obtenidos con el tratamiento de los textos, hemos llegado a las siguientes conclusiones:

- La aplicación de modelos simples obtiene mejores resultados que los modelos complejos.
- Una RNN nos proporciona resultados muy buenos ya que procesa secuencias al iterar a través de los elementos de la secuencia y mantiene un estado que contiene información relativa a lo que se ha visto hasta ahora. Esta arquitectura permite que las RNN muestren un comportamiento temporal y capture datos secuenciales, lo que lo convierte en un enfoque más *natural* cuando se trata de datos textuales. Además, esta manera de procesar la información, está más relacionada con la forma en como los humanos procesan la información de manera incremental, ya que mantiene un estado interno de lo que están procesando. Y se construye un contexto a partir de información pasada y se actualizan constantemente a medida que entra la nueva información.

Por último, hacer hincapié en la necesidad de realizar ensembles para poder obtener buenos resultados en este problema. Nos vemos, a raíz de los resultados y posterior análisis, en la necesidad de tener que combinar los distintos tipos de información de la que disponemos, para así obtener buenos resultados. En otras palabras, para poder obtener buenos resultados en este problema, la entrada a nuestra red neuronal debe ser una combinación de texto, imágenes y otros metadatos, cada uno con su pre-procesamiento (o incluso entrenamiento) previo para poder tratarlos de manera simultánea.

5.2 Otros trabajos que se podrían realizar

Durante el trabajo se ha mencionado en alguna ocasión que podría ser aconsejable transformar nuestro problema multiclase a clasificación binaria. Ya que puede no tener sentido hacer uso de varias etiquetas para predecir el tiempo que se tarda en adoptar o no a una mascota, por lo que una posible línea de investigación sería transformarlo. Con ello, estaríamos resolviendo otro problema completamente distinto. Además, podría ser un buen camino para comparar que es más rentable si la aplicación de dos etiquetas o la aplicación de varias. No obstante, también podría haberse transformado el problema para 4 clases, debido a que es un problema desbalanceado, en donde la clase 0 contiene muy pocas etiquetas.

De otra manera, podríamos haber hecho uso de la creación de nuestro propio *embedding*, sin embargo, la limitación en la cantidad de datos, ha hecho que nos decanemos por modelos pre-entrenados, los cuales nos van a aportar más información.

Como hemos ido comentando, nos hemos quejado de la limitación de textos e imágenes que se nos proporciona. Por lo que estaría bien ampliar este tipo de información con otros datos textuales, como por ejemplo, descargando más información acerca de las adopciones de las mascotas y haciendo uso de más datos. Otro ejemplo podría ser, aumentar el conjunto de textos referente a anuncios de mascotas, para así poder obtener, un procesamiento de los textos de una manera más fiable, y sobretodo representativa en el dominio en el que nos movemos.

Por último, mencionar, que en este caso el problema a abordar ha sido de clasificación, pero, con lo que hemos ido comentando, se podría haber transformado en un problema de Análisis de Sentimientos con Redes Neuronales. Podemos ver aquí, la gran cantidad de posibilidades que se nos abren a raíz de las técnicas comentadas anteriormente

.

6. Referencias

Web

- [Bro] Jason Brownlee. *Best Practices for Text Classification with Deep Learning*. URL: <https://machinelearningmastery.com/best-practices-document-classification-deep-learning/> (véase página 21).