Proyecto
"Profiling Irony and Stereotype Spreaders on Twitter: IROSTEREO @ PAN"

Jose Javier Calvo Moratilla

Aplicaciones de linguística computacional
Curso 2021/2022

1. Introducción

En el presente proyecto se describe el trabajo realizado para la tarea *"Profiling Irony and Stereotype Spreaders on Twitter IROSTEO @ PAN"* donde se clasificar si un autor es irónico o no en base a un subconjunto de tuits.

El corpus utilizado se basa en un conjunto de doscientos tuits, escritos por usuarios en la red social twitter en lengua inglesa, autores que posteriormente se han etiquetado como altamente probables de ser irónicos o no irónicos.

El corpus precisa de un proceso inicial para la extracción de características, dónde se adecuan los datos para poder alimentar el modelo de clasificación.

En primer lugar se describe la lectura de la información desde los ficheros locales para su posterior utilización en el experimento.

En segundo lugar se definen los dos procesos de experimentación realizados para la resolución de la tarea. En cada proceso se ha seguido una técnica de codificación de la información para la extracción inicial de características y se han utilizado diferentes arquitecturas para los modelos de clasificación.

Los ejercicios del proyecto se realizan en la plataforma Google Colab Pro y se ejecutan mediante el uso de una GPU Tesla P100.

2. Lectura de los datos

Para la lectura de los datos se utiliza la librería *ElementTree XML* anteriormente utilizada en los laboratorios de la asignatura.

Los tuits de cada uno de los sujetos está en un fichero individualizado en formato *XML* y las etiquetas son leídas desde el fichero *"truth.txt"*.

Una vez leídos los datos se almacenan en memoria mediante el uso de *Numpy Array*, para poder experimentar con los datos cómodamente.

En la implementación se utiliza también la estructura de datos Data Frame para poder visualizar fácilmente la información leída de los ficheros y así comprobar que el proceso se ha realizado correctamente.

3. Clasificador TFID, Gramian, Convolucional

La primera aproximación consiste en utilizar *TFID vectorizer* visto con anterioridad en los laboratorios de la asignatura, donde se utiliza para la extracción inicial de características, de cada uno de los conjuntos de tuits de cada sujeto, una vez obtenido el vector se

transforma la información para generar una imágen de tipo *Gramian*, que posteriormente se utiliza un modelo de red convolucional para realizar la clasificación.

3.1. Preparación de los datos

Los datos y las etiquetas se leen de los ficheros locales mediante el uso de la librería *ElementTree XML*.

Una vez leídos los datos se utiliza *TfidfVectorizer de Sklearn* para codificar los tuits de un autor en única muestra, con un parámetro de 200 como máximo de características, dónde se obtiene una matriz con unas dimensiones de (420, 400, 200).

La codificación de los tuits de cada usuario se transforma en una *Gramian Image*. Para poder reducir el tamaño de la matriz a unas dimensiones reducidas, se utiliza la librería *OpenCV* para aplicar una interpolación bilinear a los datos.

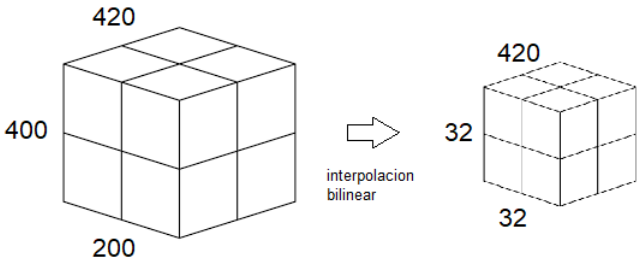

Figura 1: interpolación bilineal de los datos

Como resultado se obtiene una *Gramian Image* con forma (32, 32):

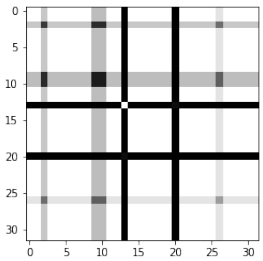

Figura 2: Transformación datos a Gramian Image

Después de haber codificado la información en una imagen se configura el *data augmentation* para intentar reducir el sobreaprendizaje del modelo clasificador, disponiendo así de un número más elevado de imágenes de entrenamiento.

Parámetro	Valores
width_shift_range	0.1
width_shift_range	0.1
horizontal_flip	False

Una vez preparado el *Data Augmentation* se definen los modelos de clasificación para realizar la experimentación, donde se han definido tres modelos, una ResNet 50v2 y una ResNet 18 y una RobertoNet utilizadas con anterioridad en los laboratorios de la asignatura Redes Neuronales Artificiales".

3.2. Experimentación

Se realiza los experimentos con los tres modelos descritos, dónde se entrenan el siguiente número de parámetros:

Modelo	Parámetros
ResNet50v2	23.568.898
ResNet18	11.191.938
RobertoNet	1.836.226

Las ejecuciones se realizan con la siguiente parametrización:

Parámetro	Valor
epochs	200
batch_size	4
Optimizer	Adam
Reductor LR	ReduceLROnPlateau
Learning Rate	1e-3

Durante el entrenamiento de los modelos se utiliza la herramienta *Early Stopper* para poder parar el entrenamiento si el algoritmo se estanca y no converge a una solución óptima.

3.3. Resultados

Después de la ejecución de los algoritmos para el entrenamiento se han obtenido los siguientes resultados:

Modelo	Accuracy
ResNet50v2	0.5374
ResNet18	0.5374
RobertoNet	0.5510

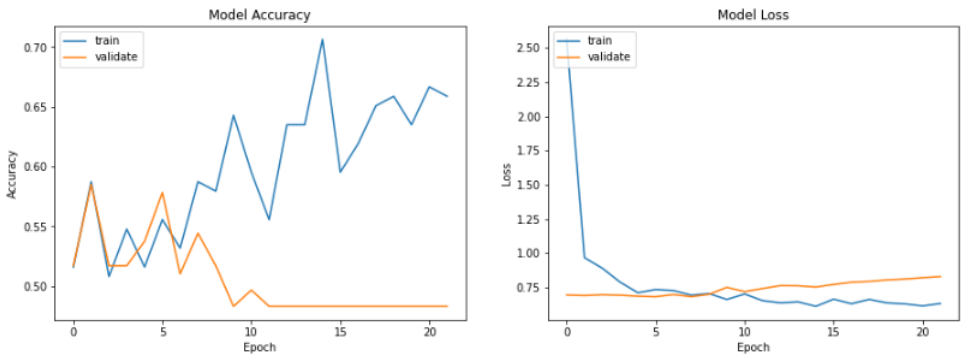

Figura 3: Resultados RobertoNet

3.4. Conclusiones

No se procede a utilizar otras métricas de evaluación, como por ejemplo el método *K_fold* dado que los resultados no han sido los esperados para la tarea.

La transformación de los datos obtenidos por el vectorizador *TFID* a imágenes en *Gramian* con la interpolación bilinear no ha permitido que los datos sean significativamente discriminativos para clasificar.

Una de las principales dificultades es poder elegir un *Data Augmentation* que no altere significativamente la codificación construida para la clasificación.

Otra de las soluciones para futuros trabajos sería poder utilizar otro tipo de codificación de los datos para ver si con dichos codificación las imágenes de tipo *Gramian* se comportan

mejor, pudiendo probar otro tipo de transformación basada en imágen que pueda ayudar a los datos para poder ser más discriminativos.

No se ha podido determinar el factor determinante para justificar los resultados obtenidos en la experimentación, por ello se decide implementar una nueva solución para obtener resultados más óptimos para la tarea.

4. Clasificador BERT, Media, Fully Connected

La segunda aproximación consiste en utilizar una arquitectura transformer para poder obtener las características de los tuits mediante *Sentence BERT*, una vez codificada la información se transforma en un vector dónde se obtiene la media de cada columna de la codificación. El vector resultante se utiliza como entrada en un modelo de clasificación basado en una arquitectura de red neuronal *Fully Connected*.

4.1. Preparación de los datos

Una vez obtenidos los datos de los ficheros en formato *XML* se utiliza una *Sentence Bert* pre entrenada para la obtención de un vector de características que identifique a cada tuit.

Los datos se almacenan en una matriz *Numpy* obteniendo unas dimensiones de (420, 200, 768), 420 sujetos, 200 tuits, 768 tamaño del vector de caracteristicas.

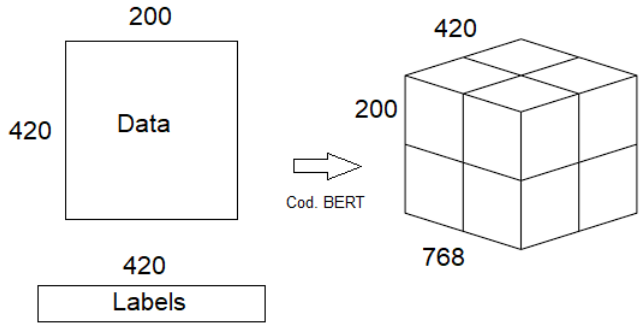

Figura 4: Transformación datos Inicial a BERT

Una vez codificada la información con la BERT Transformer se precisa realizar una nueva transformación para restar una dimensión a los datos y poder ser ejecutadas en una red neuronal *Fully Connected* sin que pierda sus características discriminativas.

Para ello se reduce una dimensión obteniendo la media de cada sujeto entre todos los tuits, obteniendo a la salida un array de (420, 768) 420 sujetos, 768 media de BERT.

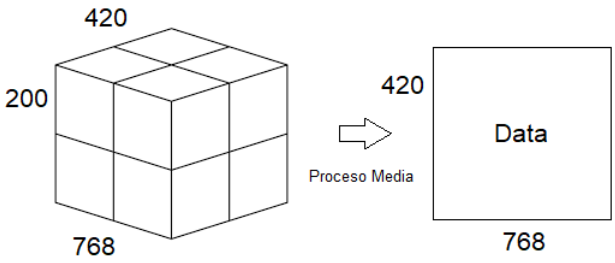

Figura 5: Transformación datos BERT a Media

Al reducir una dimensión se puede utilizar para la arquitectura neuronal definida para realizar los experimentos. A diferencia de la aproximación anterior se define una red de arquitectura *Fully Connected*.

4.2. Experimentación

Para realizar la experimentación se ha implementado una función con la que probar diferentes arquitecturas, la primera consiste en una red con capas densas reducidas, para intentar reducir el sobreaprendizaje, ya que se tienen pocos datos disponibles y ello puede empeorar los resultados obtenidos.

Consiste en un modelo con tres capas densas de (256, 128 y 2) neuronas, con 230,018 parámetros:

Layer (type)	Output Shape	Param #
dense_21 (Dense)	(None, 256)	196864
dense_22 (Dense)	(None, 128)	32896
dense_23 (Dense)	(None, 2)	258

Total params: 230,018
Trainable params: 230,018
Non-trainable params: 0

Figura 6: Resumen arquitectura, Modelo 2

En el segundo modelo se han aplicado las capas de *Batch Normalization, Gaussian Noise, Dropout* para ver el efecto que producen en una capa densa pequeña.

La estructura seguida consiste en (Densa, Batch normalization, Gaussian Noise y Dropout), cuyas capas densas son similares a las definidas en el modelo M1 con (256, 128 y 2)

```
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               196864

gaussian_noise (GaussianNoi  (None, 256)               0
se)

dropout (Dropout)            (None, 256)               0

activation (Activation)      (None, 256)               0

dense_1 (Dense)              (None, 128)               32896

gaussian_noise_1 (GaussianN  (None, 128)               0
oise)

dropout_1 (Dropout)          (None, 128)               0

activation_1 (Activation)    (None, 128)               0

dense_2 (Dense)              (None, 2)                 258

=================================================================
Total params: 230,018
Trainable params: 230,018
Non-trainable params: 0
```

Figura 7: Resumen arquitectura, Modelo 2

Se aplican valores reducidos para ver la influencia de dichos parámetros en la experimentación:

Parámetro	Valores
BN	True
GN	0.01
Dropout	0.01

Una vez definidas las arquitecturas y los parámetros individuales de cada una de ellas se define la parametrización general para experimento, con los siguientes parámetros:

Parámetro	Valores
batch size	16
epochs	200
optimizer	Adam
Reductor LR	ReduceLROnPlateau
Learning Rate	1e-3

4.3. Resultados

Una vez realizados los experimentos para los modelos M1 y M2 se han obtenido los siguientes resultados.

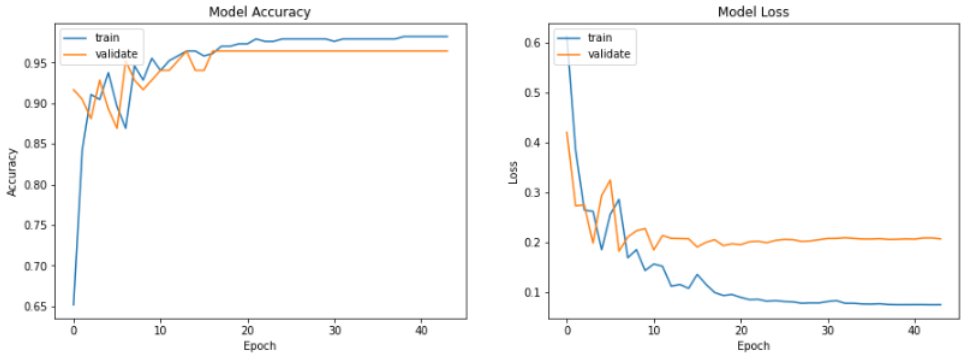

Figura 8: resultados Sentence BERT, Media, Modelo 1

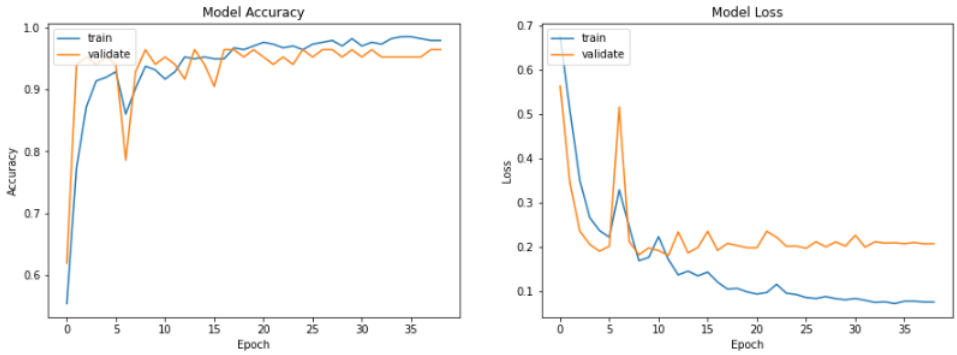

Figura 9: resultados Sentence BERT, Media, Modelo 2

Modelo	Accuracy
M1 Denso	0.9762
M2 Denso, BN, GN, Drop	0.9643

Resultados 10-Fold

Modelo	Accuracy
M1 Denso	0.9333
M2 Denso, BN, GN, Drop	0.9262

4.4. Conclusiones

Con el experimento realizado en la segunda aproximación se demuestra que la transformación de la codificación efectuada con *Sentence BERT* a la media de todos los tuits para un sujeto concreto es un buen vector de características discriminativo para la clasificación, a diferencia de la transformación efectuada en la primer aproximación.

La utilización de las capas de *Batch Normalization, Gaussian Noise y Dropout* empeoran ligeramente el rendimiento del modelo implementado, demostrando que el mejor resultado se obtiene con el modelo M1.

Las pruebas de validación 10-Fold finales dan unos resultados inferiores a la ejecución sin la validación, demostrando que el modelo M1 es el que muestra el mayor rendimiento con un Accuracy de 0.93 %.

Gracias a la realización de los laboratorios de la asignatura he podido comprender en profundidad el funcionamiento de una red Transformer y sus aplicaciones más directas en el reconocimiento de lenguaje natural.

Referencias

[1] Encoding Time Series as Images for Visual Inspection and Classification Using Tiled Convolutional Neural Networks. Zhiguang Wang, Tim Oates
Conference: Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 2015.

[2] ReduceLROnPlateau https://keras.io/api/callbacks/reduce_lr_on_plateau/

[3] A Python Package for Time Series Classification, GramianAngularField. https://pyts.readthedocs.io/en/stable/generated/pyts.image.GramianAngularField.html

[4] SentenceTransformers. https://www.sbert.net/

[5] Scikit Learn, TfidfVectorizer https://scikit-learn.org/