Trabajo Minería de Datos: Aspectos Avanzados

Carmen Biedma Rodriguez
Carlos Manuel Sequí Sánchez
Juan José Sierra González
Francisco González López

ID Grupo: **Spice Girls del Espasio Ecsterior**

1. Problema a resolver

La práctica a resolver consiste en construir y entrenar una red neuronal que clasifique las emociones transmitidas por tweets escritos en español. Este problema se puede encuadrar en el ámbito del procesamiento de lenguaje natural.

El problema se va a abordar mediante técnicas de aprendizaje automático, entrenando un modelo (red neuronal) para realizar la tarea de clasificación. Para entrenar contamos con 1008 mensajes extraídos de la red social Twitter. Por un lado, tenemos el texto del mensaje, y por otro, una etiqueta indicando el sentimiento del mismo, que puede ser P (positiva), N (negativa), NEU (neutral) o NONE (no hay opinión en el tweet). Contamos también con un conjunto de validación de 506 instancias, etiquetadas de la misma manera. Por último, el conjunto de test está formado por 1899 tweets que no contienen una clase asignada, y que servirán para evaluar el acierto de cada equipo en la competición alojada en la plataforma Kaggle.

2. Desarrollo de la práctica

Para trabajar con un problema de procesamiento de textos es habitual hacer uso de redes neuronales recurrentes. Estas redes están implementadas en diversas librerías (Tensorflow, Keras, PyTorch, etc.). Para el desarrollo de la práctica hemos utilizado Keras, ya que es el que hemos visto en clase, y es de los más accesibles. Esta librería implementa las capas y arquitecturas que se van a ver a continuación, y que hemos incluido en prácticamente todas nuestras redes desde el primer momento.

2.1. Red inicial y consideraciones generales

Para comenzar a plantear nuestro modelo, empezamos construyendo una red básica con los elementos que hemos considerado esenciales en una arquitectura para analizar lenguaje natural; recuperar los embeddings asociados a las palabras, hacerlos pasar por una capa con arquitectura recurrente (hemos empleado siempre una bidireccional) y utilizar una capa totalmente conectada para relacionar los valores obtenidos por la capa recurrente. Además, hemos utilizado dropout y regularización de los pesos de la red para mejorar la capacidad de generalización del modelo. Sobre este modelo básico hemos añadido modificaciones para intentar mejorar el rendimiento de la red. En todos los casos se ha utilizado la entropía cruzada categórica como función de loss. El número de épocas máximo en todos los casos ha sido de 200. Esto ha sido posible en parte gracias a usar una implementación basada en CuDNN (librería de Nvidia para redes neuronales profundas) de la capa LSTM, mucho más rápida que la LSTM "normal" (funcionalmente son prácticamente iguales). Así mismo, se ha utilizado la estrategia de early-stopping para parar la red cuando el resultado en validación no mejora durante más de un cierto número de épocas (30 generalmente), reducción de learning rate bajo

las mismas condiciones (partiendo de 0.001, utilizando el optimizador Adam, multiplicando el learning rate por 0.1 cada 10 épocas sin mejora en validación). Además, en todos los casos hemos utilizado los pesos de la época que mejor resultado en validación han obtenido, tanto para dar el resultado de validación como para predecir los datos de test.

En primer lugar, se ha realizado una red básica con la siguiente estructura:

- Capa embedding
- LSTM bidireccional con 8 neuronas
- Dropout de 0.25
- Capa densa con 32 neuronas, función de activación RELU y kernel regularizer "l2"
- Dropout de 0.25
- Capa densa con 4 neuronas (una por clase), función de activación SOFTMAX y kernel regularizer "l2"

Esta red inicial nos dió un loss de 1.1346 y un F1 de 0.58893 en el conjunto de validación. En el conjunto de test obtuvimos 0.57469.

2.2. Modificación de parámetros de la red

Partiendo de la red neuronal anterior, la primera alternativa que probamos fue editar los parámetros asociados a las distintas capas y tratar de optimizarlos. Estos parámetros a modificar en un primer momento fueron tanto el número de capas a incluir como el número de nodos que tendría cada capa. Esto nos permite teneruna idea de los valores en los que podemos movernos, pero la optimización de hiperparámetros para una arquitectura no nos garantiza nada para otras arquitecturas.

En esta etapa del proyecto se probaron muchos modelos con combinaciones de parámetros totalmente distintas, pero merece la pena mencionar concretamente dos de ellos que nos hicieron conseguir mejores resultados de lo que teniamos hasta ahora.

- Reducir las neuronas de la LSTM bidireccional a 4, reducir a 8 las neuronas de la primera capa densa y eliminar el dropout antes de la capa final de clasificación. **Resultado en validación: loss 1.1705327300214956 y F1= 0.6047430822971781. Resultado en test: 0.57996**
- Aumentar las neuronas de la LSTM a 40 y reducir a 20 las neuronas de la primera capa densa. **Resultado en validación: 0.604743. Resultado en test: 0.58347**

En la primera modificación se probó a eliminar el dropout previo a la capa de clasificación ya que no tenía sentido eliminar información de entrada para dicha capa. Después nos dimos cuenta de que éste cambio no afectaba mucho y que incluso iba mejor si la dejabamos, por lo que volvimos a añadirla más adelante. Por otro lado probamos tanto a reducir el número de neuronas en las capas ocultas como aumentarlo. Ambas opciones dieron buenos resultados y no variaban demasiado. Lo que si establecimos desde un principio fue no añadir muchas capas ya que no teniamos un conjunto de datos muy grande y esto podría ser contraproducente.

Otra de las cosas que cabe mencionar es que se realizó una comprobación de resultados con respecto a las medidas de loss y accuracy. De la ejecución de una red neuronal nos quedamos con el resultado que mejor accuracy daba y con el que mejor loss daba y nos dimos cuenta de que los resultados en test eran prácticamente iguales.

Modificando estos parámetros nos dimos cuenta de que la red podía mejorar pero no daba resultados buenos del todo, es por ello que decidimos añadir otras técnicas a la red para intentar mejorar lo que teníamos hasta ahora.

2.3. Residuales

Una técnica que nos ha dado buenos resultados es introducir conexiones residuales entre la salida de la LSTM y la primera capa totalmente conectada.

El esquema de la primera red residual que plantemos es el siguiente:

- Capa embedding
- LSTM bidireccional con 20 neuronas (Luego se lo sumaremos a la salida de la siguiente capa totalmente conectada)
- Dropout de 0.25
- Capa densa con 40 neuronas, función de activación RELU y kernel regularizer "l2"
- Sumamos la salida de la capa recurrente con la de la anterior capa totalmente conectada
- Dropout de 0.25
- Capa densa con 4 neuronas (una por clase), función de activación SOFTMAX y kernel regularizer "l2"

Esta arquitectura resultó en un accuracy de validación de 0.608696, mientras que en test resultó de 0.59929.

2.4. Cross Validation

Con el fin tanto de obtener unos resultados de mayor fidelidad en la validación de los modelos creados como de hacer que nuestros modelos aprendan con una cantidad mayor de datos, hemos decidido utilizar el conjunto de los datos train junto con los de validación para realizar una cross validation a la hora de realizar nuestros modelos en lugar de entrenar con los datos train y por otra parte comprobar el acierto de nuestros modelos con los datos de validación.

- Capa de entrada
- Capa dropout de 0.3
- Capa CuDNNLSTM con 40 neuronas y kernel regularizer l2 (Guardamos resultado en x_res para sumarlo más tarde)
- Capa dropout de 0.3
- Capa densa con 80 neuronas, funcion de activación elu y kernel regularizer l2.
- Añadir salida anterior con x_res
- Capa dropout de 0.3
- Capa densa con 40 neuronas, funcion de activación elu y kernel regularizer l2. (Guardamos resultado en x_res)
- Capa densa con 40 neuronas, funcion de activación elu y kernel regularizer l2. (Guardamos resultado en x_res)
- Capa dropout de 0.3
- Añadir salida anterior con x_res
- Capa dropout de 0.3
- Capa densa con 4 neuronas (una por clase), función de activación SOFTMAX y kernel regularizer "l2"

2.5. Atención

Otra técnica que hemos probado ha sido atención, aplicada directamente a las características del embedding. La implementación probada ha consistido en una capa totalmente conectada independiente en cada posición de la entrada, que nos proporcionaba un "score" para cada posición. A estos valores les aplicamos la función softmax, de forma que nos queda un vector de valores entre 0 y 1, y que suman 1, y que utilizamos como un "pesado" de las diferentes palabras de cada entrada. Sin embargo, no conseguimos obtener buenos resultados, por lo que no se ha incluido en el modelo final.

3. Modelo final seleccionado

Para la realización del modelo final se han encapsulado una serie de capas en un bloque al que nos referimos como fc_module, que tiene la siguiente forma:

- Capa densa con activación lineal sin usar término bias.
- Capa de normalización de batch
- Capa de activación elu
- Dropout de 0.2

A continuación se describe la arquitectura de la red que mejores resultados nos ha dado en la competición:

- Dropout de 0.3
- LSTM con 30 neuronas, kernel regularizer 'l2'
- Capa de normalización de batch
- Capa de activación elu (se almacena la salida en x_res)
- Dropout de 0.2
- fc_module
- Sumar salida anterior a x_res
- Dropout de 0.2
- fc_module (se almacena la salida en x_res)
- fc_module
- Sumar la salida anterior a x_res
- Dropout de 0.2
- Capa densa con 4 neuronas (una por clase), función de activación SOFTMAX y kernel regularizer "l2"

Descripción de la red	**F1 Validación**	**F1 Público**	**F1 Privado**
LSTM con 8 neuronas + Dropout 0.25 + Densa act ELU con 32 neuronas + Dropout 0.25	0.58893	0.57469	0.57368
LSTM con 4 neuronas + Dropout 0.25 + Densa act ELU con 8 neuronas sin usar Dropout previa a clasificación	0.60474	0.57996	0.58796
LSTM con 40 neuronas + Dropout 0.25 + Densa act ELU con 20 neuronas + Dropout 0.25	0.604743	0.58347	0.58872
Red residual con LSTM con 20 neuronas + Dropout 0.25 + Densa act ELU con 40 neuronas + Suma (LSTM + Densa) + Dropout 0.25	0.608696	**0.59929**	0.60225
Dropout 0.3 + LSTM con 40 neuronas + Dropout 0.3 + Densa act ELU con 80 neuronas + Add(LSTM + Densa) + Dropout 0.3 + Densa 1 act ELU con 40 neuronas + Densa 2 act ELU con 40 neuronas + Dropout 0.3 + Suma (Densa 1 + Densa 2) + Dropout 0.3	**0.644243**	0.59402	**0.60977**
Anterior + Atención	0.622189	*no subido*	*no subido*

Cuadro 1: Resultados de las redes neuronales subidas a Kaggle que nos han parecido más relevantes.

4. Trabajo futuro

En este aspecto cabría destacar algunos puntos clave que sería interesante probar continuando en la línea del proyecto:

- Utilizar la porción de validación de los datos para determinar la estrategia de entrenamiento (épocas, learning rate, cambios del learning rate, etc.), y después usando esta estrategia, entrenar con los datos de entrenamiento y los de validación.
- Explorar técnicas de aumento de muestras para datos de secuencia, ya que tenemos relativamente pocas muestras.
- Preprocesar los datos de entrada para intentar estandarizar el lenguaje utilizado.