Clasificación de tweets en español

Computación Inteligente

Máster Universitario en Sistemas Inteligentes y Aplicaciones Numéricas en la Ingeniería

Alberto Ramos Sánchez [1]

9 de febrero de 2021

[1] alberto.ramos104@alu.ulpgc.es

Índice

1. Introducción

En este trabajo se ha realizado *fine-tuning* sobre la versión en español de *BERT BETO* [1] para clasificar tweets en español. Para comparar resultados también se ha utilizado el modelo *LSTM*.

2. Dataset: TASS 2012-2019

El *dataset* utilizado ha sido extraído del TASS del SENLP de las competiciones entre el 2012 y 2019.[2] [3]

2.1. Descripción del *dataset* original

En cada competición de cada año se ofrecen distintos ficheros *XML* donde se encuentran los tweets. Estos ficheros fueron procesados para generar ficheros *csv* con los que trabajar con *pandas* en *Python*: se utilizaron scripts en *php* para leer los archivos *XML*, se cargaron en una base de datos *MySQL*, y finalmente se convirtieron las tablas de la base de datos a los 3 ficheros *csv* resultantes (*tweets.csv.gzip*, *topics.csv.gzip* y *polarities.csv.gzip*).

El formato del *dataset* varía levemente de un año para otro, aunque se mantienen ciertas etiquetas dentro de los archivos: el contenido del *tweet*, identificador único del *tweet*, la polaridad, la fecha, el idioma, el usuario.

2.2. Descripción del *dataset* recopilado

Tras el procesado de los distintos ficheros XML se obtuvieron 3 ficheros csv: *tweets.csv*, *topics.csv* y *polarities.csv*. En total, se recopilaron 31613 *tweets*.

En el fichero *tweets.csv* está el contenido de cada *tweet*. Sus columnas son:

- *tweetid*: identificador único del tweet.
- *user*: nombre de usuario quien escribió el tweet.
- *date*: fecha en la que se publicó el *tweet*.
- *lang*: idioma del tweet (todos son en español).
- *content*: texto del tweet.

En el fichero *polarities.csv* se etiquetan a los *tweets* con una etiqueta de polaridad. Sus columnas son:

- *polarityid*: identificador de la fila.
- *tweetid*: identificador del tweet al que se hace referencia. Actúa como clave foránea.
- *entity*: parte especial del *tweet*: *hashtags*, menciones, tweet citado, etc.
- *type*: etiqueta con valor .ªGREEMENT.º "DISAGREEMENT".
- *value*: polaridad del *tweet*. Tiene 6 categorías:
  - P+: positiva fuerte
  - P: positiva
  - NEU: neutra
  - N: negativa
  - N+: negativa fuerte
  - NONE: sin sentimiento (no sentiment)

En el fichero *topics.csv* se etiquetan a ciertos *tweets* con un tema. Sus columnas son:

- *topicid*: que identifica la fila de la tabla.
- *tweetid*: identificador del *tweet* al que hace.

Como la mayoría de *tweets* no están etiquetados por un tema, se ha decidido utilizar la etiqueta de polaridad para clasificar los *tweets*.

En total tenemos 31613 *tweets* divididos en 6 categorías:

- N con 6219 *tweets*.
- N+ con 976 *tweets*.
- NEU con 2755 *tweets*.
- NONE con 5597 *tweets*.
- P con 5442 *tweets*.
- P+ con 2793 *tweets*.

El *dataset* no está balanceado para cada categoría. Vemos que hay tres categorías con menos *tweets* que las demás NEU y P+ con aproximadamente 3000 y N+ que tiene mucho menos que las demás con aproximadamente 1000.

Balancear el *dataset* provocaría eliminar gran parte de los tweets resultando en un *dataset* de 2928 *tweets*. Por tanto, con estas categorías se han realizado dos pruebas:

- Agrupar los *tweets* en 3 categorías (dos negativos con N y N+, dos neutros con NEU y NONE, y dos positivos con P y P+) y balancearlo, quedando un *dataset* de 21585 *tweets* con 7195 *tweets* por categoría. De este modo puede balancearse el *dataset* sin perder tantos datos.
- Clasificar los *tweets* en 6 categorías sin balancear el *dataset*.

3. Limpieza del dataset

Para todos los clasificadores se ha realizado el mismo preprocesado de los tweets:

- Se han eliminado los nombres de usuario.
- Se han eliminado las URL.
- Se han eliminado la etiqueta # de los *hashtags*.
- Se han eliminado las vocales seguidas más de dos veces: convertimos *largooooooo* en *largoo*.
- Se han eliminado *stopwords*.
- Y se han seleccionado solamente caracteres alfanuméricos, eliminando emoticonos y cualquier caracter especial).

Posteriormente, se han *tokenizado* las palabras a valores numéricos, para ser utilizado posteriormente por la capa *embedding* de cada modelo. En los *notebooks* se detalla el *tokenizado* realizado para cada clasificador.

4. Modelos de clasificación

En este repositorio se muestran 3 tipos de clasificadores de texto: dos de ellos basados en *Bert*, y un *LSTM*.

Los modelos basados en *bert* utilizan las clases de *transformers* disponibles en el módulo *transformers* para *Pytorch* de *HuggingFace* [4]. Estos modelos permiten de forma muy sencilla cargar pesos de modelos ya entrenados. En este caso, se han utilizado los pesos del modelo *BERT* para español *BETO*. [1]

En el entrenamiento también se ha utilizado *early stopping* para evitar el sobreajuste de los modelos. La clase *EarlyStopping* pertenece a [5]. (https://github.com/Bjarten/early-stopping-pytorch)](https://github.com/Bjarten/early-stopping-pytorch).

4.1. BertModel

En este clasificador se ha adaptado la clase *BertModel* de *HuggingFace* para clasificar *tweets*. La entrada de este modelo consiste en un vector de la frase tokenizada, en cuya primera posición hay un *token* de clasificación *[CLS]*. La tokenización se realiza mediante la clase *BertTokenizer*.

El *token [CLS]* es utilizado por el clasificador para generar la predicción, tomando la salida del *transformer* para ese token y realizando una combinación lineal mediante dos capas *FFN*.

4.2. BertForSequenceClassification

En el módulo *transformer* se ofrece una clase ya preparada para clasificación de texto. La entrada de esta red es la misma que para el modelo anterior. Indicándole el número de etiquetas, permite clasificar los textos *tokenizados* en el número de categorías indicadas.

4.3. LSTM

Para comparar *Bert* con otro modelo, se ha utilizado el modelo *LSTM* que trae *Pytorch*. La entrada de este modelo está generada por las clase *Field* de *torchtext*, que transforma cada palabra a un índice tras construir un vocabulario con el texto aportado. Este vector es utilizado por la capa *embedding* para generar vectores de palabras los cuales son linealizados por una capa *FFN*.

5. Resultados

Cada modelo ha sido testeado sobre: dos conjuntos de datos, agrupados en 3 categorías y en 6 categorías; y en varios conjuntos de hiperparámetros. Los parámetros utilizados y los resultados de cada clasificador se encuentran detallados en cada *notebook*.

Los modelos obtenidos para cada configuración de hiperparámetros se incluyen dentro de la carpeta *modelos*. Cada modelo está acompañado de un fichero *parametros.txt* que indica que hiperparámetros se utilizaron para su entrenamiento. Utilizando el método *load_from* de cada clase, pueden cargarse los pesos de dichos modelos.

Como conclusión, el BERT es capaz de alcanzar buenos resultados si el *dataset* está balanceado, superando al LSTM en todas las pruebas. El tiempo para entrenar BERT es mayor que para otros clasificadores y, en la mayoría de las pruebas, requiere que el valor de *learning rate* sea bajo para que alcance buenos resultados.

Referencias

[1] José Cañete, Gabriel Chaperon, Rodrigo Fuentes, Jou-Hui Ho, Hojin Kang, and Jorge Pérez. Spanish pre-trained bert model and evaluation data. Disponible en `https://github.com/dccuchile/beto`.

[2] SEPLN. Tass - sepln (taller de análisis de sentimientos en la sepln). Disponible en `http://tass.sepln.org`.

[3] SEPLN. Tass. dataset de competiciones. 2012 a 2019. Disponible en `http://tass.sepln.org/tass_data/download.php`.

[4] Hugging Face. Bert model. Disponible en `https://huggingface.co/transformers/model_doc/bert.html`.

[5] Bjarte Mehus Sunde. Early stopping for pytorch. Disponible en `https://github.com/Bjarten/early-stopping-pytorch`.