Problema 5: Análisis de sentimientos

Descripción del problema

El análisis de sentimientos abarca todo aquello relacionado con el tratamiento computacional del sentimiento, la subjetividad u opinión expresada en texto. Este campo de NLP es muy diverso y puede suponer varias tareas, como por ejemplo, la obtención de los aspectos sobre los que se opina con respecto a un objeto de opinión [1] ("Imagen" es un aspecto del objeto de opinión "Televisión") o la clasificación de la polaridad de una frase de opinión, esto es, decidir si en una frase el autor de la frase muestra un sentimiento positivo, negativo o neutro.

En definitiva, existen diversas tareas asociadas al Análisis de sentimientos. En este problema se propone abordar el problema de clasificación de la polaridad en un texto. Se trabajará sobre el conjunto de datos en el dominio de *Tweets* del conjunto de datos Sentiment140 , formado por 1.6 millones de tweets en el conjunto de Train y 498 tweets en el conjunto de test etiquetados como polaridad positiva (clase '4') y polaridad negativa (clase '0').

Descarga de datos

Los datos se pueden obtener de muchas maneras al ser un conjunto de datos comúnmente utilizado, pero la más sencilla es utilizar la librería Tensorflow_datasets .

Metodologías propuestas

Para abordar el problema, se proponen las siguientes aproximaciones en orden creciente de complejidad:

1. Empleo de métodos de análisis de sentimientos (SAMs) preentrenadas para la clasificación de la polaridad y análisis del funcionamiento de estos. Como propuestas:
    a. CoreNLP: https://stanfordnlp.github.io/stanza/
    b. VADER: https://github.com/cjhutto/vaderSentiment
    c. TextBlob: https://github.com/sloria/TextBlob
    d. Syuzhet: https://cran.r-project.org/web/packages/syuzhet/vignettes/syuzhet-vignette.html (Implementación en R).

    Se anima a explorar otros métodos preentrenados de análisis de sentimientos (Sentistrength, MeaningCloud, Azure Text Analytics,…).

2. Creación de un propio método de análisis de sentimientos (un clasificador sencillo). Para ello habrá que caracterizar las frases por medio del *tf-idf*, donde cada frase se caracteriza según la frecuencia de sus términos dentro de la propia frase y en relación al resto del conjunto de textos. Aquí se muestra un ejemplo en Python, tanto calculando el *tf-idf* de forma manual como empleando el paquete Sklearn. Una vez caractericemos cada frase por su *tf-idf*, podemos entrenar cualquier clasificador clásico (Random Forest, SVM, etc.).

    Al haber 1.6 millones de tweets, los tiempos de ejecución pueden dispararse, por lo que se recomienda realizar este proceso empleando un subconjunto representativo y balanceado del conjunto de Train. También se recomienda el empleo de técnicas de preprocesamiento de texto, como la eliminación de *Stop words* (términos sin influencia en la semántica de la frase como artículos) o lematización. Aquí se muestra un ejemplo de preprocesamiento empleando sobre todo la librería nltk.

3. Comparación y análisis de los modelos empleados (incluyendo el clasificador propio) y estudio de posibles combinaciones con el fin de obtener un clasificador más robusto [2].

Ampliaciones

Se propone como ampliación de la práctica el uso de modelos más avanzados:

1. Con respecto a la representación del texto se propone explorar el uso de *Word embeddings*. Estos son una representación vectorial densa de palabras que contienen información semántica. Esta representación se puede entrenar sobre el propio conjunto de datos. Sin embargo, existen modelos preentrenados sobre grandes conjuntos de datos que se pueden emplear, como por ejemplo Word2vec [3]
2. , Glove, fastText, BERT, etc. Algunos de estos modelos se pueden descargar en http://vectors.nlpl.eu/repository/ . Se tratan de unos diccionarios donde cada palabra tiene asignada una representación vectorial. Es importante que los Word embeddings coincidan con la naturaleza de vuestra base de datos (mismo idioma, mismo preprocesamiento, etc.).
3. Aplicación de modelos de Deep Learning. Debido a las dependencias temporales existentes en el texto, los modelos de Deep Learning que mejor funcionan son las redes neuronales recurrentes (por ejemplo LSTM). Recomendamos el empleo de estas técnicas en combinación con representación por medio de *Word embeddings* como en los siguientes tutoriales https://www.kaggle.com/oksanakalytenko/glove-lstm , https://www.kaggle.com/jackttai/twitter-sentiment-classification-keras-lstm . También se pueden probar otras arquitecturas neuronales.

Referencias

[1] Ana Valdivia, Eugenio Martínez-Cámara, Iti Chaturvedi, Maria Victoria Luzon, Erik Cambria, Yew Ong, Francisco Herrera, What do people think about this monument? Understanding negative reviews via deep learning, clustering and descriptive rules. *Journal of Ambient Intelligence and Humanized Computing*.

[2] Miguel López, Ana Valdivia, Eugenio Martínez-Cámara, M. Victoria Luzón, Francisco Herrera, E2SAM: Evolutionary ensemble of sentiment analysis methods for domain adaptation, *Information Sciences*, Volume 480, 2019, Pages 273-286

[3] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, Efficient Estimation of Word Representations in Vector Space, 2013