TFA Inteligencia Artificial Avanzada

Mediante probabilidades

Preprocesamiento

Para el preprocesamiento se utiliza las librerías ***nltk*** y ***re***.

El paquete ***re*** se utiliza para la eliminación de links, nombres de usuario, *hastags* y etiquetas estilo HTML. Se planteó usarse también para la eliminación de emoticonos, pero posteriormente se decidió que los emoticonos tienen valor a la hora de clasificar.

Por otro lado, ***nltk*** se utiliza para comprobar si son palabras claves y lematizar. Para eso, se tokeniza y se obtiene el tipo de palabra (adj, adv, verb, noun). Tras esto, con la palabra y el token de su tipo, se lematiza y se sustituye la palara original por la tokenizada. Solo se va a lematizar en el caso de que la palabra no sea una palabra clave (p. ej. *a*, *is*, *do*, *however*, etc.) y no contenga números. En el caso de que no sea posible la lematización (es un nombre propio o palabras de índole social como *xd*, *lol*, *hahaha*), se añaden al vocabulario sin modificar.

Librerías usadas

- **nltk** para la tokenización y lematización
- **re** para la eliminación de usuarios, hastags, links, etc.
- **pandas** para la lectura del dataframe y la división del mismo por clases (`aprendizaje.py`)
- **math** para el cálculo del logaritmo neperiano de las probabilidades
- **string** para el tratamiento de las cadenas de texto, saber si contiene números y pasar a minúsculas

Programas

`vocabulario.py`

Se encarga de obtener todo el vocabulario de un conjunto de tweets. Para eso, obtiene todos los tweets del corpus de entrenamiento, aplica el preprocesamiento y exporta en `vocabulario.txt` todas las palabras del corpus, sin repetirlas.

`aprendizaje.py`

Se encarga de desarrollar los modelos para los tweets positivos y negativos. Para ello, divide el corpus de entrenamiento en dos listas según si son positivos o negativos, aplica el preprocesamiento a cada lista y calcula en cada grupo la frecuencia de las palabras del diccionario, así como su probabilidad para luego el logaritmo neperiano de la misma. Finalmente, exporta en dos ficheros (uno para la clase positiva y otro para la negativa), donde aparecen todas las palabras del vocabulario que superan un número de apariciones totales, con la frecuencia en cada corpus y el logaritmo neperiano de la probabilidad usando dicha frecuencia.

Tiene una variable interna *k* que indica el número de apariciones que debe superar cada palabra en todo el corpus para no ser sustituido por el token `__unknown__` . En la versión actual, k = 10, es decir, entre todos los tweets positivos y negativos, cada palabra debe aparecer más de 10 veces. Cada modelo almacena las palabras con la frecuencia en los tweets de la clase del modelo y el logaritmo neperiano de su probabilidad, pues si se usara la probabilidad normal habrían problemas de underflow o de precisión, pues la probabilidad de una única palabra extraña entre decenas de miles de tweets puede ser ínfima.

`clasificacion.py`

A partir de los modelos, clasifica un conjunto de mensajes como positivos o negativos. Para ello, aplica el preprocesado a los mensajes, y para clasificar cada mensaje, suma los logaritmos neperianos de las probabilidades de cada palabra en cada clase (positiva o negativa). Tras calcularlo con todas las palabras y sumarlo, tendrá dos valores, la probabilidad de aparición de las palabras en el modelo positivo y del modelo negativo. Clasificará el mensaje como la clase con mayor calor de entre las dos variables. Tras esto, genera dos ficheros. Uno, `resumen_aluXXX.txt` donde imprime P o N por cada mensaje según su clasificación. El otro, `clasificacion_aluXXX.txt` imprime los primeros 10 caracteres de cada mensaje, la suma de las probabilidades de las palabras del mensaje en el modelo positivo, en el modelo negativo y finalmente la clase en la que se clasificó el mensaje.

Tiene dos modos, uno en el caso de que el fichero de entrada solo contenga mensajes y otro en el caso de que contenga el valor real de la clase (debug). En este último caso, muestra el error de las predicciones con el valor real. Actualmente, el acierto es sobre el 70%.

Mediante redes neuronales

Preprocesamiento en redes neuronales

Para el preprocesamiento se utiliza las librerías ***nltk*** y ***string***.

El paquete ***string*** se utiliza únicamente para la eliminación de signos de puntuación

Por otro lado, ***nltk*** se utiliza para comprobar si son palabras claves y lematizar. Para eso, se tokeniza y se obtiene el tipo de palabra (adj, adv, verb, noun). Tras esto, con la palabra y el token de su tipo, se lematiza y se sustituye la palara original por la tokenizada. Solo se va a lematizar en el caso de que la palabra no sea una palabra clave (p. ej. *a*, *is*, *do*, *however*, etc.) y no contenga números. En el caso de que no sea posible la lematización (es un nombre propio o palabras de índole social como *xd*, *lol*, *hahaha*), se añaden al vocabulario sin modificar.

Librerías usadas con redes neuronales

- **nltk** para la tokenización y lematización
- **sklearn** para el aprendizaje y predicción mediante redes neuronales
- **pandas** para la lectura del dataframe y la división del mismo por clases (`aprendizaje.py`)
- **string** para el tratamiento de las cadenas de texto, saber si contiene números y pasar a minúsculas
- **numpy** para la transformación de dataframe a array convencional

Programa en redes neuronales

Primero se importa el conjunto de datos de entrenamiento y se genera un vector TfidfVectorizer, donde se especifica que se usarán unigramas (ngram_range) y que la K (número de repeticiones para que no se considere \<unk>) valdrá 1 (min_df). Posteriormente se crea un modelo linear, que será especificamente LogisticRegression. Se entrena y posteriormente se clasifica el corpus de testeo. En el caso de que el corpus de testeo contenga el resultado real, se ejecutará una función que calcule la precisión del modelo. En el caso de que no se tenga, se clasificará y se generarán los ficheros clasificacion_XXX.txt y resumen_XXX.txt

Author

Pablo Pérez González
alu0101318318@ull.edu.es