Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Inteligencia Artificial Avanzada:

Proyecto Final:

Clasificador de textos.

Saúl Sosa Díaz
(alu0101404141@ull.edu.es)

La Laguna, martes 2 de Abril de 2023

Índice:

Preprocesado.

Para preprocesar las noticias se han seguido los siguientes pasos.

- **Eliminación de números y símbolos de puntuación.**

- **Eliminación de las StopWords inglesas**: se han eliminado las StopWords de la lengua inglesa que se encuentran en la librería nltk.

- **Eliminación de marcas HTML**: se han borrado todas las marcas HTML utilizando la librería BeautifulSoup.

- **Conversión a minúsculas**: todas las letras de las noticias han sido convertidas a minúsculas para evitar problemas de sensibilidad a las mayúsculas.

- C**orrección ortográfica**: se ha utilizado la librería TextBlob para corregir cualquier error de ortografía en el texto.
  Para mejorar el tiempo de ejecución del programa, se ha implementado una estrategia que consiste en verificar si la palabra existe en un diccionario de la librería de nltk antes de llevar a cabo la corrección ortográfica. Si la palabra está en el diccionario, no se realiza ninguna corrección y se procede directamente a la lematización. De esta manera, se ha logrado reducir el tiempo que se dedica a la corrección ortográfica, que es la tarea que más tiempo consumía en el programa.

- **Lematización**: se ha utilizado la librería spacy para llevar a cabo la lematización, lo que significa que se han transformado las palabras de las noticias a su forma base, eliminando inflexiones gramaticales y simplificando el análisis semántico.

- **Comprobar que la palabra resultante se encuentre en un diccionario.**
  El diccionario seleccionado para esta tarea es el aportado por nltk.

Librerías utilizadas.

Para ejecutar este programa son necesarias las siguientes librerías:

- Pandas: Se utiliza para manipular y analizar datos.

```
Python
python3 pip install pandas
```

- NLTK: Se utiliza para el procesamiento del lenguaje natural.

```
Python
python3 pip install nltk
```

- BeautifulSoup 4: Para analizar y extraer datos de documentos HTML y XML.

```
Python
python3 pip install beautifulsoup4
```

- TextBlob: Se utiliza para realizar tareas de análisis de texto como etiquetado de partes del discurso, análisis de sentimientos, análisis de sujetos y extracción de frases clave.

```
Python
python3 pip install textblob
python3 -m textblob.download_corpora
```

- Spacy: Se utiliza para realizar tareas avanzadas de procesamiento de texto, como etiquetado de partes del discurso, análisis sintáctico, reconocimiento de entidades nombradas, análisis semántico y más.

```
Python
python3 pip install spacy
python3 -m spacy download en_core_web_sm
```

- tqdm: Es una librería de Python que se utiliza para agregar barras de progreso a los bucles o iteraciones en los que se requiere un seguimiento del progreso. En el proyecto utilizado para llevar la cuenta de las palabras y noticias que se están preprocesando.

Estructura de directorios.

.
├── src ← Contiene el código fuente del programa.
├── data ← Contiene el ficheros útiles para la ejecución.
├── corpus ← Contiene los corpus necesarios para la creación de los modelos.
├── solutions ← Contiene las soluciones al problema actual.
└── models ← Contiene los modelos generados por el programa.

En el directorio "data" se encuentran alojados los archivos correspondientes al conjunto de datos de entrenamiento y validación. Adicionalmente, se lleva a cabo el proceso de generación del vocabulario y el preprocesamiento de las noticias de validación. Este procedimiento se realiza con la finalidad de evitar el procesamiento constante de las noticias de validación en caso de requerir la ejecución del programa en múltiples ocasiones.

El archivo que contiene las noticias preprocesadas tiene asignado un nombre constante, *NewsProcessed.txt*. Cada noticia se presenta en una línea separada por caracteres de retorno de carro, es decir, mediante el uso del símbolo "\n".

Inicialmente los directorios *corpus*, *solutions* y *models* se encontrarán vacíos. No obstante, con el fin de evitar retrasos innecesarios, estos directorios ya tendrán sus respectivos ficheros en la entrega del proyecto. Es importante destacar que cualquier persona que descargue el repositorio deberá ejecutar el programa con todos los parámetros que se detallan en la siguiente sección, para asegurar el correcto funcionamiento del programa.

Instrucciones de uso.

Para ejecutar el programa hay que situarse en la carpeta principal del proyecto y lanzar el siguiente comando:

```python
python3 ./src/main.py
```

El programa admite varios parámetros:

Parámetro	Utilidad
-v, --vocab	Crea el vocabulario de las noticias de entrenamiento.
-c, --corpus	Crea los corpus y los guarda en la carpeta corpus.
-m, --models	Crea los modelos.
-t, --text	Preprocesa el texto a clasificar

Para garantizar que el programa se ejecute correctamente, se ha implementado una funcionalidad que obliga a proporcionar los parámetros necesarios en caso de que falte algún archivo.

Implementación del programa.

El programa se divide en múltiples ficheros.

- aciertos.py: Utilizado para estimar los errores.

  Este microprograma lee los ficheros *resumen_alu0101404141.csv* y el fichero que contiene los resultados correctos, comparando los resultados y dando el porcentaje de acierto del clasificador

- categorizeText.py:

  Utilizado para categorizar las noticias, producirá en la carpeta *solutions* dos ficheros, *clasificacion_alu0101404141.csv* y *resumen_alu0101404141.csv*.

  En este fichero se compara las probabilidades de que una noticia pertenezca a una clase u a otra y le asigna la mayor.

  Se utiliza una función llamada *getProb* que tiene como parámetros el modelo, las noticias preprocesadas y el número total de noticias. Esta función devuelve una lista con todas las probabilidades de las noticias en orden, esto se hace para los tres modelos, y se va comparando posición a posición que clase tiene mayor probabilidad en cada noticia.

- createCorpusTrain.py:

  En este fichero se crean los corpus, los cuales serán guardados en la carpeta *corpus* con los nombres constantes de:

  - ➔ corpusN.txt
  - ➔ corpusP.txt
  - ➔ corpusT.txt

- dependencies.py:

  Fichero con las librerías necesarias para la ejecución del programa.

- createModels.py:

  Este fichero se encarga de crear los modelos. Los cuales son guardados en la carpeta *models* con los nombres por defecto:

  - ➔ modelo_lenguaje_N.txt
  - ➔ modelo_lenguaje_P.txt
  - ➔ modelo_lenguaje_T.txt

- createVocab.py:

  En este fichero se crea el vocabulario. Se guarda en la carpeta *data* con el nombre *vocabulario.txt*

- main.py:

  Este es el fichero principal del programa y gestiona si es necesario crear algún fichero antes de categorizar la noticia.

- preprocessText.py:

  Preprocesa las noticias a categorizar y crea un nuevo fichero en la carpeta *data*. Se hace de esta manera para no tener que preprocesar las noticias todas las veces que se ejecuta la aplicación y evitar los extenuantes tiempos de ejecución.

- Los siguientes ficheros contienen las funciones que hacen que funcionen los ficheros anteriores.
  - utilsCorpus.py
  - utilsVocab.py
  - utilsText.py
  - utilsModels.py

  La función más relevante de estos ficheros es *getProb*, y se encuentra en utilsModels.py, a continuación la definición de la misma.

```python
def getProb(model, news, numberOfNews):
    """
    Given a model, news, and the number of news articles, calculate the probability of
the news.
    @param model - the model used to calculate the probability
    @param news - the news article
    @param numberOfNews - the number of news articles
    @return - List The probability of the news article
    """
    model = model.split('\n')
    numberOfNewsThisType = int(model[0].split(":")[1])
    probs = []
    # Create a dictionary with the words to make it easier to look them up and
calculate probability
    words = {} # key: word, value: log probability
    for j in range(2, len(model)):
        words[model[j].split(" ")[0].split(":")[1]] = float(model[j].split(":")[-1]) #
The last element is the log probability
    for sentence in news.split('\n')[:-1]:
        prob = 0
        for word in sentence.split(" "):
            try:
                prob += words[word]
            except:
                prob += words["<unk>"]
        prob += math.log(numberOfNewsThisType/numberOfNews) # Add the probability of
the type of news
        probs.append(prob)

                return probs
```

En esta función, se genera un diccionario cuyas claves corresponden a las palabras y sus valores están conformados por las probabilidades logarítmicas en base *e*. La finalidad de crear este diccionario es para optimizar la velocidad de ejecución del programa y evitar la necesidad de realizar búsquedas en cada palabra. Al utilizar una función hash para la búsqueda, el proceso resulta considerablemente más rápido que si se utilizara una lista.

Asimismo, se aprovecha la propiedad de que si una palabra no se encuentra en el diccionario, se generará la excepción *KeyError*, y en su lugar, se empleará la probabilidad logarítmica en base *e* de la palabra *unk*. De esta manera, se garantiza una ejecución fluida y eficiente del programa.

Estimación de errores.

Se ha observado que, al utilizar los mismos datos para el entrenamiento y la validación del modelo, se obtiene un porcentaje de acierto del 89.08%.

Por otro lado, al dividir los datos en dos conjuntos, uno para el entrenamiento y otro para la prueba, se ha obtenido un porcentaje de acierto del 77.8%.

Para poder ver el % de acierto del clasificador con los datos divididos debemos introducir el siguiente comando:

```python
python3 ./src/aciertos.py
```