Proyecto: Procesamiento de Textos en Lenguaje Natural

Sergio Nicolás Seguí

Enlace de GitHub

0. Ejecución del programa

cargo run <fichero_entrenamiento_vocabulario> <fichero_salida_vocabulario> <fichero_entrenamiento_corpus> <fichero_test_corpus> <fichero_salida_modelo_lenguaje_p> <fichero_salida_modelo_lenguaje_s> <fichero_salida_clasificación> <fichero_salida_resumen> <true | false>

Ejemplo funcional dentro de los ficheros del proyecto:

- cargo run src/data/PH_train.csv vocabulario.txt src/data/PH_train1000.csv src/data/PH_test5000.csv modelo_lenguaje_P.txt modelo_lenguaje_S.txt clasificacion_alu0101389936.txt resumen_alu0101389936.txt true

El true o false es para evaluar si el fichero de prueba tiene los resultados e imprimir el porcentaje de acierto.

1. Implementación de la práctica de Procesamiento Natural

Esta práctica, se ha decidido hacer en el lenguaje de programación de rust. Hay varias estructuras que se han creado para la realización de dicha práctica, como pueden ser:

- Una estructura para el vocabulario.
- Una estructura para la creación de cada corpus.
- Una estructura para las noticias.

1.1 Preprocesamiento:

Para el preprocesamiento lo que he hecho es lo siguiente:

- Si todas las palabras son signos de puntuación, se convierte a <PUNCT>
- Si todos los elementos de la palabra son números, se convierte a <NUM>
- Si algún elemento de la palabra no es un símbolo alfabético, se convierte a <PUNCT>
- Además, hago un procesamiento de palabras, de forma que cada vez que una palabra aparezca más de 3 veces, se devolverá esa palabra con solo tres de esas letras. Ejemplo: practicaaaaaa -> practicaaa.

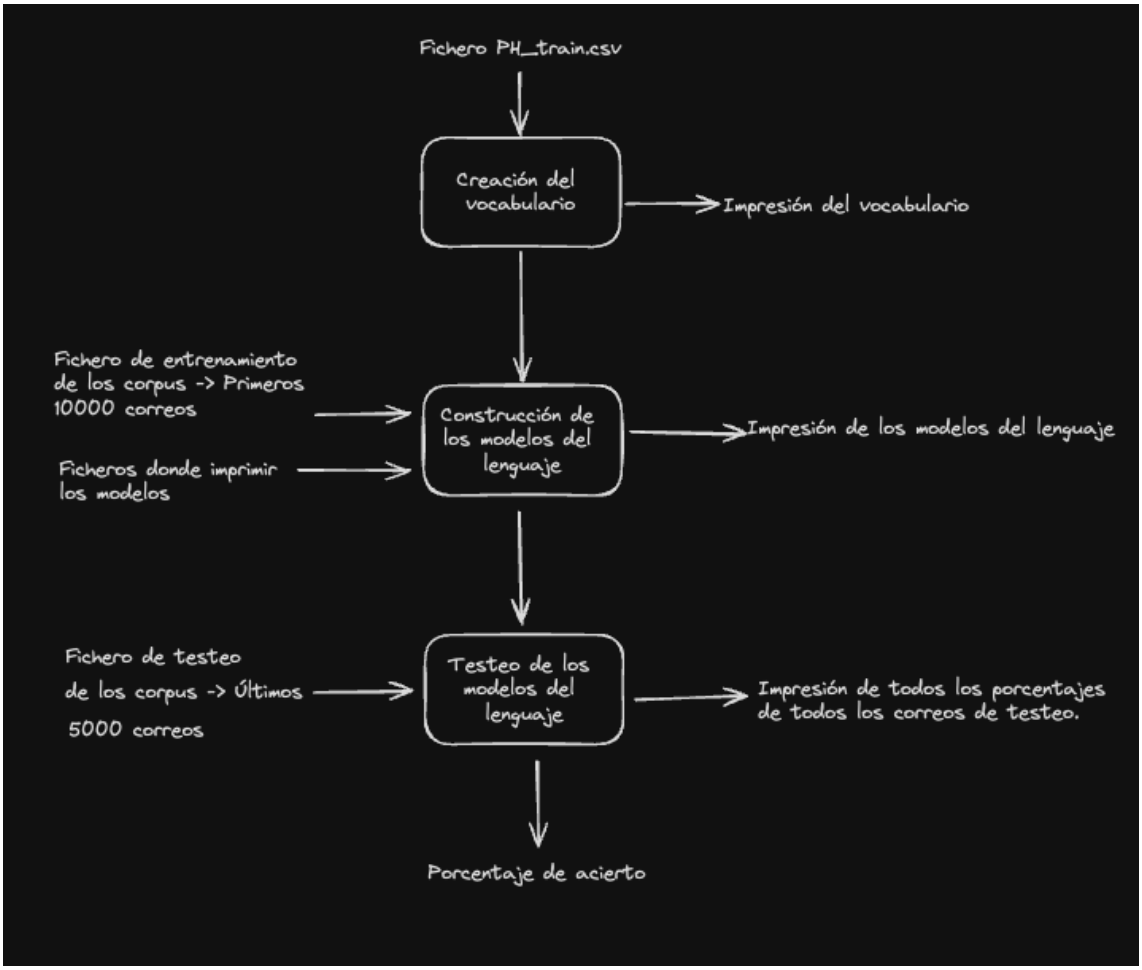
- Por último, se realiza un proceso de lematización de la palabra procesada en minúscula. (Al final este proceso se eliminó porque da peores resultados)

1.2 Librerías utilizadas

La única librería que he utilizado para la realización de la práctica, es una para el proceso de lematización llamada rust-stemmers. -> Lo cual al final se eliminó ya que aumenta x3 el tiempo de ejecución y da peores resultados.

1.3 Implementación del programa

En esta práctica, he decidido que se realizase el proceso completo. Otra opción, sería realizar un programa para cada parte, pero me parecía más interesante realizarlo de esta manera. A continuación, dejo un esquema de cómo sería la estructura del programa:



1.4 Estimación de errores

Porcentajes de acierto para los siguientes casos:

- Habiendo entrenado el modelo con todos los correos: 95,73%
- Habiendo dividido el PH_train en dos, uno con los 10000 primeros correos para el entrenamiento, y otro con los 5000 últimos correos para el testeo: 96,46%