Práctica I: Tokenización

Pablo Chantada Saborido
Marcelo Ferreiro Sánchez
Lucía Bardanca Rojo

Grupo 2: Viernes

1 Manual de Usuario

El programa se compone de **5 archivos**, con 3 documentos de texto para analizar el comportamiento de los algoritmos. Estos archivos presentan la implementación de algoritmos de tokenización tradicionales como *n-granms*, y otros más avanzados como *Byte Pair Encoding (BPE)* o *WordPiece*. A continuación se muestra un listado explicando cada archivo:

- **tokenize_simple.py** Implementa una clase con 3 métodos estáticos, tokenización por espacios, signos y n-grams
- **bpe.py**: Implementación del algoritmo Byte Pair Encoding.
- **wordPiece.py**: Implementación del algoritmo WordPiece.
- **evaluate_models.py**: Evaluación de los algoritmos BPE y WordPiece con un tamaño de vocabulario de 150.
- **visual_evaluation.py**: Comparación visual de los 5 algoritmos implementados. Para los algoritmos de BPE y WordPiece se ha utilizado un entrenamiento por lotes (crecimiento de 500 palabras por lote), y para la visualización se ha realizado una interpolación de los valores para un resultado más legible.

Para ejecutar cualquiera de los archivos, simplemente hágalo por línea de comandos como un archivo de python normal:

```
python nombre_del_archivo.py
```

Si falta alguna dependencia, ejecutar el comando:

```
pip install scipy regex numpy pandas matplotlib
```

2 Análisis de Resultados

Para analizar los resultados tenemos dos métodos en este caso: un archivo de entrenamiento *(training_sentences.txt)* y un archivo de test *(test_sentences.txt)*, un archivo más largo para ver cómo avanza el tamaño del vocabulario a medida que aumentamos las oraciones procesadas *(majesty_speeches.txt)*.

Los métodos tradicionales cumplen correctamente con su función. Al ser algoritmos simples y sencillos, podemos esperar un funcionamiento seguro, pero con desventajas apreciables:

- Texto: El gato duerme tranquilo *emoji*.
- Tokenización por espacios: ['El', 'gato', 'duerme', 'tranquilo', '*emoji*.']
- Tokenización por signos de puntuación: ['El', 'gato', 'duerme', 'tranquilo', '*emoji*', '.']
- Tokenización en n-gramas (n=2): ['El gato', 'gato duerme', 'duerme tranquilo', 'tranquilo *emoji*.']

El método de BPE también obtiene un resultado correcto, comparado con la solución de ejemplo. Sin embargo, debido a los empates de pares más frecuentes, se producen leves variaciones como: ra-ta, en vez de rat-a. Tanto la solución como la salida del BPE, están realizadas con un tamaño máximo de vocabulario de 150.

El método de WordPiece, al igual que los métodos anteriores, cumple con su objetivo de tokenización correctamente. En este caso, no se han observado diferencias significativas entre la salida de WordPiece y la solución de referencia.

2.1 Discusión de Métodos

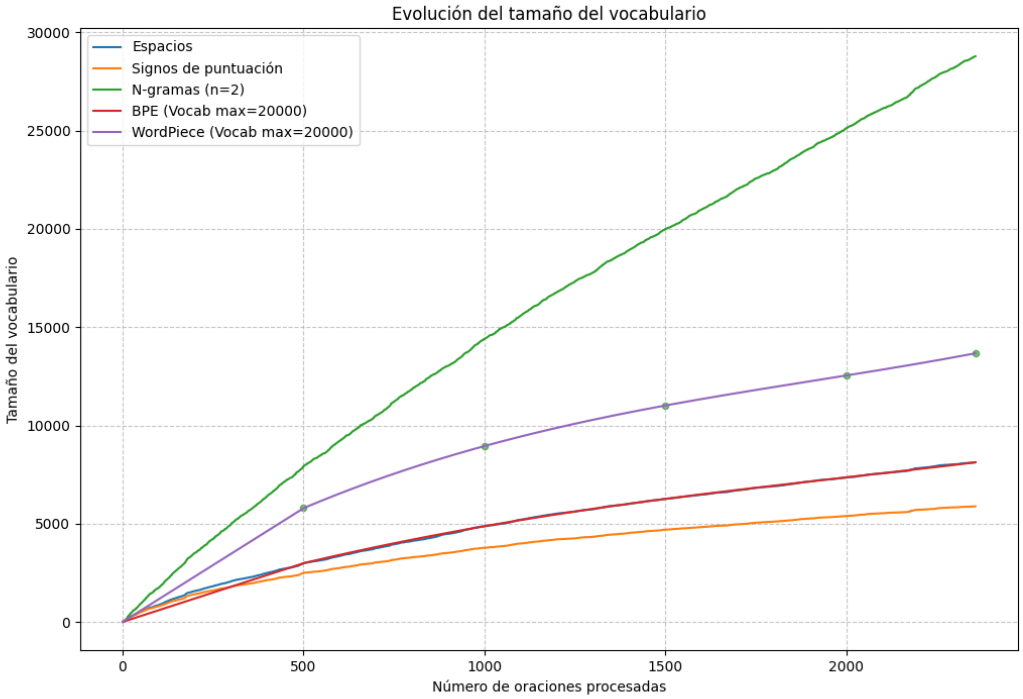

Figure 1: Comparación de vocabularios de los métodos, conforme se procesan las oraciones

El tiempo de ejecución depende principalmente de los algoritmos más complejos, es decir, BPE y WordPiece. Ya que los más tradicionales tienen implementaciones muy sencillas que no suponen una gran complejidad computacional.

Enfocándonos en el tamaño de vocabulario, podemos apreciar claramente cómo el algoritmo con peor rendimiento en este apartado es **n-grams**, esto se debe a que es menos usual encontrar pares de palabras para poder reutilizar el vocabulario. Entonces, cada n-palabras, estamos añadiendo un nuevo token al vocabulario; evidentemente, a menor n, mayor capacidad de tokenizar tendremos, a coste de un mayor tamaño de vocabulario.

Los algoritmos de **Espacios** y de **Puntuación** obtienen un vocabulario relativamente similar (1000-2000 palabras de diferencia). Esto se debe a que su lógica es prácticamente la misma, únicamente que en el algoritmo de separación por puntuación añadimos unas condiciones extras aparte de los espacios. Pero la gran mayoría de las palabras serán tokenizadas de la misma forma que en el algoritmo de espacios, por ello la mínima separación.

El algoritmo de BPE obtiene un rendimiento prácticamente igual al de espacios. Esto se debe principalmente a que BPE inicialmente tokeniza caracteres individuales y luego fusiona los pares más frecuentes hasta alcanzar el tamaño de vocabulario deseado. Con un vocabulario máximo suficientemente grande (10000 en este caso), BPE tiende a reconstruir palabras completas que serían similares a las obtenidas por tokenización por espacios. Sin embargo, BPE nos otorga una tokenización más flexible y precisa que la de los algoritmos más básicos, con el coste de un mayor tiempo de procesamiento debido a su complejidad computacional.

Por último, el algoritmo de WordPiece nos otorga un punto medio entre el tamaño de n-grams y de BPE/Espacios. Esto se debe a que WordPiece utiliza una estrategia basada en la probabilidad de aparición de subpalabras dentro del corpus, priorizando las combinaciones que maximizan la verosimilitud del corpus de entrenamiento. Este enfoque tiende a generar un vocabulario más grande que BPE pero más eficiente que n-grams, especialmente para corpus con alta variabilidad.

3 Conclusiones

Tras el análisis de los diferentes métodos de tokenización implementados, podemos extraer varias conclusiones importantes:

1. Los métodos tradicionales (espacios y puntuación) ofrecen una implementación sencilla y eficiente, pero carecen de la flexibilidad necesaria para capturar la estructura de las palabras.

2. El método de n-gramas genera vocabularios excesivamente grandes que crecen de forma desproporcionada con el tamaño del corpus, lo que lo hace poco práctico para aplicaciones de procesamiento de lenguaje natural mas complejas.

3. BPE proporciona un equilibrio entre eficiencia y flexibilidad, manteniendo un tamaño de vocabulario controlado mientras captura subpalabras significativas.

4. WordPiece ofrece una tokenización más orientada a la semántica, pero a costa de un vocabulario mayor que BPE.