Procesamiento del lenguaje natural

Práctica 2

Francisco Jesús Najas Martos - 21025923J

Ejercicio 3:

Según la función time(), es más eficiente el Punkt tokenizer, alrededor de unas 2 décimas de segundo. Esto se debe porque al trabajar con muchos textos, la primera tiene que llamar al modelo todas las veces, sin embargo, con la otras está cargado en memoria.

Ejercicio 4:

- TreebankWordTokernizer

```
['Sorry', ',', 'I', 'ca', "n't", 'go', 'to', 'the', 'meeting', '.']
```

- WhitespaceTokenizer

```
['Sorry,', 'I', "can't", 'go', 'to', 'the', 'meeting.']
```

- SpaceTokenizer

```
['Sorry,', 'I', "can't", 'go', 'to', 'the', 'meeting.\n']
```

- WordPunctTokenizer

```
['Sorry', ',', 'I', 'can', "'", 't', 'go', 'to', 'the', 'meeting', '.']
```

En el TreebankWordTokenizer, vemos que coge tokens por palabra, es decir, por ejemplo en ‘can’t’, es una abreviatura de ‘can not’, pero no coge ‘can’ como ‘can’, si no como ‘ca’. En el WhitespaceTokenizer separa por espacios en blanco diferenciando caracteres especiales como ‘\n’, sin embargo en el SpaceTokenizer no. En el WordPunctTokenizer, separa por todo, como en el ‘can’t’, separa incluso el apóstrofe.

Ejercicio 5:

```
['Sorry', 'I', "can't", 'go', 'to', 'the', 'meeting']
```