Integrantes: Fabian Pallares, Daniel Mejia, Laura Mora

Septiembre 6/2022

**Taller 2 NLP.**

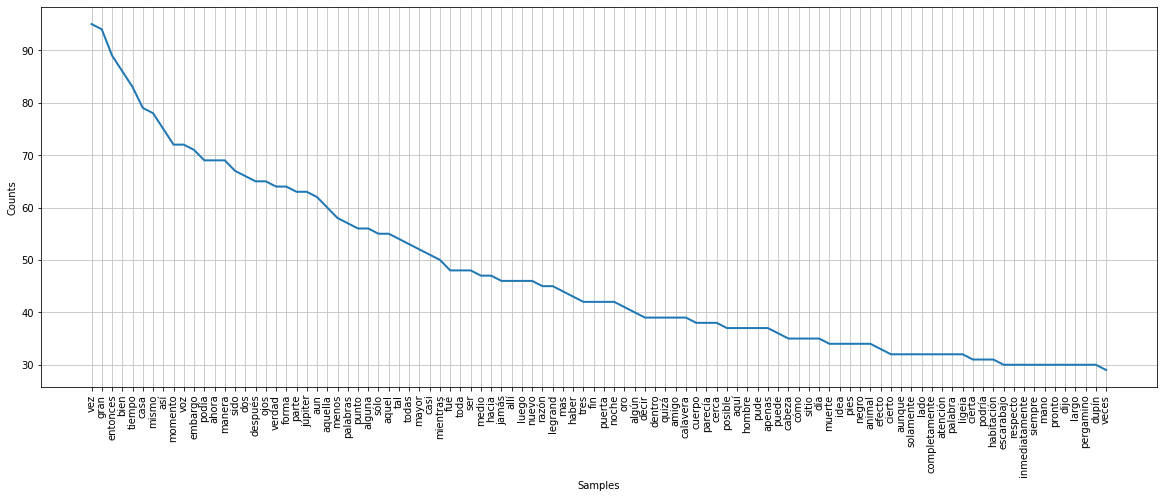
Con el fin de dar solución al taller, se decide usar el libro de “cuentos\_allan\_poe\_esp.txt”, adicionalmente se importa la librería nltk y el componente de PlaintextCorpusReader.

1. Escriba una función que encuentre las 50 palabras más frecuentes en el libro seleccionado que no sean palabras vacías.

Inicialmente se importa desde la librería de nltk lo relacionado con stopwords, posteriormente se aplica una pequeña limpieza de datos, donde se realiza la toma de las palabras vacías (stopwords) en español y se agregan a estas "i", "l", "ugh", "crd", "cr", "eh", "sino", "tan", "si", "vos"; a continuación se guarda el libro sin palabras vacías y con todo su contenido en minúscula.

Con el fin de obtener las palabras más frecuentes, se utiliza la función de nltk FreqDist, la cual nos da como resultado la frecuencia de todas las palabras, para obtener solo las 50 más comunes se imprime del objeto, donde quedan todas las palabras con su frecuencia, el método most\_common, enviando como parámetro 50

Finalmente se importan la librería matplotlib.pyplot, para ver de forma gráfica las palabras más comunes.



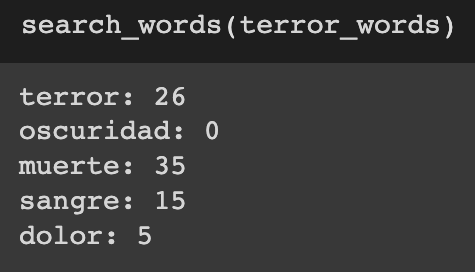
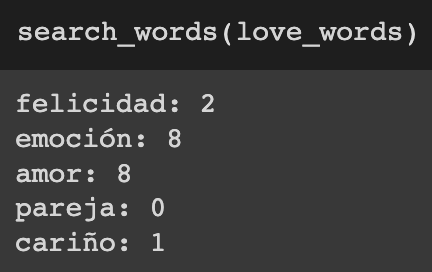
1. Escribir una función para imprimir los 50 bigramas más frecuentes (pares de palabras adyacentes) en el libro seleccionado, omitiendo los bigramas que contienen palabras vacías.

Teniendo en cuenta que para el punto anterior se realizó la limpieza de los datos, con el fin de dar solución a este punto se utilizó la función de nltk FreqDist, a la cual se le paso como parámetro la función que retorna los bigramas del libro seleccionado ya limpio; para obtener los 50 más frecuentes se usa la función most\_common y como parámetro se envía 50.

1. Escribir una función para crear una tabla de frecuencias de palabras que signifiquen algo en el contexto, como la dada en el ejemplo de los modales. Elija sus propias palabras y trate de encontrar palabras cuya presencia (o ausencia) sea típica en el libro seleccionado.

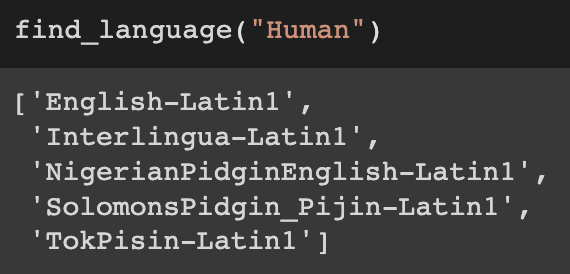
Con el fin de encontrar la frecuencia de las palabras seleccionadas, se crea la función “search\_words”, la cual recibe como parámetro una lista de palabras, donde a través del método de nltk FreqDist encuentra la frecuencia de las palabras del libro, y posteriormente se imprimen las palabras de la lista parámetro con su respectiva frecuencia.

Para ejemplificar se crearon dos listas, una con palabras que el equipo consideraba podían aparecer frecuentemente debido a la temática del libro (terror\_words), y otra donde se consideraba que aparecerían poco (love\_words), los resultados se muestran a continuación.

1. Defina una función find\_language () que toma una cadena como su argumento, y devuelve una lista de idiomas que tienen esa cadena como una palabra. Utilice el corpus udhr y limite sus búsquedas a archivos en la codificación Latin-1. Probar con diferentes tweets en otros idiomas.

Inicialmente se inicializa el tokenizer con la función WhitespaceTokenizer(), posteriormente se define la función, donde se define el corpus para la codificación de Latin1, se pasa a tokenizar el parámetro para el caso donde sea una frase, y finalmente se va buscando en los lenguajes las diferentes palabras o la palabra parámetro, todo esto se guarda en un arreglo y este se retorna. A continuación se muestra un ejemplo del resultado con la palabra “Human”.



1. ¿Cómo se generan hiperónimos e hipónimos en wordnet? Dar un ejemplo con Python y Nltk.

Para generar los hiperónimos y los hipónimos en Word net, inicialmente se debe importar esta librería del corpus de nltk y se crea una variable llamada cat, a la que se le asignan como resultado las palabras más cercanas obtenidas por la función de wordnet synet.

Con el fin de obtener los hipónimos se usa la función hyponyms, la cual nos retorna dos tipos de datos, finalmente se obtienen los lemas de los tipos de dato.

Con el fin de obtener los hiperónimos se utiliza la función hypernyms, y obtenemos todas las palabras relacionadas a gato.

1. Crear un programa que divida en oraciones los 50 primeros parágrafos presentes en el libro seleccionado, haciendo uso de la función “sent\_tokenize”, y que calcule el número medio de oraciones por comentario.

Inicialmente se selecciona como raw el libro mencionado anteriormente (cuentos\_allan\_poe\_esp.txt), se obtienen los párrafos usando el BlanklineTokenizer() en el libro, posteriormente se pasa a iterar en los párrafos obteniendo la cantidad de oraciones, usando el sent\_tokenize en el párrafo correspondiente a la iteración, tras obtener este dato se suma a la cuenta general de oraciones. Finalmente tras realizar todas la iteraciones, se divide la cantidad general de oraciones entre 50 para obtener el promedio.

1. Dividir en tokens el tweet referencia empleando los siguientes tokenizadores: “TreebankWordTokenizer”, “WhitespaceTokenizer”, “SpaceTokenizer” y “WordPunctTokenizer”. ¿Qué diferencias se observan en la salida producida por cada uno de ellos?

Tras utilizar los diferentes tokenizadores en el tweet (La sabiduria popular enseña.\n"Las adulaciones son como los perfumes: logran deleitar, pero no se deben tragar") se destaca que de las salidas, para este caso específico, el whitespace tokenizer da un mejor resultado al solo incluir las palabras. Los otros tokenizadores tienen problemas con las comillas, y el SpaceTokenizer no identifica el salto de línea.

1. Crear un tokenizador basado en expresiones regulares que extraiga sólo las palabras presentes en el texto, es decir, que no devuelva como salida los signos de puntuación ni los tabuladores/saltos de línea... Además, el tokenizador no deberá́ separar las contracciones si existen. Probar con el tweet de referencia.

Para crear el tokenizador se importa de nltk la función regexp\_tokenize, la cual tiene como parámetro el texto a tokenizar, en este caso el tweet y los parámetros que para este caso son "\s|[\.,;:\"']"

1. Del libro seleccionado deben crear un programa en el que muestre la siguiente información:

1. Total de oraciones.  
2. Total de tokens.  
3. Total de palabras vacías.  
4. Número medio de oraciones por comentario.  
5. Número medio de tokens por oración.  
6. 5 raíces más comunes (sin tener en cuenta las palabras vacías).

Se crea una función “info\_book”, la cual recibe como parámetro el libro donde a través del sent\_tokenize se obtiene la cantidad de oraciones, a través de la función words se obtienen la cantidad de palabras y la cantidad de palabras vacías, usando la función FreqDist junto con stem y most\_common las 5 raíces más comunes, adicionalmente se realizan los cálculos para el número medio de oraciones por comentario y el número medio de tokens por oración.