Clase 10

plot() Con esta función de las distribuciones de frecuencia, es posible obtener una gráfica con los datos de frecuencia de cada una de las palabras de un texto. Pueden dar un parámetro opcional numérico para determinar el número de palabras que se van a tomar para hacer la gráfica. Las palabras se ordenan por frecuencia.

cumulative=True Si desean que su gráfica sea acumulativa, pueden dar este parámetro ocional.

hapaxes() Los *hapaxes* son las palabras que ocurren una sola vez en un texto. Con esta función, se obtiene una lista con las palabras que tienen una única aparición en el texto.

nltk.corpus.stopwords.words("spanish") Con el **`download()`** de NLTK pueden conseguir un módulo de *stopwords* (en la sección de Corpora). Para usarlas, pueden obtener la lista que tiene NLTK para el español con esta función.

TF

Frecuencia de término:

- Como su nombre lo indica, este valor es la frecuencia con la que ocurre el término en un documento. En el último ejercicio creamos diccionarios en los que es sencillo obtener este valor para cada par documento-palabra.
- Para los que se quieran ver más profesional y no dar preferencia a los documentos largos, pueden agregar una operación extra al cálculo del TF: dividirlo entre la frecuencia máxima del documento.

$$TF(d,t) = f(d,t) \qquad TF(d,t) = \frac{f(d,t)}{max\{f(d,x) : x \in d\}}$$

IDF

Frecuencia inversa de documento:

- Este valor no varía entre documentos, solo entre palabras. Sin embargo, para calcularla es necesario saber cuántos documentos hay en nuestro corpus (indicados más abajo con la letra D) y en cuántos aparece la palabra.
- El *Idf* se define como el logaritmo de la razón que existe entre la cantidad total de documentos del corpus entre la cantidad que contienen la palabra.
- Para los que se quieren ver más profesionales, pueden agregar la suma de uno a esa división para evitar división entre cero.

$$IDF(t) = \log \frac{D}{\{d \in D : t \in d\}} \qquad IDF(t) = \log \frac{D}{1 + \{d \in D : t \in d\}}$$

TF-IDF

$$TF(d,t) = f(d,t) \qquad TF(d,t) = \frac{f(d,t)}{max\{f(d,x) : x \in d\}}$$

$$IDF(t) = \log \frac{D}{\{d \in D : t \in d\}} \qquad IDF(t) = \log \frac{D}{1 + \{d \in D : t \in d\}}$$

$$TFIDF(d,t) = TF(d,t) \cdot IDF(t)$$

```
# NOTA: para calcular logaritmos:

import math

logaritmo=math.log(x)
```

Ejercicio 19

De verdad les recomiendo que partan del ejercicio anterior

- Programen una función que calcule la Tf-idf dando como parámetros de entrada una palabra y el nombre de un documento de su corpus.

NLTK

Colocaciones

- Las colocaciones son secuencias de palabras que ocurren juntas de una forma inusualmente frecuente.
- NLTK tiene una función particular para obtener las colocaciones de un texto.

```
# Aquí ya tenemos tokens y tokens_limpios.

texto_nltk=nltk.Text(tokens)
texto_limpio_nltk=nltk.Text(tokens_limpios)

texto_nltk.collocations()
print()
texto_limpio_nltk.collocations()
```

- Se puede usar tanto el texto completo como el "limpio", ambos dan información que puede ser útil, como "rubro citado" , "S.A. C.V." o "Ciudad México". Estos ejemplos aparecen sólo en uno de los casos.

NLTK

Etiquetado PoS

- Un recurso muy útil para el PLN es el etiquetado PoS (de *Part of Speech*).
- Con esta herramienta, la computadora hace su mejor esfuerzo por asignarle a cada palabra de un texto la parte de la oración que le corresponde (sustantivo, verbo, adjetivo, determinante, etc.).
- NLTK cuenta con un etiquetador PoS. Desafortunadamente, es únicamente para inglés.
- Afortunadamente, también tiene acceso a un etiquetador externo (de Stanford) que sí es capaz de manejar el español.
- Esta herramienta no la tiene por defecto, y al ser externa tampoco se descarga desde el sistema de descarga que hemos estado usando hasta ahora.
- `https://nlp.stanford.edu/software/tagger.shtml`

NLTK

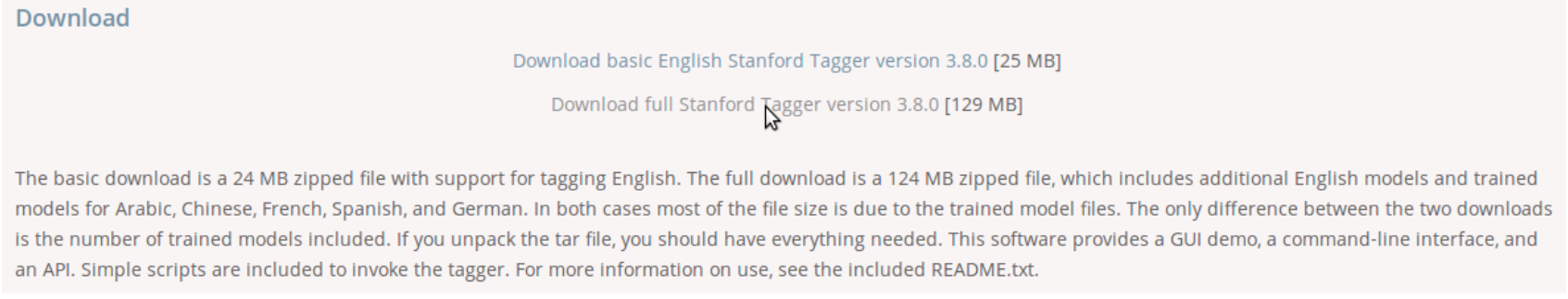
Etiquetado PoS

- En la página de Stanford podemos encontrar toda la información sobre el etiquetador y ligas de descarga.

NLTK

Etiquetado PoS

- La descarga básica de su etiquetador también funciona sólo con inglés, así que necesitamos la descarga completa, es la segunda liga.
- Esto va a descargar un archivo comprimido, es importante que lo descompriman y que recuerden dónde lo colocan. Recuerden la forma de obtener rutas completas, será necesario para usarlo.



- Una vez que hayan descargado y descomprimido el zip, hay dos archivos que deben localizar.
- El primero está directamente dentro de la carpeta que acaban de descomprimir, se llama **`stanford-postagger.jar`**.
- El segundo, está dentro de la carpeta **`models`**, y su nombre es **`spanish.tagger`** sin nada más.
- Para ambos archivos van a necesitar la ruta completa para usarlos con Python y NLTK.

- La función **StanfordPOSTagger()** recibe de parámetros los archivos que obtuvimos, primero el **spanish.tagger**, luego el **stanford-postagger.jar**.

```
from nltk.tag import StanfordPOSTagger

# Aquí obtenemos la lista de tokens en "tokens"

tagger="C:\\Users\\user\\Downloads\\...\\spanish.tagger"
jar="C:\\Users\\user\\Downloads\\...\\stanford-postagger.jar"

etiquetador=StanfordPOSTagger(tagger,jar)
etiquetas=etiquetador.tag(tokens)

for etiqueta in etiquetas:
        print(etiqueta)
```

- Con esa información, el **`StanfordPOSTagger()`** crea un etiquetador que luego podemos utilizar para etiquetar nuestra lista de tokens. En este caso, es mejor utilizar la lista que no está "limpia".

```
from nltk.tag import StanfordPOSTagger

# Aquí obtenemos la lista de tokens en "tokens"

tagger="C:\\Users\\user\\Downloads\\...\\spanish.tagger"
jar="C:\\Users\\user\\Downloads\\...\\stanford-postagger.jar"

etiquetador=StanfordPOSTagger(tagger,jar)
etiquetas=etiquetador.tag(tokens)

for etiqueta in etiquetas:
        print(etiqueta)
```

NLTK

Etiquetado PoS

- ¿Y esas etiquetas qué?
- Cada etiqueta tiene un significado, en la página `https://nlp.stanford.edu/software/spanish-faq.shtml#tagset` pueden encontrar las etiquetas con sus significados y ejemplos.

6. What POS tag set does the parser use?

We use a simplified version of the tagset used in the AnCora 3.0 corpus / DEFT Spanish Treebank. The default AnCora tagset has hundreds of different extremely precise tags. This may be useful for some linguistic applications, but did not bode well for even a state-of-the-art part-of-speech tagger. We reduced the tagset to *85 tags*, a more manageable size that still allows for a useful amount of precision.

The tags are designed to remain compatible with the EAGLES standard. In our tags, we simply null out most of the fields (using a label `0`) that are not relevant for our purposes. The resulting compressed tagset is listed below.

Tag	Description	Example(s)
Adjectives		
ao0000	Adjective (ordinal)	*primera, segundo, últimos*
aq0000	Adjective (descriptive)	*populares, elegido, emocionada, andaluz*
Conjunctions		
cc	Conjunction (coordinating)	*y, o, pero*
cs	Conjunction (subordinating)	*que, como, mientras*
Determiners		
da0000	Article (definite)	*el, la, los, las*
dd0000	Demonstrative	*este, esta, esos*
de0000	"Exclamative" (TODO)	*qué (¡Qué pobre!)*
di0000	Article (indefinite)	*un, muchos, todos, otros*
dn0000	Numeral	*tres, doscientas*
do0000	Numeral (ordinal)	*el **65** aniversario*
dp0000	Possessive	*sus, mi*
dt0000	Interrogative	*cuántos, qué, cuál*

FIN

Extras

Ejercicio

- Elijan un documento, uno de esos que tienen en su diccionario con distribuciones de frecuencia.
- Para cada término de su documento, escriban en un archivo TXT una línea con el término, seguido de una coma, seguido de su frecuencia (o aún mejor, el tf-idf).

Ejercicio

- Ahora, intenten abrir ese archivo que acaban de crear con Excel (o con Calc de Libreoffice, o con Gnumeric, etc.)

CSV

- Lo que acaban de hacer es en realidad un CSV (comma separated values), por lo regular estos archivos se usan con extensión *csv* (no *txt*) y como acaban de ver, es el equivalente en texto para las hojas de cálculo.
- Python tiene un módulo especial para facilitar el manejo de estos archivos.

```
import csv
```

CSV

- El módulo **csv** proporciona la su función **writer()**, que recibe como parámetro un archivo abierto. Opcionalmente pueden usar los parámetros **delimiter=** y **quotechar=**.
- La variable que devuelve **writer()** contiene funciones para escribir en el archivo. La función **writerow()** recibe una lista, y escribe el contenido en un renglón del csv. Las comas y los caracteres especiales los maneja automáticamente.

```
import csv

# Por aquí se declaran los diccionarios.

with open("frecuencias.csv","w") as archivo_csv:
    escritor_csv = csv.writer(archivo_csv)
    for termino in diccionario_frecuencias["texto1.txt"]:
        frecuencia=diccionario_frecuencias["texto1.txt"][termino]
        escritor_csv.writerow([termino,frecuencia])
```

CSV

- Y como es de esperar, también tiene lector de csv, para eso se usa la función **`reader()`**. De igual manera recibe un archivo, pero abierto en modo de lectura.
- El lector no necesita más funciones para leer. Lo pueden recorrer como una lista, y les va a regresar, en otra lista, el contenido de los elementos de un renglón del csv. Pueden considerar al lector, como una lista de listas.

```
import csv

with open("frecuencias.csv","r") as archivo_csv:
        lector_csv = csv.reader(archivo_csv)
        for linea in lector_csv:
                print(linea)
```

Expresiones Regulares

Cuando compilan una expresión regular, es posible dar opciones especiales de cómo se de deben comportar ciertos símbolo especiales, aquí les muestro casos interesantes:

re.MULTILINE Esta opción cambia el comportamiento del acento circunflejo (^) ya que en lugar de coincidir únicamente con el inicio del texto, va a coincidir con el inicio de cada línea.

re.DOTALL Esta opción cambia el comportamiento del punto, y permite que coincida también con saltos de línea.

Algo importante que deben saber, es que solo pueden usar una opción al compilar sus expresiones regulares.

Expresiones Regulares

Algunos se podrán preguntar para qué quieren cambiar el comportamiento del acento si pueden leer línea por línea y conseguir lo mismo. Así que les traigo un ejemplo interesante:

```
import re

with open("La Tragedia Del Rey Ricardo.txt") as archivo:
        texto=archivo.read()

expresion_texto="^[A-Z]\w+:(.*[\n.])+?(?=[A-Z]\w+:)"
expresion=re.compile(expresion_texto,re.MULTILINE)
resultados=expresion.finditer(texto)
for resultado in resultados:
        print(resultado.group(0))
        print("========================================")
```

Expresiones Regulares

Y además hago uso de todavía un recurso más, la instrucción especial (?=). Esta instrucción hace que lo que se encuentra entre paréntesis se busca, pero no coincide.

```
import re

with open("La Tragedia Del Rey Ricardo.txt") as archivo:
        texto=archivo.read()

expresion_texto="^[A-Z]\w+:(.*[\n.])+?(?=[A-Z]\w+:)"
expresion=re.compile(expresion_texto,re.MULTILINE)
resultados=expresion.finditer(texto)
for resultado in resultados:
        print(resultado.group(0))
        print("========================================")
```