Clase 9

similar() Esta es otra de las funciones que brinda NLTK con su `Text`. Obtiene las palabras similares (dentro del `Text` a la palabra que se da como parámetro. Para encontrar dichas palabras similares busca aquellas palabras que tengan contextos parecidos a la palabra dada.

common_contexts() Una vez que han buscado palabras similares con la función `similar()` pueden buscar los contextos comunes que tuvo su palabra con las palabras similares proporcionadas. Esta función recibe una lista con dos palabras y muestra en pantalla los contextos comunes que tienen en el texto.

Repaso
NLTK

dispersion_plot() Con esta función de los `Text` de NLTK ustedes pueden graficar una gráfica de dispersión, es decir, una gráfica en la que se muestra el uso de una lista de palabras a lo largo del texto. La función recibe como parámetro la lista de palabras que quieren graficar.

FreqDist() Con esta función pueden obtener una distribución de frecuencias a partir de un `Text` de NLTK. La distribución de frecuencias mantiene un conteo de todas las palabras de su texto. El resultado devuelto, además, se comporta como un diccionario, ya que pueden acceder a la frecuencia de la palbra dándola como índice de la distribución.

Repaso

most_common() Esta es una función de las variables devueltas por **`FreqDist()`**. La función devuelve una lista con pares de valores palabra-frecuencia ordenada de mayor a menor frecuencia. Pueden dar un parámetro opcional numérico para obtener solo los primeros *n* valores más frecuentes. Para cada uno de estos pares de valores puede utilizar los corchetes con índice para acceder a los valores finales; es decir *mas_comunes*[0][0] les dará la palabra más común, y *mas_comunes*[0][1] la frecuencia que tiene.

diccionario={} Los diccionarios son muy parecidos a las listas. La principal diferencia radica en que se utilizan palabras (llaves) en lugar de índices para acceder al contenido. Para definirlo se usan llaves en lugar de corchetes.

Ejercicio 18

- Obtengan una lista con los nombres de todos los documentos que estamos manejando.
- Obtengan también una lista (o un conjunto) con el vocabulario de TODO su corpus (es decir, los tokens sin repetición, también se les llama tipo).

Ejercicio 18

- Para cada documento de su lista, coloquen una entrada de diccionario cuyo contenido sea su distribución de frecuencias.
- Para cada tipo de su vocabulario, coloquen una entrada de diccionario cuyo contenido sea una lista con los nombres de los documentos en los que aparece.

NLTK

Gráfica de frecuencias

- Muchas veces nos interesa observar de una manera visual cómo cambia la frecuencia de las palabras.
- Las distribuciones de NLTK tienen opciones para obtener gráficas.

```
# A esta altura ya tenemos la lista de tokens en "tokens".

texto_nltk=nltk.Text(tokens)
distribucion=nltk.FreqDist(texto_nltk)

distribucion.plot()
```

NLTK

Gráfica de frecuencias

- Seguramente su gráfica muestra demasiadas palabras como para que se entienda claramente el valor de cada una.
- Se puede disminuir el número de palabras que se muestran, pueden dar el número que desean como parámetro a la función.

```
# A esta altura ya tenemos la lista de tokens en "tokens".

texto_nltk=nltk.Text(tokens)
distribucion=nltk.FreqDist(texto_nltk)

distribucion.plot(40)
```

NLTK

Gráfica de frecuencias

- Una nota: es probable que su gráfica muestre las palabras cortadas en el eje horizontal.
- Esto se puede corregir si agregan las siguientes líneas en algún punto ANTERIOR a usar la gráfica:

```
from matplotlib import rcParams
rcParams.update({"figure.autolayout": True})
```

- No se preocupen mucho por este código, son líneas especiales de configuración de las gráficas.

NLTK

Hapaxes

- Las distribuciones de NLTK también cuentan con una función que regresa los *hapaxes* de un texto. Para ello usamos la función **`hapaxes()`** de la distribución, y obtenemos una lista con las palabras.
- Un *hapax* es una palabra que aparece únicamente una vez en el texto.

```
# A esta altura ya tenemos la lista de tokens en "tokens"

texto_nltk=nltk.Text(tokens)
distribucion=nltk.FreqDist(texto_nltk)
hapaxes=distribucion.hapaxes()

for hapax in hapaxes:
        print(hapax)
```

NLTK

Gráfica acumulativa de frecuencias

- La gráfica de la distribución tiene un parámetro opcional para transformar la gráfica en una acumulativa.
- Notarán que se usa la palabra especial `True`, que es, como su nombre lo dice, el valor "verdadero". Este valor se utiliza para cuando se quiere valores binarios (como cuando se revisa un `if`), también existe el valor `False`

```
distribucion.plot(cumulative=True)
distribucion.plot(40,cumulative=True)
```

NLTK

Gráfica acumulativa de frecuencias

- Podemos comparar el total de palabras con el aumento acumulativo de la gráfica.
- Observen como las primeras 40 palabras más usadas cuentan por más de la mitad del número total de palabras (que son más de 300).

```
# A esta altura ya tenemos la lista de tokens en "tokens"

tokens_conjunto=set(tokens)
palabras_totales=len(tokens)
palabras_diferentes=len(tokens_conjunto)

texto_nltk=nltk.Text(tokens)
distribucion=nltk.FreqDist(texto_nltk)

print(palabras_totales)
print(palabras_diferentes)

distribucion.plot(cumulative=True)
distribucion.plot(40,cumulative=True)
```

NLTK

Palabras funcionales

- Algo que pueden notar de las gráficas de distribuciones, y que sin duda ya esperaban, es que las palabras más frecuentes, son palabras como "el", "de", "la", etc. A este tipo de palabras, se le llaman palabras funcionales.
- En todos los archivos que analicen siempre estas palabras ocuparán los lugares mas frecuentes. Es por esta razón que buscar las palabras más frecuentes en realidad no da información sobre el contenido de un archivo.
- Afortunadamente, estas palabras son muy conocidas y estudiadas, y ya que todo el mundo por lo regular quiere quitarlas, no es difícil encontrar listas enumerándolas.
- NLTK cuenta con su propia lista de estas palabras, también llamadas *stopwords*.

NLTK

Palabras funcionales

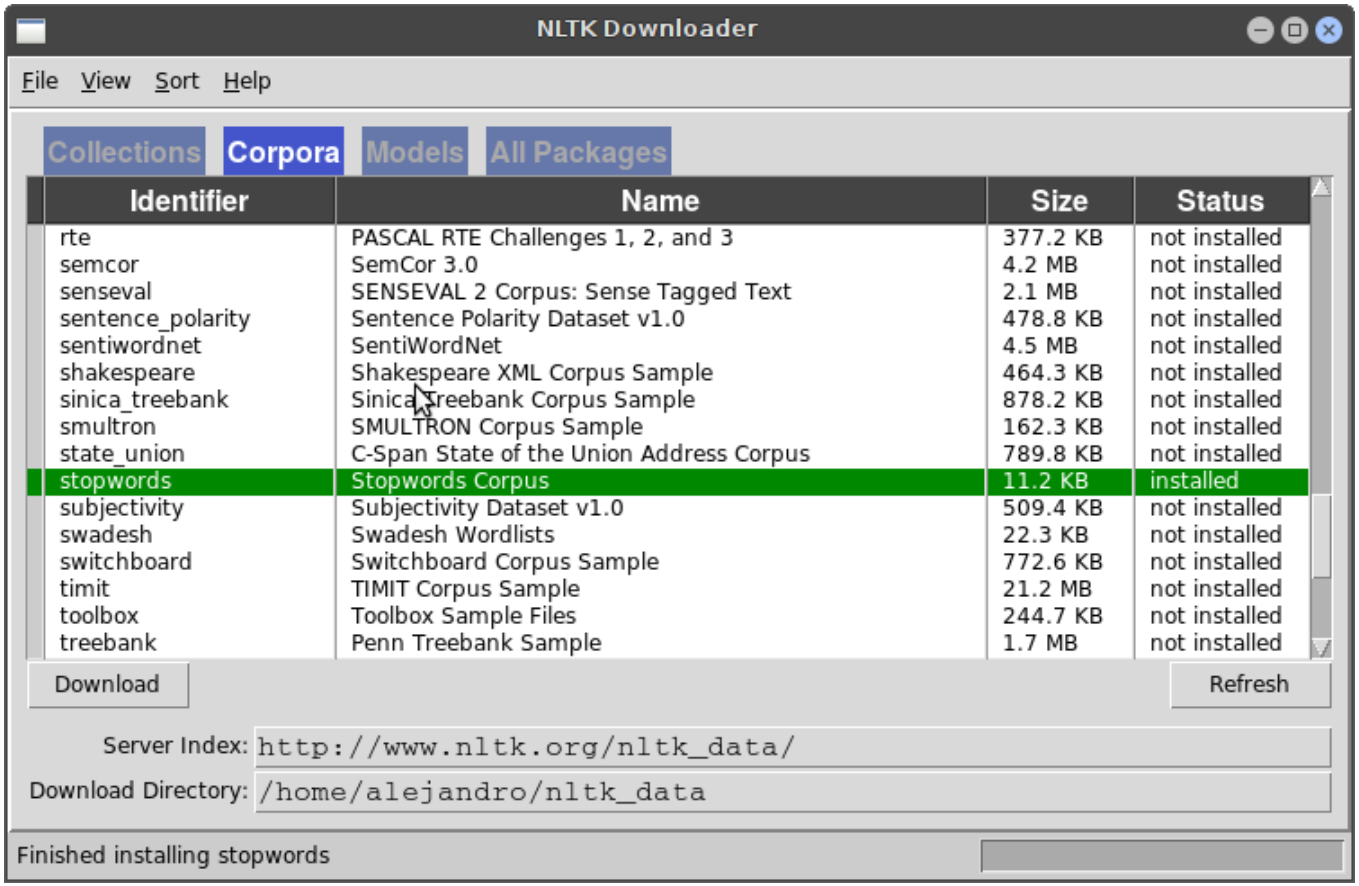
- Para obtenerla, hay que descargarla.
- Recuerden que NLTK tiene su propio sistema de descarga que se usa desde Python.

```
import nltk

nltk.download()
```

NLTK

Palabras funcionales



- Una vez que aparezca la pantalla de descargas vayan a la sección de **Corpora** y descarguen **stopwords**, que como verán, son listas de palabras funcionales en varios idiomas.

NLTK

Palabras funcionales

- Y ahora que ya descargamos las listas, podemos cerrar esa ventana.
- A partir de ahora podemos usar las listas de *stopwords* de NLTK. Veamos que palabras tiene para el español.

```
import nltk

palabras_funcionales=nltk.corpus.stopwords.words("spanish")

for palabras_funcional in palabras_funcionales:
        print(palabras_funcional)
```

NLTK

Palabras funcionales

```
import nltk

carpeta_nombre="Documentos\\"
archivo_nombre="P_IFT_290216_73_Acc.txt"

with open(carpeta_nombre+archivo_nombre,"r") as archivo:
        texto=archivo.read()

palabras_funcionales=nltk.corpus.stopwords.words("spanish")

tokens=nltk.word_tokenize(texto,"spanish")

tokens_limpios=[]
for token in tokens:
            if token not in palabras_funcionales:
                    tokens_limpios.append(token)

print(len(tokens))
print(len(tokens_limpios))
```

NLTK

Palabras funcionales

- Podemos también obtener las gráficas de los nuevos datos sin palabras funcionales.
- Por cierto, quizá notaron en el ejemplo anterior que mostramos en pantalla (con **`print()`**) el resultado directo de una función, sin asignarlo antes a una variable. En Python podemos hacer poner una cadena de funciones y usar los resultados inmediatamente, incluso podemos hacer algo tan drástico como el siguiente ejemplo, pero por norma general, no se recomienda, se vuelve poco legible y no se pueden recuperar los datos intermedios.

```
# Después del ejemplo anterior...

nltk.FreqDist(nltk.Text(tokens_limpios)).plot(40)
```

- Como pueden observar, aún hay elementos muy frecuentes que sería útil quitar, como los sinos de puntuación.
- Usaré este punto para explicarles un detalle sobre el texto que no había mencionado antes. Y es que se puede comportar un poco como una lista de letras.
- El ejemplo siguiente es una solución ingenua para quitar los signos de puntuación, es decir, es muy simple, pero no es la solución ideal ya que esto quitará más cosas además de los signos de puntuación, pero depende del objetivo final, puede que la pérdida no sea importante.

NLTK

Palabras funcionales

```
import nltk

carpeta_nombre="Documentos\\"
archivo_nombre="P_IFT_290216_73_Acc.txt"

with open(carpeta_nombre+archivo_nombre,"r") as archivo:
        texto=archivo.read()

palabras_funcionales=nltk.corpus.stopwords.words("spanish")

tokens=nltk.word_tokenize(texto,"spanish")

tokens_limpios=[]
for token in tokens:
        if token not in palabras_funcionales:
                if len(token) > 1:
                        tokens_limpios.append(token)

nltk.FreqDist(nltk.Text(tokens_limpios)).plot(40)
```

NLTK

Palabras funcionales

- Del ejemplo anterior, podemos ver como las palabras "limpias" más comunes, ya nos dan información importante sobre un texto.
- Por cierto, para ustedes que no tienen que ahorrar espacio, recuerden que esto no es recomendable:

```
nltk.FreqDist(nltk.Text(tokens_limpios)).plot(40)
```

- Es mejor esto:

```
texto_limpio_nltk=nltk.Text(tokens_limpios)
distribucion_limpia=nltk.FreqDist(texto_limpio_nltk)

distribucion_limpia.plot(40)
```

TF-IDF

- Para una métrica más confiable de las palabras relevantes de un texto, podemos utilizar el Tf-idf.
- La idea de esta medida numérica es que la importancia de una palabra es proporcional al número de veces que una palabra aparece en un documento.
- Pero al mismo tiempo, una palabra es más importante entre MENOS aparezca en OTROS documentos.
- Esta medida es una herramienta básica en la tarea de recuperación de información (la obtención de documentos a partir de una consulta).

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.