

Actividad NLP - 1.0

October 11, 2023

1 Actividad NLP - 1.0 - Creación de Diccionario

1.0.1 Musel Tabares

1.0.2 A00830710

1.1 Contador de palabras

```
[ ]: #Importamos librerias
import matplotlib.pyplot as plt
import string
import nltk.metrics.distance
import numpy as np
```

```
[ ]: #Funciones auxiliares para leer archivo y guardarlo en un diccionario
def list_from_file(filename):
    myfile = open(filename, 'r')
    data = myfile.read().split()
    col = []
    for word in data:
        col.append(word)
    return col

def myhist(col):
    hist = {}
    for word in col:
        word = word.lower()
        word = word.strip(string.punctuation + string.whitespace)
        hist[word] = hist.get(word, 0)+1
    return hist
```

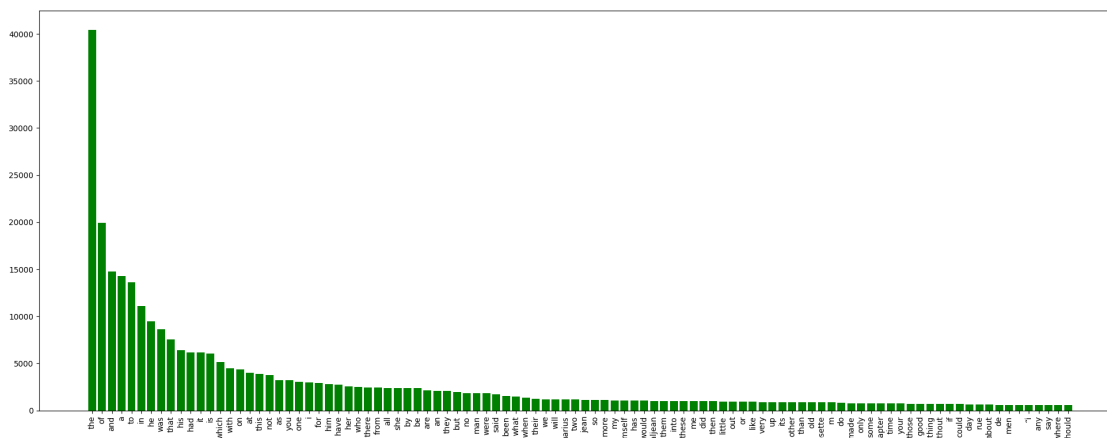
```
[ ]: #leer archivo y guardarlo en un diccionario
col = list_from_file('les_miserables.txt')
colf = myhist(col)
```

```
[ ]: #ordenamos el diccionario por frecuencia
colf_sorted = dict(sorted(colf.items(), key=lambda item: item[1], reverse=True))
```

```
[ ]: #seleccionamos los primeros 100 elementos
top_100 = {k: colf_sorted[k] for k in list(colf_sorted)[:100]}
```

1.1.1 Histograma de las 100 palabras mas frecuentes

```
[ ]: #graficamos
plt.figure(figsize=(20,8))
plt.bar(list(top_100.keys()), top_100.values(), color='g')
plt.xticks(rotation='vertical')
plt.tight_layout()
```

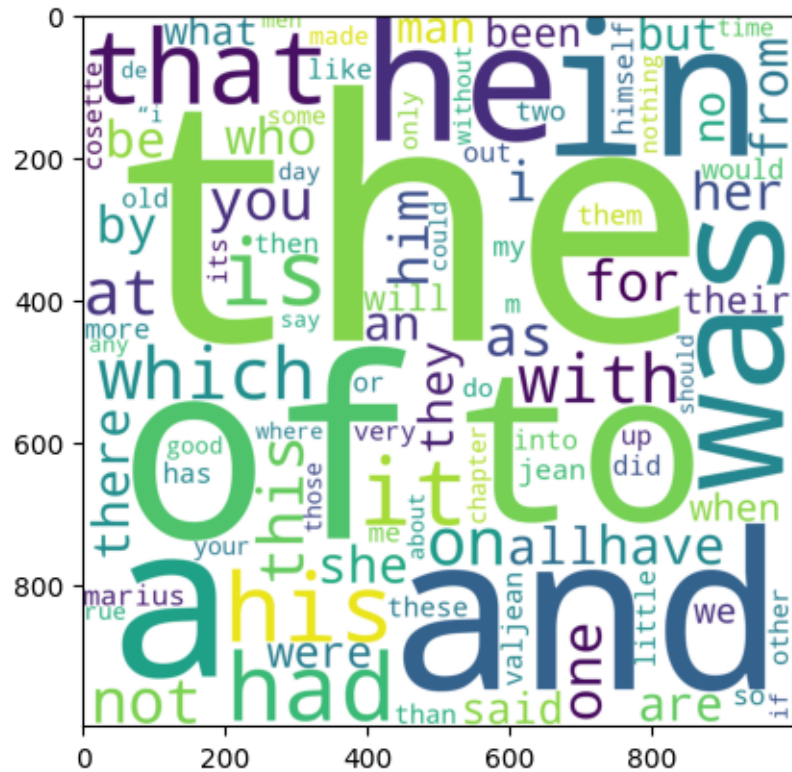


1.1.2 Nube de palabras de las 100 mas frecuentes

```
[ ]: from PIL import Image
import matplotlib.pyplot as plt
from wordcloud import WordCloud

wc = WordCloud(background_color="white",width=1000,height=1000,
    ↪max_words=100,relative_scaling=0.5,normlize_plurals=False).
    ↪generate_from_frequencies(colf)
plt.imshow(wc)
```

```
[ ]: <matplotlib.image.AxesImage at 0x20a5e383460>
```



1.2 Bonus: (opcional)

1.2.1 Creación de un diccionario de palabras.

```
[ ]: diccionario = list(colf_sorted.keys())
```

1.2.2 Realizar una corrección del texto con respecto al diccionario usando cálculo de distancia de “strings”.

```
[ ]: def correccion(s1):
    distancias = [] #array que guardara cada una de las distancia, por cada
    ↪palabra
    #calculamos distancia de levenshtein para cada palabra dentro del
    ↪diccionario
    for i in range(len(diccionario)):
        s2 = diccionario[i] # palabra del diccionario actual
        distance = nltk.edit_distance(s1, s2, substitution_cost=1,
    ↪transpositions=False) # calculo de distancia
        distancias.append(distance) # guardamos distancia
        index_min = np.argmin(distancias) # nos quedamos con el indice de la
    ↪palabra con menor distancia
    return diccionario[index_min],distancias[index_min]
```

```
[ ]: palabra = "caprishious"
palabra_corregida, distancia_edicion = correccion(palabra)
print(f'''
Palabra original: {palabra}
La correccion de su palabra es: {palabra_corregida}
Distancia de levenshtein de: {distancia_edicion}
''')
```

Palabra original: caprishious
 La correccion de su palabra es: capricious
 Distancia de levenshtein de: 2