UNIVERSIDAD
POLITÉCNICA
DE MADRID

Ingeniería Linguística

Práctica 2

Fernando Cortés Sancho
Jérémy Angora

Master Universitario en Inteligencia Artificial

23 de diciembre de 2020

Índice

Índice de figuras

1. Introducción

Este documento tiene como finalidad explicar paso a paso la construcción de un clasificador documental con glosario, que clasificará documentos en 3 tipos distintos de clases: deportes, política y salud. Estas clases, a parte de ser las recomendadas por el enunciado son lo suficientemente distintas como para que se pueda distinguir a cual de ellas pertenece un documento.

Para ello se han extraído documentos de la prensa española manualmente. Se han ido buscando noticias en distintos medios que se adecuaran con las 3 clases ya descritas. En total se han obtenido 90 documentos, 30 por clase. 15 serán utilizados para la creación del glosario, como se explicará en la sección 2. Los otros 15 serán utilizados para las pruebas como mas tarde se verá en la sección 3.

Los documentos tienen una longitud mínima de 400 palabras, como se indicó en el enunciado y una longitud máxima indefinida.

Esos documentos han sido procesados y guardados en un archivo con extensión txt, para su mejor tratamiento posterior.

Para la implementación del clasificador se ha utilizado Python como lenguaje, debido a la cantidad de librerías de aprendizaje automático que posee, que hace que la tarea sea mas rápida y simple. En concreto se ha utilizado un *python notebook* que se enseñará en el anexo, para que se pueda ver mas claro cada paso con su explicación correspondiente.

A petición del enunciado, también se ordenan los documentos por similaridad con el glosario. Esto también se verá en la sección 3.

Después, en la sección 4 se expondrán los resultados obtenidos por el clasificador y se valorará su rendimiento.

Por último, en la sección 5 se extraerán conclusiones de lo explicado anteriormente y se propondrán lineas de mejora sobre el clasificador.

2. Creación del glosario

Para la creación del glosario se ha utilizado la mitad de los documentos de cada tema,es decir, 15 por cada uno.

Para su construcción se ha hecho uso de una herramienta online, que, entre otras carácterísticas, posee un extractor terminológico de palabras llamada *Voyant Tools* [1]. Su funcionamiento es muy simple, se pueden cargar varios documentos a la vez, y la herramienta extraerá una lista de los términos mas repetidos, ordenados por su frecuencia de aparición. Además, muestra un gráfico con la tendencia del término a lo largo de los documentos. Esta última función es muy útil, ya que el glosario debía ser representativo de cada tema, y podían aparecer términos en posiciones altas de la lista que en realidad solo aparecieran en un par de documentos, por lo que no serían demasiado representativos.

Por ejemplo, la palabra 'Maradona' aparecía en una posición alta, pero al mirar el gráfico de la tendencia del término se podía ver como aparecía aunque muchas veces pero en un único documento, por lo que no se debía tener en consideración para el glosario.

El panel de esta herramienta puede verse en la Fig. 1 , para la extracción de términos de deportes.

Figura 1: *Voyant Tools* para la extracción de términos del tema deportes.

[1] `https://https://voyant-tools.org//`

Una vez se cargan los documentos deseados a la herramienta antes mencionada, se realizó una búsqueda intensiva a lo largo de la lista para extraer los términos mas relevantes y colocarlos en una lista de candidatos para el glosario final. Cabe destacar que *Voyant Tools* solo extrae unigramas, es decir, términos de una sola palabra, por lo que hubo que hacer una pasada por todos los términos para considerar si eran parte de un n-grama con un número mayor de elementos. Un ejemplo de ellos es el término 'Real Madrid', que aparecía como 'real' y 'madrid', por lo que hubo que juntarlos.

Después de tener una lista de candidatos para cada tema, se procedió al filtrado de la misma. Se quería que el tamaño de los glosarios fuera mas o menos el mismo, para no dar lugar a sesgo. En un principio se tenían 25 términos para el glosario deportes y tan solo 18 de salud. Esto, se considera que puede deberse a que los documentos elegidos para este último tema son muy variados y extraídos de fuentes muy distintas. Al contrario que los documentos de deportes, que son sacados de pocas fuentes y en su mayoría hablan sobre fútbol.

Para solucionarlo se quitaron los términos de la lista de candidatos que menos representativos eran y se añadieron un par de términos más a lista des salud, para que tuvieran un tamaño similar y no dar lugar a sesgo. Este tamaño finalmente fue de 21 términos para cada glosario de cada tema.

Además, se procedió a realizar lo que se llama "variabilidad lingüística", que en este proyecto consiste en variar morfológicamente algunos términos. cambiando por ejemplo el género o número, siempre y cuando tengan representación en los documentos. Eso se hizo para términos como 'artículo', al ver que su variante 'artículos' aparecía también muchas veces en los documentos. Sin embargo, había otros términos que podían sugerir que se deberían añadir sus variantes, pero se podría observar que no estaban suficientemente presente en los documentos. Esto ocurrió por ejemplo con 'equipo'. Se pensó que se podía añadir el término 'equipos', pero no aparecía con demasiada frecuencia en los documentos.

Por otro lado, se tuvo problemas con términos como 'partido' y 'partidos', ya que se encontró que aparecían con mucha frecuencia tanto en el tema de deporte como el tema de política. Lo mismo ocurrió con el término 'pandemia' que aparecía numerosas veces en los documentos de política y de salud. Estos términos, inicialmente se incluyeron en los glosarios de los temas correspondientes, pero se decidió eliminarlos para que no dieran lugar a confusión, ya que se consideraba que esto podía afectar posteriormente al clasificador.

Finalmente, se obtuvieron los glosarios que se pueden ver en la tabla a continuación:

Deporte	**Política**	**Ssalud**
ganar	artículo	asma
derbi	artículos	defectos
Real Madrid	ley	medicamentos
Madrid	PSOE	niños
Atlético de Madrid	PP	salud
Barcelona	economía	personas
olímpicos	reforma	azúcar
final	mayoría	caso
rival	constitución	estudio
balón	debate	riesgo
deportivo	española	medicación
penalti	congreso	embarazado
liga	comisión	crujía
fútbol	país	síntomas
equipo	derecho	dolor
aficionados	presidente	mujeres
futbolistas	propuesta	enfermedad
torneo	rey	problema
temporada	política	factores
minutos	laboral	asociación
gol	gobierno	productos

3. Clasificación de documentos

Una vez tenidos los glosarios, el objetivo es la creación de un clasificador que determine el tema del que trata basándose en el glosario. Además, se debía clasificar internamente los documentos que mas se parecen al glosario, para obtener un *ranking*.

Primero, se cargaron todos los documentos junto con su clase correspondiente. En concreto se utiliza la función *load_files* de *sklearn.datasets*, que carga los documentos y les asigna una clase en función de la carpeta en la que se encuentren. Además, como los documentos estaban guardados en un txt, había que indicar tratara a los documentos con codificación utf-8. Este código es el que se puede ver en la Fig. 2.

```
from sklearn.datasets import load_files
articles = load_files(r"./Documentos/",encoding='utf-8')
X, y = articles.data, articles.target
```

Figura 2: Código de carga de documentos y sus clases.

Para transformar los textos en vectores que pudiera leer entender una máquina, primero había que prepocesarlos. Para ello se eliminaron todos los caracteres especiales, como los signos de puntuación o saltos de linea, y después se pusieron todas las palabras en minúscula, con el fin de que no hubiera lugar a error entre dos tipos de palabras que fueran iguales pero una estuviera en minúsculas y otra en mayúsculas.

Esto se hace de igual manera para cada uno de los términos del glosario.

Para representar los documentos se hace uso del Modelo de Espacio Vectorial (VSM). La representación funciona mediante vectores que contienen una tabla de frecuencia de las palabras del vocabulario construido, en este caso, del glosario. Esta representación es muy útil para decidir que documentos son similares a otros y hay diversos tipos de medidas de similaridad.

En la implementación se guardaron en un txt todos los términos de cada glosario, para luego leerlos en una lista , que será la utilizada como vocabulario para la representación de los textos. ES lista se pasa como parámetro en la función *CountVectorizer* de *sklearn.feature_extraction.text*, de la forma en la que se puede ver en la Fig. 3 . Además, hubo que indicar que en el glosario había no solo unigramas si no también bigramas, como se ha comentado anteriormente.

```
vectorizer_ = CountVectorizer(vocabulary = glosario,ngram_range=(1, 2))

print(vectorizer_.vocabulary)
```

```
['asma', 'defectos', 'medicamentos', 'salud', 'azúcar', 'caso', 'estudio', 'riesgo', 'medicación', 'embarazo', 'cirujía', 'síntomas', 'dolor', 'mujeres', 'enfermedad', 'problema', 'factores', 'pandemia', 'niños', 'productos', 'artículo', 'gobierno', 'ley', 'constitución', 'reforma', 'mayoría', 'debate', 'parte', 'rey', 'congreso', 'comisión', 'país', 'derecho', 'ministerio', 'públicos', 'comunidad', 'partido', 'laboral', 'política', 'electoral', 'ganar', 'derbi', 'Real Madrid', 'Madrid', 'Atlético de Madrid', 'Atleti', 'olímpicos', 'final', 'rival', 'partidos', 'deportivo', 'penalti', 'liga', 'fútbol', 'equipo', 'equipos', 'aficionados', 'futbolistas', 'torneo', 'club', 'balón', 'gol']
```

Figura 3: Código en el que se establece el glosario como vocabulario para el VSM.

Después, utilizando ese vectorizador se obtienen los vectores de cada documentos como se puede ver en la Fig. 4. Como se puede observar, cada vector contiene 63 números, 21 por glosario, que corresponden a la frecuencia de apariciones de sus términos. En el ejemplo que se puede ver en la Fig. 4 se puede ver como un documento de deportes tiene muchos términos de los últimos 21 términos, correspondientes a su tema.

```
X_ = vectorizer_.fit_transform(documents).toarray()
print((X_[1]))
```

```
[0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 2 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 2
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 1 2 3 0 0 1 0 0 3 4]
```

Figura 4: Código en el que se obtienen los vectores de cada documento.

Antes de construir el clasificador se evaluó la similaridad de los documentos de cada tema con su glosario. Para ello se decidió que la medida a utilizar iba a ser la medida del coseno, muy utilizada en el Modelo de Espacio Vectorial, que consiste en calcular el ángulo entre dos vectores en un espacio virtual multidimensional de la manera en la que se puede ver en la ecuación 1 . Esta similaridad se puede como la inversa de la distancia, que se puede obtener de *scipy* mediante la función *spatial.distance.cosine*. Por ello, para calcular la similaridad se resta a 1 la distancia.

$$\cos(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{AB}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} \mathbf{A}_i \mathbf{B}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{A}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{B}_i)^2}} \tag{1}$$

Después se construyó una lista con las similaridades de los vectores de cada documento con un vector que tuviera a 1 la frecuencia de los del glosario en cuestión. Por ejemplo, el vector del glosario deportes tiene los últimos 21 valores a 1, el vector del glosario política tiene los valores del 22 al 42 a 1 y el vector del glosario de salud tiene los primeros 21 valores a 1.

Un ejemplo de de la similaridades de los documentos de política con su glosario se puede ver en la Fig. 5, siendo el documento número 54 el menos similar y el documento 43 el que más similar es.

```
54  0.026660
89  0.125988
38  0.134687
12  0.166390
8   0.197386
37  0.197386
56  0.264628
52  0.273009
90  0.273009
79  0.308607
82  0.333026
67  0.333333
50  0.343093
23  0.363696
35  0.370479
25  0.377964
34  0.404061
19  0.407096
6   0.420897
7   0.461957
61  0.464399
75  0.500517
58  0.500626
42  0.511827
77  0.526361
3   0.558081
2   0.568057
76  0.568057
48  0.586353
43  0.599131
```

Figura 5: Similaridades de los documentos de política con su glosario.

Para el diseño del clasificador se dividieron los documentos en 2 datasets, uno para entrenamiento y otro para test. En concreto se decidió utilizar 45 para entrenamiento y 45 para test. Sobre el modelo se decidieron utilizar tres tipos de clasificadores distintos, **Linear Support Vector Classification**, **Naive Bayes** y **K-Neighbors**

El **Support Vector Machines (SVM)** es un método de aprendizaje que trata los documentos como vectores de un espacio multidimensional. Para clasificarlos, se busca un hiperplano que divida los puntos de cada clase con otra de manera óptima. Este hiperplano debe ser el que tenga la máxima distancia con los puntos mas cercanos, quedando así el espacio multidimensional dividido en secciones lo mas grandes posibles. Se clasificará por tanto el siguiente punto según la región en la que se encuentre.

Una de las ventajas de este método es su eficacia en espacios de grandes dimensiones, ideal para los vectores que representan los documentos, antes explicados. Por otro lado, no proporcionan directamente las estimaciones de probabilidad, se tienen que calcular aparte.

El **Naive Bayes** es un método de clasificación probabilístico basado en el teorema de Bayes. Este concepto probabilístico puede ser descrito como:

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B|A)\Pr(A) + \Pr(B|\neg A)\Pr(\neg A)} \tag{2}$$

Una desventaja es que este teorema asume la independencia de las variables predictoras, es decir en este caso, los términos de los documentos. Esta independencia no tiene por que darse en el mundo del lenguaje natural ya que ciertas palabras pueden ir ligadas a otras y estar condicionadas por ellas.

Por otro lado, este clasificador obtiene muy buenos resultados en una grandisima variedad de casos.

Por último el método **K-Neighbors** es un método de clasificación supervisada, que estima la clase del nuevo punto fijándose de que clase son los k puntos mas cercanos. Para que no hubiera problemas habría que especificar un valor de k impar para que se pueda determinar con mas facilidad la clase mayoritaria de sus vecinos, sin dar lugar a empate. En las pruebas realizadas se terminó un valor de k de 5.

Una vez explicados los clasificadores que se utilizaron, estos fueron los resultados obtenido con cada uno de ellos:

Modelo	Acurracy total	Deportes	Política	Salud
LinearSVC	0.866	1.0	0.944	0.667
Naive Bayes	0.888	1.0	1.0	0.667
K-Neighbors	0.80	1.0	0.722	0.733

Como se puede ver, el modelo que presenta mejor rendimiento fue el Naive Bayes, por lo que se decidió utilizar para las pruebas en el clasificador final.

Para ello se cogieron los 15 documentos de cada tema que no se utilizaron para la creación glosario y se metieron en una carpeta llamada Pruebas. Después. se ejecuta el *script* del *python notebook* y se obtienen 3 carpetas que contienen los documentos anteriores ordenados por temas en función de las predicciones del modelo anterior.

El funcionamiento del *script* es el que se puede ver en la Fig. 6. Primero se eliminan las carpetas de deportes,política y salud, por si no es la primera vez que se ejecuta *script.*A continuación se cargan todos los documentos y sus nombres de archivos. Después se preprocesan como se ha explicado en la sección 3 y se vectorizan. Después, el clasificador predice las clase a la que pertenece cada vector. Y por último se crean las carpetas de las clases y se copian los archivos a la carpeta a la que pertenecen.

Además, como en la carpeta de pruebas los documentos tienen en el nombre la clase a la que pertenece, es muy fácil revisar los errores. Por ejemplo en la Fig. 7 se puede ver fácilmente como ha clasificado como deportes el documento de política número 17 y el número 30 de salud.

```
shutil.rmtree('./Pruebas/deporte')
shutil.rmtree('./Pruebas/politica')
shutil.rmtree('./Pruebas/salud')

filenames_documents = glob.glob('./Pruebas/*')
documentos = load_documents(filenames_documents)


documents = []
for d in range(0, len(documentos)):
    document = re.sub(r'\W', ' ', str(documentos[d]))
    document = document.lower()
    documents.append(document)


vectorizer_ = CountVectorizer(vocabulary = glosario,ngram_range=(1, 2))

X = vectorizer_.fit_transform(documents).toarray()

predictions = classifier.predict(X)

os.mkdir("./Pruebas/deporte/")
os.mkdir("./Pruebas/politica/")
os.mkdir("./Pruebas/salud/")

for prediction,filename in zip(predictions,filenames_documents):
    if prediction == 0:
        shutil.copyfile("./Pruebas/"+filename[10:], "./Pruebas/deporte/"+filename[10:])
    elif prediction == 1:
         shutil.copyfile("./Pruebas/"+filename[10:], "./Pruebas/politica/"+filename[10:])
    else:
        shutil.copyfile("./Pruebas/"+filename[10:], "./Pruebas/salud/"+filename[10:])
```

Figura 6: Script de ordenación de documentos en carpetas

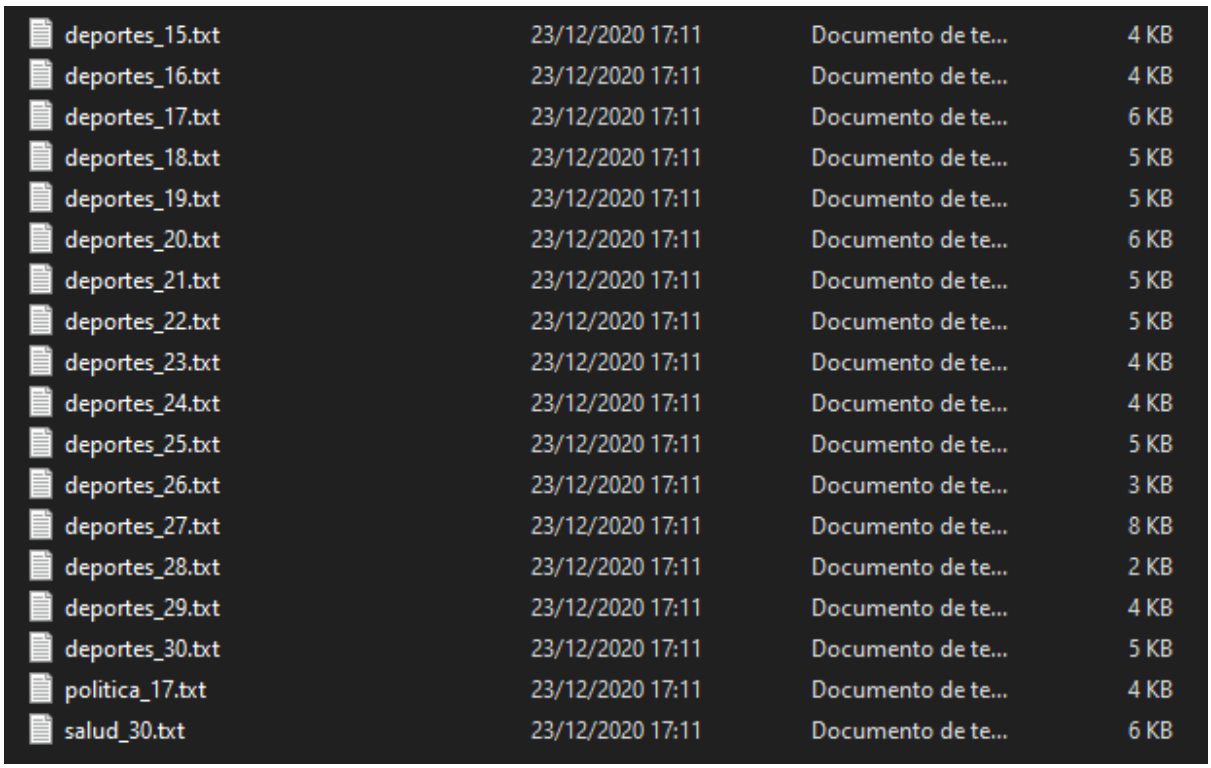

Figura 7: Carpeta de pruebas de deportes

4. Análisis de resultados

En esta sección se comentarán y analizarán los resultados del clasificador explicado en la sección 3. Para ello nos fijaremos en la tabla antes mencionada, que contiene las precisiones totales y las precisiones por cada tema, con el fin de obtener una visión mas completa de los resultados.

En primer lugar, vemos como el modelo SVM tiene un buen rendimiento total de 86.6 %, llegando a clasificar bien todos el 100 % de los documentos de deportes, el 94 % de los de política y un 66.7 % de los de salud.

El modelo de Naive Bayes también presenta un buen rendimiento, obteniendo unos valores de precisión ligeramente superiores al anterior, obteniendo un 6 % mas de precisión para los documentos de política.

Por último, el K-Neighbors es el que peor rendimiento presenta, con una precisión inferior en el tema de política (72.2 %) pero ligeramente superior (73.3 %) en salud.

A nivel general se puede concluir que el clasificador funciona satisfactoriamente, clasificando todos los documentos de deportes bien. Los documentos de política, también están bien clasificador a nivel general. Sin embargo, se nota un descenso de la precisión en salud.

Esto bien puede ser debido a lo explicado en la sección 2, dónde se decía que los documentos extraídos sobre salud son muy variados y costó llegar a tener un glosario del mismo tamaño que los otros dos. Esto implica que el glosario es demasiado pequeño y contiene términos que no son lo suficientemente representativos como para que el clasificador considere que un documento trata sobre salud.

Sin embargo, se llegan a clasificar bien todos los documentos de deportes, lo que implica que el glosario es capaz de representar muy bien el dominio.

Se debe destacar también que debido a que el proceso de extraer documentos manualmente es pesado solo se han utilizado 30 documentos por cada tema para construir el clasificador y probarlo, utilizando 15 documentos por tema para glosario y entrenamiento. Consideramos que si el número de documentos hubiera sido mayor, se podrían haber obtenido unos glosarios mas representativos y por consecuente un resultado mucho mejor.

5. Conclusiones

Para finalizar, en esta sección se van a indicar observaciones sobre el proyecto realizado y los resultados obtenidos.

Primero, cabe destacar que el resultado obtenido en las pruebas y resultados es satisfactorio, ya que se tiene un clasificador que clasifica bien un 0.88 % de los documentos.

A continuación se van a proponer algunos cambios que podrían mejorar el funcionamiento. En primer lugar, se decidió extraer el glosario mediante una herramienta externa y una toma de decisiones manual, pero se podría haber elegido utilizar la frecuencia de aparición de todos los términos de cada documento de cada tema junto con la frecuencia de documentos inversa. Es decir, lo que se conoce como tf-idf. Así se podría haber visto la frecuencia relativa de cada término y poder haber ajustado mas finamente con el glosario. Esto habría evitado también el ruido y habría aportado mas robustez al clasificador. Sin embargo, se considera que los resultados no hubieran variado demasiado con este cambio.

Otra cambio que se planteó fue utilizar tf-idf para refinar los vectores que representan a cada documento. Es decir, en vez de tener un vector que indica el número de veces de aparición de cada término del glosario, tener la frecuencia de cada término con su frecuencia inversa para establecer que tan relevante es ese término en el documento. Esto es muy útil cuando se tienen documentos de distintos tamaños, ya que en documentos mas grandes es mas probable que el número de veces que aparezca un términos sea mayor. Sin embargo, tampoco se considera este cambio tan relevante, ya que todos los documentos tienen un tamaño similar, por lo que se considera que esto no afecta al rendimiento.

Por último, se podrían haber utilizado muchos tipos de clasificadores mas, y haber intentado conseguir una precisión superior al 90 %. Pero se considera que los resultados obtenidos son suficientemente buenos.

Finalmente, se quiere cerrar este documento con la reflexión de que este proyecto ha sido una buena experiencia enriquecedora que sin duda nos van a ayudar de cara al futuro. Ha sido muy entretenido construir este clasificador ver los resultados obtenidos.

6. Anexo

1 Práctica 2 Ingeniería Linguística

[20]:
```
from sklearn.feature_extraction.text import␣
 ↪CountVectorizer,TfidfTransformer,TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
from sklearn.linear_model import SGDClassifier
from sklearn.datasets import load_files
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score, mean_absolute_error
from sklearn.model_selection import train_test_split
import re
import nltk
import os
import shutil
import glob
```

[64]:
```
file = open("./Glosarios/glosario_all.txt",encoding='utf-8')
glosario = file.read().split("\n")
```

[70]:
```
glosario_ = []
for x in glosario:
    glosario_.append(x.lower())

glosario = glosario_
print(glosario)
```

```
['asma', 'defectos', 'medicamentos', 'niños', 'salud', 'personas', 'azúcar',
'caso', 'estudio', 'riesgo', 'medicación', 'embarazo', 'cirujía', 'síntomas',
'dolor', 'mujeres', 'enfermedad', 'problema', 'factores', 'asociación',
'productos', 'artículo', 'artículos', 'ley', 'psoe', 'pp', 'economía',
'reforma', 'mayoría', 'constitución', 'debate', 'española', 'congreso',
'comisión', 'país', 'derecho', 'presidente', 'propuesta', 'rey', 'laboral',
'política', 'gobierno', 'ganar', 'derbi', 'real madrid', 'madrid', 'atlético de
madrid', 'barcelona', 'olímpicos', 'final', 'rival', 'balón', 'deportivo',
```

```
'penalti', 'liga', 'fútbol', 'equipo', 'aficionados', 'futbolistas', 'torneo',
'temporada', 'minutos', 'gol']
```

[71]:
```
articles = load_files(r"./Documentos/",encoding='utf-8')
X, y = articles.data, articles.target
```

[72]:
```
print(len(y))
print(len(X))
print(y)
```

```
90
90
[0 0 1 1 2 1 1 1 2 2 0 0 1 2 0 0 0 1 2 1 0 0 2 1 2 1 2 2 2 0 2 2 0 0 1 1 2
 1 1 0 0 0 1 1 0 2 2 0 1 0 0 1 0 1 0 1 2 1 2 0 1 2 0 2 0 0 1 2 2 2 0 2 2 0
 1 1 1 2 1 0 2 1 0 2 0 2 2 2 1 1]
```

[73]:
```
documents = []
for d in range(0, len(X)):
    document = re.sub(r'\W', ' ', str(X[d]))
    document = document.lower()
    documents.append(document)
```

[74]:
```
vectorizer_ = CountVectorizer(vocabulary = glosario,ngram_range=(1, 2))

print(vectorizer_.vocabulary)
```

```
['asma', 'defectos', 'medicamentos', 'niños', 'salud', 'personas', 'azúcar',
'caso', 'estudio', 'riesgo', 'medicación', 'embarazo', 'cirujía', 'síntomas',
'dolor', 'mujeres', 'enfermedad', 'problema', 'factores', 'asociación',
'productos', 'artículo', 'artículos', 'ley', 'psoe', 'pp', 'economía',
'reforma', 'mayoría', 'constitución', 'debate', 'española', 'congreso',
'comisión', 'país', 'derecho', 'presidente', 'propuesta', 'rey', 'laboral',
'política', 'gobierno', 'ganar', 'derbi', 'real madrid', 'madrid', 'atlético de
madrid', 'barcelona', 'olímpicos', 'final', 'rival', 'balón', 'deportivo',
'penalti', 'liga', 'fútbol', 'equipo', 'aficionados', 'futbolistas', 'torneo',
'temporada', 'minutos', 'gol']
```

[75]:
```
X_ = vectorizer_.fit_transform(documents).toarray()
print((X_[1]))
```

```
[0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 2 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 2 0 0 0 0 4 0 0 0 3 0 0 1 2 3 0 1 0 0 1 4]
```

[76]:
```
#tfidfconverter = TfidfTransformer()
#X_ = tfidfconverter.fit_transform(X_).toarray()
#print(X_[1])
```

[77]:
```
from scipy import spatial
```

```
glosario_deportes = np.zeros(X_[0].shape)
glosario_deportes[43:63] = 1
glosario_politica = np.zeros(X_[0].shape)
glosario_politica[22:43] = 1
glosario_salud = np.zeros(X_[0].shape)
glosario_salud[0:21] = 1

result = 1 - spatial.distance.cosine(X_[4], glosario_deportes)
#print(result)

X_deportes = []
X_politica = []
X_salud = []
cosine_distance_deporte = {}
cosine_distance_politica = {}
cosine_distance_salud = {}

for i in range(90):
    if y[i] == 0:
        X_deportes.append(X_[i])
        cosine_distance_deporte[i] = 1 - spatial.distance.
 ↪cosine(glosario_deportes,X_[i])

    elif y[i] == 1:
        X_politica.append(X_[i])
        cosine_distance_politica[i] = 1 - spatial.distance.
 ↪cosine(glosario_politica,X_[i])

    else:
        X_salud.append(X_[i])
        cosine_distance_salud[i] = 1 - spatial.distance.
 ↪cosine(glosario_salud,X_[i])


cosine_distance_deporte  = {k: v for k, v in sorted(cosine_distance_deporte.
 ↪items(), key=lambda item: item[1])}
cosine_distance_politica = {k: v for k, v in sorted(cosine_distance_politica.
 ↪items(), key=lambda item: item[1])}
cosine_distance_salud = {k: v for k, v in sorted(cosine_distance_salud.items(),␣
 ↪key=lambda item: item[1])}


print(cosine_distance_deporte)
print('#############################################################################################
print(cosine_distance_politica)
print('#############################################################################################
```

```
print(cosine_distance_salud)
```

```
{65: 0.21081851067789192, 15: 0.22360679774997894, 64: 0.24445060351935233, 52:
0.25, 54: 0.2510395055232011, 82: 0.2724746304565331, 21: 0.27279773578818944,
70: 0.27975144247209416, 16: 0.282842712474619, 32: 0.31008683647302115, 41:
0.31943828249996997, 73: 0.32659863237109044, 62: 0.34156502553198664, 79:
0.3585685828003181, 33: 0.3726779962499649, 49: 0.3795360576382948, 50: 0.4, 14:
0.4050957468334666, 11: 0.4129483209670113, 29: 0.45325006416223335, 0:
0.4751644452187054, 84: 0.4880935300919764, 20: 0.49193495504995366, 44:
0.49303180135043745, 59: 0.49852724275079074, 1: 0.5077963596336064, 39:
0.532948040099012, 47: 0.546303217058937, 40: 0.6028035277130152, 10:
0.6085806194501846}
##########################################################################
####################
{35: 0.02646280620124819, 38: 0.1346870059402947, 60: 0.16402261043504096, 37:
0.19738550848793068, 78: 0.19738550848793068, 48: 0.21821789023599236, 75:
0.2519763153394847, 7: 0.273009453115974, 5: 0.33942211665106536, 66:
0.3430925612177752, 17: 0.3624771699523123, 88: 0.363696483726654, 57:
0.3640126041546321, 2: 0.3704792868174742, 25: 0.3779644730092273, 23:
0.39322785175034114, 51: 0.39477101697586137, 3: 0.4234048992199706, 34:
0.45834924851410563, 89: 0.5066739868946863, 19: 0.5095101710852533, 76:
0.5237417843607335, 55: 0.5398861920759762, 42: 0.5580809183750252, 81:
0.5747175593066843, 43: 0.5791405068790082, 12: 0.6002842255014932, 6:
0.6302087912488812, 53: 0.6302087912488812, 74: 0.6467181814331597}
##########################################################################
####################
{45: 0.06434894520877865, 13: 0.06579516949597686, 67: 0.09759000729485334, 9:
0.12434118282549844, 56: 0.17526010166374772, 80: 0.19518001458970669, 24:
0.19738550848793068, 22: 0.2070196678027063, 61: 0.21821789023599236, 85:
0.21821789023599236, 31: 0.23970216768552732, 83: 0.2532457254658621, 69:
0.26898219492864306, 8: 0.28867513459481287, 26: 0.3346491788457354, 36:
0.363696483726654, 18: 0.3779644730092272, 4: 0.38018781261549783, 30:
0.4000661320993194, 68: 0.4270489675302974, 87: 0.4285714285714286, 28:
0.4287526810339921, 63: 0.4300329525375306, 27: 0.4919011134174627, 71:
0.4938291646584312, 46: 0.4991687442297914, 86: 0.5523060897787772, 72:
0.5684289256629115, 77: 0.6061430104032589, 58: 0.6112648394271777}
```

[78]:
```
cosine_distance_politica = pd.DataFrame.
 ↪from_dict(cosine_distance_politica,orient='index')
print(cosine_distance_politica)
```

```
           0
35  0.026463
38  0.134687
60  0.164023
37  0.197386
```

```
78  0.197386
48  0.218218
75  0.251976
7   0.273009
5   0.339422
66  0.343093
17  0.362477
88  0.363696
57  0.364013
2   0.370479
25  0.377964
23  0.393228
51  0.394771
3   0.423405
34  0.458349
89  0.506674
19  0.509510
76  0.523742
55  0.539886
42  0.558081
81  0.574718
43  0.579141
12  0.600284
6   0.630209
53  0.630209
74  0.646718
```

[79]:
```
X_train, X_test, y_train, y_test = train_test_split(X_, y, test_size=0.5,␣
↪random_state=0)
print(len(X_train))
print(len(X_test))
print(len(y_train))
print(len(y_test))
```

```
45
45
45
45
```

[80]:
```
print(y_train)
print(y_test)
```

```
[1 2 0 1 2 0 1 1 0 0 2 1 1 2 0 0 1 0 2 1 0 0 2 1 2 0 0 2 1 1 2 0 0 2 1 0 2
 0 2 2 2 2 0 0 0]
[1 2 1 0 1 2 2 1 1 1 1 2 0 2 2 0 2 2 2 1 2 1 2 1 2 0 0 0 0 1 0 1 2 2 0 1 1
 0 1 2 0 1 0 1 1]
```

[84]:
```
classifier = LinearSVC()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print('Acuraccy :',accuracy_score(y_test, y_pred))
print('Error:',1-accuracy_score(y_test, y_pred))
#print(classifier.predict_proba(X_test))
#print(classification_report(y_test, y_pred))
matrix = confusion_matrix(y_test, y_pred)
matrix.diagonal()/matrix.sum(axis=1)
```

```
Acuraccy : 0.8666666666666667
Error: 0.1333333333333333
```

[84]:
```
array([1.        , 0.94444444, 0.66666667])
```

[85]:
```
classifier = KNeighborsClassifier(5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print('Acuraccy :',accuracy_score(y_test, y_pred))
print('Error:',1-accuracy_score(y_test, y_pred))
#print(classifier.predict_proba(X_test))
#print(classification_report(y_test, y_pred))
matrix = confusion_matrix(y_test, y_pred)
matrix.diagonal()/matrix.sum(axis=1)
```

```
Acuraccy : 0.8
Error: 0.19999999999999996
```

[85]:
```
array([1.        , 0.72222222, 0.73333333])
```

[86]:
```
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print('Acuraccy :',accuracy_score(y_test, y_pred))
print('Error:',1-accuracy_score(y_test, y_pred))
#print(classifier.predict_proba(X_test))
#print(classification_report(y_test, y_pred))
matrix = confusion_matrix(y_test, y_pred)
matrix.diagonal()/matrix.sum(axis=1)
```

```
Acuraccy : 0.8888888888888888
Error: 0.11111111111111116
```

[86]:
```
array([1.        , 1.        , 0.66666667])
```

2 Ordenador por carpetas

[36]:
```
#os.rename("path/to/current/file.foo", "path/to/new/destination/for/file.foo")
#shutil.move("path/to/current/file.foo", "path/to/new/destination/for/file.foo")
#os.replace("path/to/current/file.foo", "path/to/new/destination/for/file.foo"
```

[47]:
```
def load_documents(filenames):
    documents = []
    for i in filenames:
        file_document = open(i,'r',encoding='utf-8')
        documents.append(file_document.read().split())

    return documents
```

[48]:
```
shutil.rmtree('./Pruebas/deporte')
shutil.rmtree('./Pruebas/politica')
shutil.rmtree('./Pruebas/salud')

filenames_documents = glob.glob('./Pruebas/*')
documentos = load_documents(filenames_documents)


documents = []
for d in range(0, len(documentos)):
    document = re.sub(r'\W', ' ', str(documentos[d]))
    document = document.lower()
    documents.append(document)


vectorizer_ = CountVectorizer(vocabulary = glosario,ngram_range=(1, 2))

X = vectorizer_.fit_transform(documents).toarray()

predictions = classifier.predict(X)

os.mkdir("./Pruebas/deporte/")
os.mkdir("./Pruebas/politica/")
os.mkdir("./Pruebas/salud/")

for prediction,filename in zip(predictions,filenames_documents):
    if prediction == 0:
        shutil.copyfile("./Pruebas/"+filename[10:], "./Pruebas/deporte/
 ↪"+filename[10:])
    elif prediction == 1:
         shutil.copyfile("./Pruebas/"+filename[10:], "./Pruebas/politica/
 ↪"+filename[10:])
    else:
```

```
        shutil.copyfile("./Pruebas/"+filename[10:], "./Pruebas/salud/
↪"+filename[10:])
```

[ ]: