Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

Laboratorio: Desambiguación del sentido de las palabras

Procesamiento del Lenguaje Natural

Integrantes:

1. Darwin Yusef Gonzalez

2. Orlando Silva

3. Jose Barrios

4. Ana Hernandez

June 2025

Todo el proyecto se encuentra en el siguiente link

https://github.com/darwinyusef/desambiguacionPLN

Objetivos

Con este laboratorio el alumno conseguirá aplicar diferentes algoritmos basados en aprendizaje automático supervisado para desambiguar el sentido de las palabras. Además, va a aprender a utilizar la herramienta de *software* abierto *Natural Language Toolkit* (NLTK) con la que implementar tareas de procesamiento del lenguaje natural en Python.

Descripción

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

En este laboratorio debes desarrollar e implementar diferentes algoritmos basados en aprendizaje automático supervisado para desambiguar el sentido de las palabras en Python y utilizando la herramienta de *software* abierto Natural Language Toolkit (NLTK).

Para preparar este laboratorio, simplemente descarga e instala **NLTK 3.3** en tu equipo.

Accede a NLTK 3.3 a través del aula virtual o desde la siguiente dirección web: https://www.nltk.org/install.html

NLTK requiere de Python versiones 2.7, 3.4, 3.5 o 3.6 para funcionar. Por lo que, si no tienes instalado Python, descárgalo e instálalo.

Accede a **Python** a través del aula virtual o desde la siguiente dirección web: https://www.python.org/downloads/

Asegúrate de que has instalado NLTK 3.3 adecuadamente antes de la sesión de laboratorio y de revisar el contenido teórico, Semántica léxica y temas anteriores, para tener frescos los diferentes conceptos sobre el procesamiento del lenguaje natural estudiados en esta asignatura.

Durante la sesión del laboratorio debes solucionar un problema sobre desambiguación del sentido de las palabras utilizando el corpus etiquetado en inglés llamado **Senseval 2** y que viene disponible en NLTK.

Accede a más información sobre **Senseval 2** a través del aula virtual o desde la siguiente dirección web:

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

http://www.nltk.org/howto/corpus.html

El primer paso es importar el corpus etiquetado utilizando los siguientes comandos:

```
import nltk
from nltk.corpus import senseval
```

El corpus **Senseval 2** contiene datos etiquetados que sirven para entrenar un clasificador que permita desambiguar el sentido de las palabras. Cada elemento del corpus **Senseval 2** se corresponde con una palabra ambigua. Concretamente en este laboratorio se trabajará con las palabras en inglés «*hard*» y «*serve*», aunque en el corpus hay información de otras dos.

Para poder extraer la información sobre las palabras es imprescindible la manera en la que se identifican en el corpus, es decir, sus identificadores. Con el siguiente comando, se extraen los identificadores de las palabras tratadas en el corpus.

```
senseval.fileids()
```

```
['hard.pos', 'interest.pos', 'line.pos', 'serve.pos']
```

Para cada una de las palabras ambiguas, el corpus contiene una lista de instancias correspondientes a las ocurrencias de esa palabra. Para cada instancia se proporciona la palabra, una lista de sentidos que se aplican a la aparición de esa palabra y el contexto de la palabra.

En la siguiente figura se observa el comando utilizado para visualizar la información que contiene cada instancia de la palabra ambigua «*hard*».

```
senseval.instances('hard.pos')
```

```
[SensevalInstance(word='hard-a', position=20, context=[('``', '``'), ('he', 'PR
P'), ('may', 'MD'), ('lose', 'VB'), ('all', 'DT'), ('popular', 'JJ'), ('suppor
t', 'NN'), (',', ','), ('but', 'CC'), ('someone', 'NN'), ('has', 'VBZ'), ('to',
'TO'), ('kill', 'VB'), ('him', 'PRP'), ('to', 'TO'), ('defeat', 'VB'), ('him',
'PRP'), ('and', 'CC'), ('that', 'DT'), ("'s", 'VBZ'), ('hard', 'JJ'), ('to', 'T
O'), ('do', 'VB'), ('.', '.'), ("''", "''")], senses=('HARD1',)), SensevalInsta
nce(word='hard-a', position=10, context=[('clever', 'NNP'), ('white', 'NNP'),
('house', 'NNP'), ('``', '``'), ('spin', 'VB'), ('doctors', 'NNS'), ("''",
"''"), ('are', 'VBP'), ('having', 'VBG'), ('a', 'DT'), ('hard', 'JJ'), ('time',
'NN'), ('helping', 'VBG'), ('president', 'NNP'), ('bush', 'NNP'), ('explain',
'VB'), ('away', 'RB'), ('the', 'DT'), ('economic', 'JJ'), ('bashing', 'NN'),
('that', 'IN'), ('low-and', 'JJ'), ('middle-income', 'JJ'), ('workers', 'NNS'),
('are', 'VBP'), ('taking', 'VBG'), ('these', 'DT'), ('days', 'NNS'), ('.',
'.')], senses=('HARD1',)), ...]
```

Por ejemplo, en la primera instancia (`SensevalInstance`) la palabra ambigua (`word`) es `'hard-a'`, lo que indica que la palabra es `'hard'` y en este caso la categoría gramatical es un adjetivo, identificado por el sufijo `'-a'`.

El campo `position` indica la posición en la oración en la que se encuentra la palabra ambigua, en este caso la palabra `'hard'` se encuentra en la posición 20.

El campo `context` representa el contexto, es decir, la oración en la que se encuentra la palabra ambigua, en este ejemplo «*he may lose all popular support , but someone has to kill him to defeat him and that 's hard to do trasplantes .* ». El contexto viene representado por pares formados por una palabra y la correspondiente etiqueta gramatical. Por ejemplo, el par (`'he'`, `'PRP'`) que aparece en el contexto indica que la categoría gramatical asociada a la palabra `'he'` es un pronombre personal `'PRP'`.

Por último, el campo `senses` contiene los posibles sentidos de la palabra ambigua, en el ejemplo `'HARD1'`. Los sentidos del corpus hacen referencia a los sentidos de la palabra recogidos en la base de datos de relaciones léxicas WordNet[1].

1 Puede que los sentidos que aparecen en Senseval 2 difieran de los que se encuentran actualmente en WordNet, debido a la constante actualización de este. En este laboratorio no será necesario trabajar con WordNet, se menciona como información adicional.

En este caso 'HARD1' hace referencia la primera definición de la palabra 'hard' que aparece en WordNet, a «difícil», «difficult, hard (not easy; requiring great physical or mental effort to accomplish or comprehend or endure)». Esta información se puede obtener utilizando la interfaz de búsqueda web de WordNet cuyo resultado se muestra en la siguiente figura.

Accede al interfaz de búsqueda de **WordNet** a través del aula virtual o desde la siguiente dirección web:

http://wordnetweb.princeton.edu/perl/webwn

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

Nota: NLTK implementa también un lector para la información disponible en la base de datos de relaciones léxicas WordNet. **Aunque no es necesario para realizar esta actividad de laboratorio,** WordNet se puede importar utilizando el siguiente comando:

```
from nltk.corpus import wordnet
```

En este laboratorio vas a trabajar con algoritmos basados en aprendizaje automático supervisado, por lo tanto, vas a tener que entrenar diferentes clasificadores que permitan desambiguar las palabras ambiguas en inglés «*hard*» y «*serve*», y vas a tener que evaluar el desempeño de los clasificadores creados.

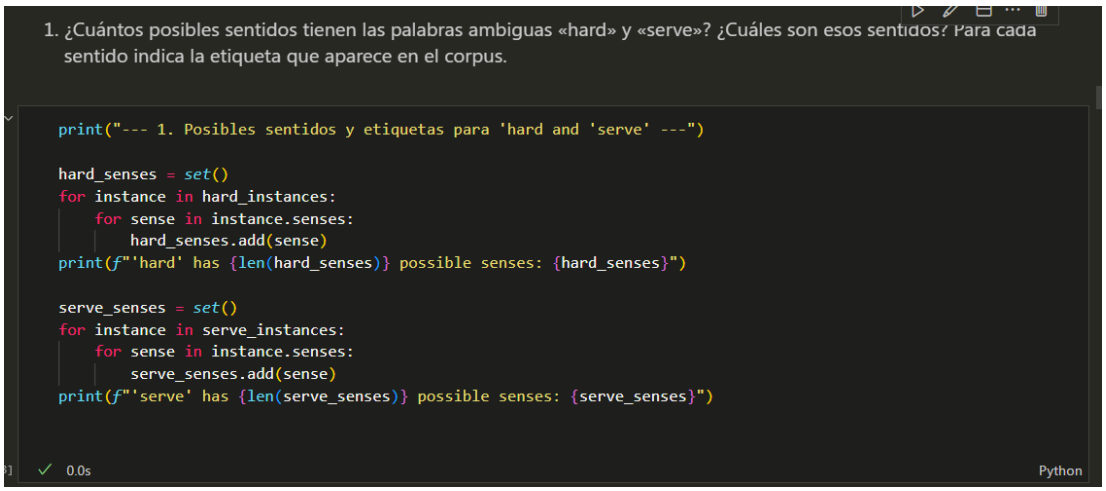
Las diferentes partes que forman este laboratorio se indican a continuación.

Parte 1: Análisis del corpus

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

Analiza el corpus Senseval 2 que vas a utilizar para entrenar los clasificadores. Para realizar el análisis utiliza las funcionalidades que aporta NLTK. Desarrolla el código necesario y responde a las siguientes preguntas.

- ¿Cuántos posibles sentidos tienen las palabras ambiguas «*hard*» y «*serve*»? ¿Cuáles son esos sentidos? Para cada sentido indica la etiqueta que aparece en el corpus.



- ¿Cuántas instancias hay en el corpus para cada uno de los sentidos de las palabras ambiguas «*hard*» y «*serve*»? Es decir, cuantas oraciones hay en el corpus etiquetadas con cada uno de los sentidos.

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

¿Cuántas instancias hay en el corpus para cada uno de los sentidos de las palabras ambiguas «hard» y «serve»? Es decir, cuantas oraciones hay en el corpus etiquetadas con cada uno de los sentidos.

```
print("\n--- 2. Numero de instancias para cada sentido ---")
# --- 2. Número de instancias para cada sentido ---
# Contamos las instancias para cada sentido de la palabra ambigua 'hard'
hard_sense_counts = {}
for instance in hard_instances:
    for sense in instance.senses:
        hard_sense_counts[sense] = hard_sense_counts.get(sense, 0) + 1
print("\nConteo de sentidos para 'hard':")
for sense, count in hard_sense_counts.items():
    print(f"  Sentido '{sense}': {count} instancias")

# Contamos las instancias para cada sentido de la palabra ambigua 'serve'
serve_sense_counts = {}
for instance in serve_instances:
    for sense in instance.senses:
        serve_sense_counts[sense] = serve_sense_counts.get(sense, 0) + 1
print("\nConteo de sentidos para 'serve':")
for sense, count in serve_sense_counts.items():
    print(f"  Sentido '{sense}': {count} instancias")
```

✓ 0.0s Python

```
--- 2. Numero de instancias para cada sentido ---

Conteo de sentidos para 'hard':
  Sentido 'HARD1': 3455 instancias
  Sentido 'HARD2': 502 instancias
  Sentido 'HARD3': 376 instancias

Conteo de sentidos para 'serve':
  Sentido 'SERVE10': 1814 instancias
  Sentido 'SERVE12': 1272 instancias
  Sentido 'SERVE2': 853 instancias
  Sentido 'SERVE6': 439 instancias
```

- En el contexto, las palabras ambiguas pueden aparecer en diferentes formas gramaticales. Por ejemplo, en el caso de la palabra ambigua «hard», esta aparece tanto la forma base, el adjetivo «hard» como en comparativo «harder» y como en superlativo «hardest». ¿Qué formas gramaticales aparecen en el contexto para cada una de las palabas ambiguas «*hard*» y «*serve*»?

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

3. ¿Tienen todas las instancias que forman el corpus el formato que se ha descrito anteriormente? Si hay alguna instancia que no cumpla con ese formato, indica cuales serían las incongruencias que presenta y muestra algunos ejemplos.

```python
# 3. Formatos gramaticales para "hard" y "serve"
print("--- 3. Formatos gramaticales para 'hard' y 'serve' ---")

# Para 'hard'
hard_forms = set()
for instance in hard_instances:
    # instance.context puede contener tuplas (palabra, etiqueta) o solo palabras.
    # Intentaremos manejar ambos casos verificando el tipo de 'item'.
    for item in instance.context:
        word_to_check = ''
        if isinstance(item, tuple):
            word_to_check = item[0]  # Obtener la palabra de la tupla
        elif isinstance(item, str):
            word_to_check = item  # 'item' ya es la palabra como cadena

        if word_to_check:  # Asegurarse de que no esté vacío
            if word_to_check.lower() in {'hard', 'harder', 'hardest'}:
                hard_forms.add(word_to_check.lower())
print(f"Formatos gramaticales en 'hard' encontrados: {hard_forms}")

# Para 'serve'
serve_forms = set()
for instance in serve_instances:
    for item in instance.context:
        word_to_check = ''
        if isinstance(item, tuple):
            word_to_check = item[0]
        elif isinstance(item, str):
            word_to_check = item

        if word_to_check:
            if word_to_check.lower().startswith('serve'):  # Capturar serve, serves, served, serving, etc.
                serve_forms.add(word_to_check.lower())
print(f"Formatos gramaticales en 'serve' encontrados: {serve_forms}")
```

✓ 0.3s Python

```
--- 3. Formatos gramaticales para 'hard' y 'serve' ---
Formatos gramaticales en 'hard' encontrados: {'hardest', 'harder', 'hard'}
Formatos gramaticales en 'serve' encontrados: {'serves', 'server', 'serve', 'served'}
```

- ¿Tienen todas las instancias que forman el corpus el formato que se ha descrito anteriormente? Si hay alguna instancia que no cumpla con ese formato, indica cuales serían las incongruencias que presenta y muestra algunos ejemplos.

4. ¿Tienen todas las instancias que forman el corpus el formato que se ha descrito anteriormente? Si hay alguna instancia que no cumpla con ese formato, indica cuales serían las incongruencias que presenta y muestra algunos ejemplos.

```python
def verify_corpus_format(file_path=corpus_content):

# --- Run the verification ---
verify_corpus_format("corpus_etiquetado.txt")
```

✓ 0.0s Python

```
--- Verifying format of 'corpus_etiquetado.txt' ---

--- Summary for 'corpus_etiquetado.txt' ---
Total documents processed: 20

All processed lines conform to the 'Word\tLemma\tPOS_Tag' format within documents.

--- Examples of Valid Lines (Word\tLemma\tPOS_Tag) ---
  Doc 1, Line 3: 'Habla hablar  VMIP3S0' (Valid)
  Doc 1, Line 4: 'con   con     SP' (Valid)
  Doc 1, Line 5: 'el    el      DA' (Valid)

--- Additional Notes ---
In this specific corpus format, an 'inconsistency' means a line that doesn't have exactly 3 tab-separated components (Word, Lemma, POS Tag).
Missing blank lines between documents or malformed <doc id> tags would also be inconsistencies.
```

Parte 2: Extracción de características

Para poder entrenar un clasificador es necesario extraer un conjunto de características lingüísticas a partir del corpus etiquetado. Por lo tanto, debes crear el código en Python que te permita extraer diferentes conjuntos de características a partir de Senseval 2.

Extracción de características basada en las palabras vecinas

Debes extraer un **conjunto de características basado en las palabras vecinas**. Para una instancia del corpus, debes desarrollar el código que sea capaz de extraer el vector de características que indican si las palabras de un vocabulario, que se debe construir previamente, aparecen o no en el contexto (la oración completa en que aparece la palabra ambigua).

Para una instancia de la palaba ambigua «*hard*» su contexto se muestra a continuación:

```
In [22]: inst = instances_hard[1]

In [23]: inst.context

Out[23]: [('clever', 'NNP'),
          ('white', 'NNP'),
          ('house', 'NNP'),
          ('``', '``'),
          ('spin', 'VB'),
          ('doctors', 'NNS'),
          ("''", "''"),
          ('are', 'VBP'),
          ('having', 'VBG'),
          ('a', 'DT'),
          ('hard', 'JJ'),
          ('time', 'NN'),
          ('helping', 'VBG'),
          ('president', 'NNP'),
          ('bush', 'NNP'),
          ('explain', 'VB'),
          ('away', 'RB'),
          ('the', 'DT'),
          ('economic', 'JJ'),
          ('bashing', 'NN'),
          ('that', 'IN'),
          ('low-and', 'JJ'),
          ('middle-income', 'JJ'),
          ('workers', 'NNS'),
          ('are', 'VBP'),
          ('taking', 'VBG'),
          ('these', 'DT'),
          ('days', 'NNS'),
          ('.', '.')]
```

Extracción de características basada en las palabras vecinas

```
inst.context
```

✓ 0.0s

```
[('clever', 'NNP'),
 ('white', 'NNP'),
 ('house', 'NNP'),
 ('``', '``'),
 ('spin', 'VB'),
 ('doctors', 'NNS'),
 ("''", "''"),
 ('are', 'VBP'),
 ('having', 'VBG'),
 ('a', 'DT'),
 ('hard', 'JJ'),
 ('time', 'NN'),
 ('helping', 'VBG'),
 ('president', 'NNP'),
 ('bush', 'NNP'),
 ('explain', 'VB'),
 ('away', 'RB'),
 ('the', 'DT'),
 ('economic', 'JJ'),
 ('bashing', 'NN'),
 ('that', 'IN'),
 ('low-and', 'JJ'),
 ('middle-income', 'JJ'),
 ('workers', 'NNS'),
 ('are', 'VBP'),
 ('taking', 'VBG'),
 ('these', 'DT'),
 ('days', 'NNS'),
 ('.', '.')]
```

Suponiendo que el vocabulario usado para crear extraer las características es el siguiente:

```
In [21]: vocab
Out[21]: ['time', 'would', 'get', 'work', 'find', 'make']
```


Entonces el vector de características extraídas para esa instancia sería:

```
Out[24]: {'contains(time)': True,
          'contains(would)': False,
          'contains(get)': False,
          'contains(work)': False,
          'contains(find)': False,
          'contains(make)': False}
```

Este vector de características indica que en el contexto de la palabra ambigua aparece la palabra «time» y no aparecen las palabras «would», «get», «work», «find» y «make».

Para extraer el vector de características basado en las palabras vecinas debes seguir los siguientes pasos:

```
# Contexto de la instancia de "hard"
# Simulamos un contexto donde "time" aparece y el resto no.
contexto_hard = ["este", "vector", "de", "características", "indica", "que", "en", "el", "contexto", "de", "la", "palabra",
                 "ambigua", "aparece", "la", "palabra", "time", "y", "no", "aparecen", "las", "palabras",
                 "would", "get", "work", "find", "y", "make"]

# Extraer el vector de características
vector_caracteristicas_hard = extraer_caracteristicas_palabras_vecinas(contexto_hard, vocab)

print(f"Vocabulario: {vocab}")
print(f"Contexto de 'hard': {' '.join(contexto_hard)}")
print(f"Vector de características para 'hard': {vector_caracteristicas_hard}")



# Si el contexto_hard fuera el que describiste:
contexto_hard_solicitado = ["este", "es", "un", "ejemplo", "con", "time", "pero", "sin", "would", "get", "work", "find", "make"]
vector_caracteristicas_hard_solicitado = extraer_caracteristicas_palabras_vecinas(contexto_hard_solicitado, vocab)
print(f"Contexto de 'hard' (ejemplo solicitado): {' '.join(contexto_hard_solicitado)}")
print(f"Vector de características para 'hard' (ejemplo solicitado): {vector_caracteristicas_hard_solicitado}")
```

Python

```
Vocabulario: {'get', 'make', 'time', 'would', 'find', 'work'}
Contexto de 'hard': este vector de características indica que en el contexto de la palabra ambigua aparece la palabra time y no aparecen las palabras would get
Vector de características para 'hard': [1, 1, 1, 1, 1, 1]
Contexto de 'hard' (ejemplo solicitado): este es un ejemplo con time pero sin would get work find make
Vector de características para 'hard' (ejemplo solicitado): [1, 1, 1, 1, 1, 1]
```

1. **Construcción del vocabulario o *bags of words*.** Como ya se ha indicado, para poder obtener el vector de características, se debe construir previamente un vocabulario. Para cada una de las palabras del vocabulario, se debe consultar si la palabra aparece en el contexto de la palabra ambigua. Si la palabra del vocabulario aparece en el contexto entonces en el vector de características aparecerá True para esa palabra y si no, False.

   Tomemos como ejemplo un vocabulario creado sobre el que se ha construido el vector de características. Este vocabulario es `['time', 'would', 'get', 'work',`

`'find', 'make']`. Este vocabulario se usará posteriormente para construir el vector de características del ejemplo que usamos para darte una orientación.

- **¿Cómo construyo mi vocabulario o bags of words?**

Lo que debes hacer para tu entrega de este laboratorio es utilizar como vocabulario las **m** palabras más frecuentes que aparecen en las instancias que conforman el conjunto de datos, es decir en las oraciones que contienen las palabras ambiguas y que forman parte del corpus. Entonces, para crear la ***bag of words*** (bolsa de palabras) debes extraer el conjunto de las n palabras más frecuentes. Para ello te puedes ayudar de la función `nltk.FreqDist()` que proporciona información sobre la distribución de frecuencias de las palabras que aparecen en un texto.

- Cuando obtengas las palabras más frecuentes, debes eliminar los signos que puntuación y las palabras vacías (aquellas sin significado como artículos, pronombres o preposiciones, las llamadas *stop words* en inglés). También debes eliminar las diferentes formas gramaticales de la palabra ambigua, por

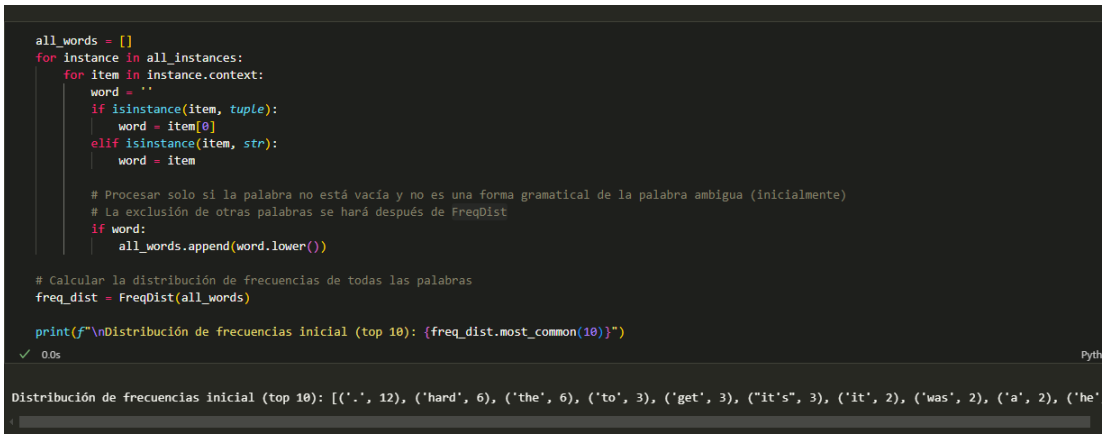
ejemplo, para desambiguar la palabra «*hard*» no tendría sentido utilizar la palabra «*harder*» ni la palabra «*hardest*».

- Para la eliminación del conjunto de palabras no útiles del vocabulario se puede usar un código parecido al que se indica a continuación. Debes tener en cuenta que **en este código faltaría añadir las palabras que has identificado en la Parte 1** de este laboratorio como las diferentes formas gramaticales de las palabras ambiguas.

```
from nltk.corpus import stopwords
import string
OTHER_WORDS = ["'", "'d", "'ll", "'m", "'re", "'s", "'t", "'ve",
'--', '000', '1', '2', '3', '4', '5', '6', '8', '10', '15', '30',
'I', 'F', '``', 'also', "don'", 'n', 'one', 'said', 'say', 'says',
'u', 'us']
STOPWORDS_SET =
        set(stopwords.words('english')).union(set(string.punctuation),
        set(OTHER_WORDS))
```

- **Ejemplo de vocabulario para un tamaño de 6**. Por ejemplo, si se quiere entrenar un clasificador que permita identificar los diferentes sentidos de la

palabra «*hard*» y se utilizan para entrenar y validar el modelo las instancias etiquetadas para esta palabra, la bolsa de palabras en el caso de considerar las seis palabras más frecuentes (m=6) sería la presentada anteriormente `['time', 'would', 'get', 'work', 'find', 'make']`.



2. **Construcción del conjunto de características basado en palabras vecinas.** Utiliza un diccionario en Python para guardar el conjunto de características. La clave del diccionario serán las palabras del vocabulario y el valor debe ser un *booleano* para indicar la aparición o no de las palabras en el contexto. Por ejemplo, en el vector de características `{'contains(time)': True, 'contains(would)': False, 'contains(get)': False, 'contains(work)': False, 'contains(find)': False, 'contains(make)': False}` una de las claves del diccionario es `'contains(time)'` y su valor es `True` lo que indica que en el contexto de la palabra ambigua aparece la palabra «time».

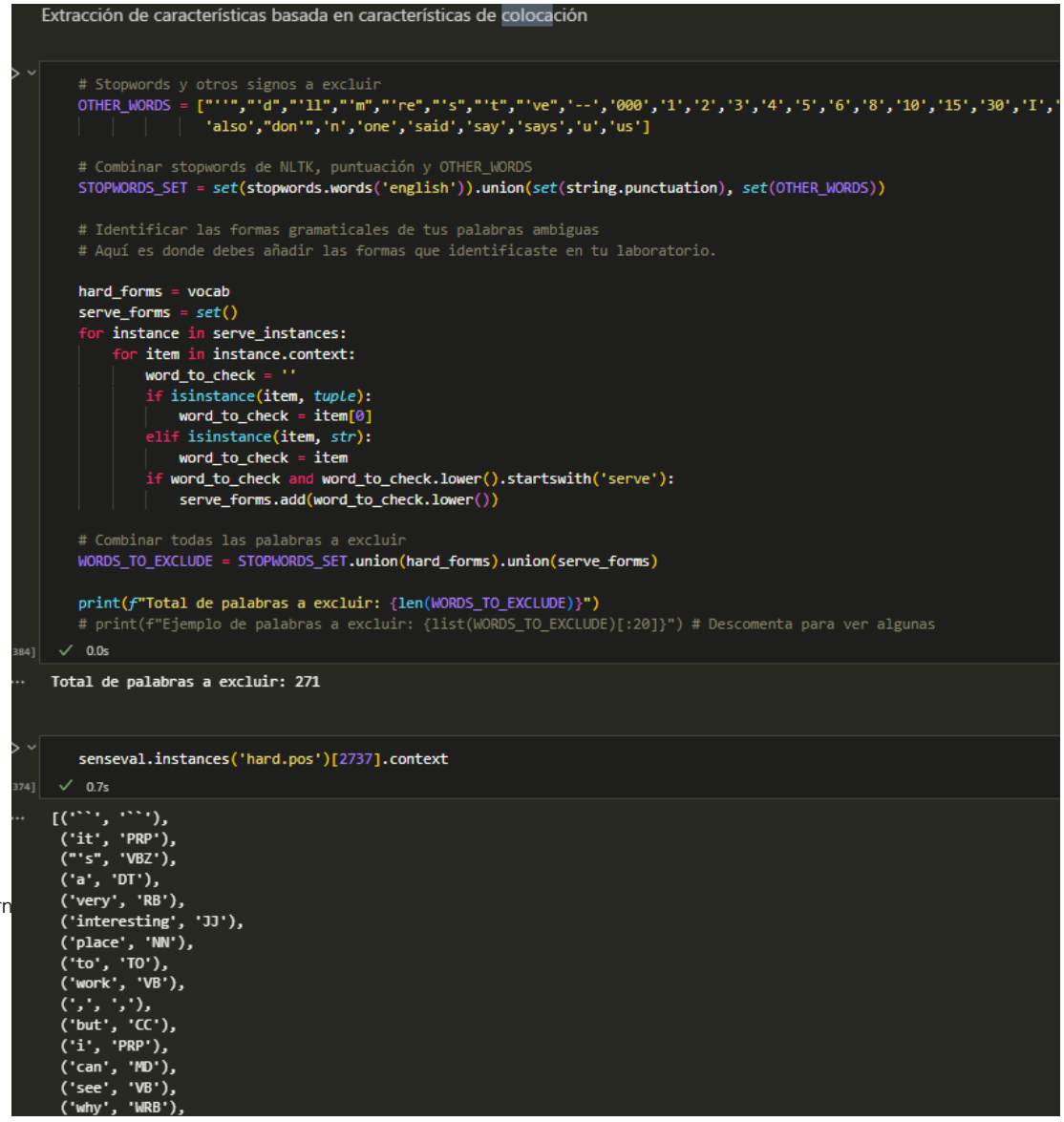
   - Debes mostrar el vector de características resultante para una de las instancias del corpus.

   - **Importante:** En el cómputo del vector de características basado en las palabras vecinas debes utilizar como contexto la oración completa donde aparece la palabra ambigua. Es decir, todas las palabras que forman la oración guardada en el campo `context` de la instancia.

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

Extracción de características basada en características de colocación

Debes extraer también un **conjunto de características de colocación**. Para una instancia del corpus, debes desarrollar el código que sea capaz de extraer el vector de características formado por la secuencia de *n* palabras que ocurren antes de la palabra ambigua y la secuencia de *n* palabras que ocurren después de la palabra ambigua, los llamados n-gramas.



Para una instancia de la palaba ambigua «*hard*» su contexto se muestra a continuación:

Entonces el vector de características de colocación para el bigrama anterior y posterior sería:

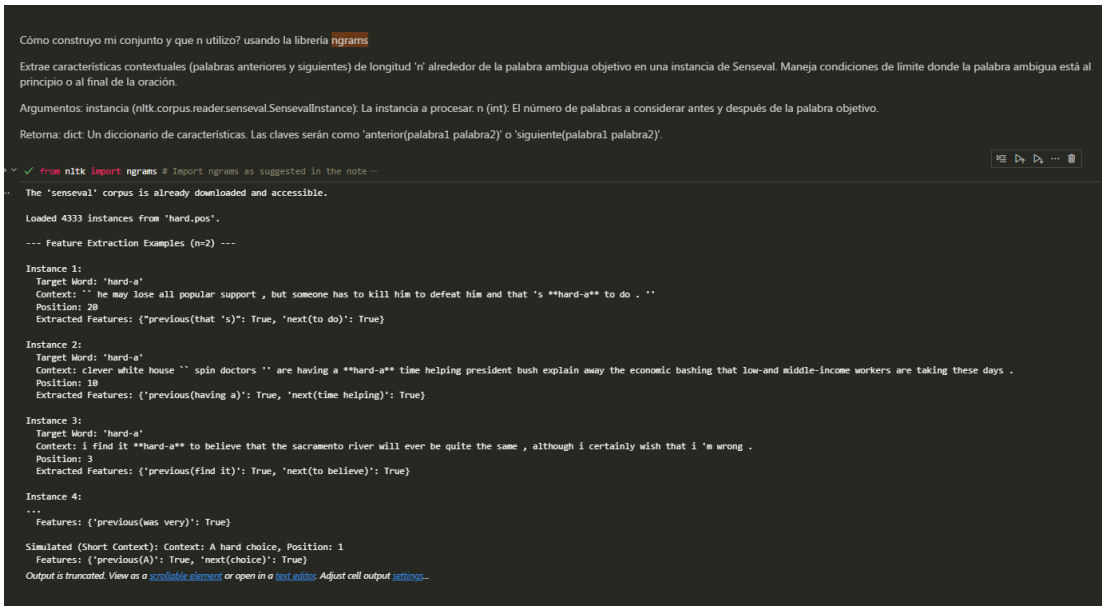
```
Out[36]: {'previous(have a)': True, 'next(time imagining)': True}
```

Este vector de características indica que antes de la palabra ambigua se encuentran las palabras «have a» y después de la palabra ambigua las palabras «time imagining».

- **¿Cómo construyo mi conjunto y que n utilizo?** Utiliza un diccionario en Python para guardar el conjunto de características, la clave del diccionario debe indicar la secuencia de palabras de contexto y si aparecen antes o después de la palabra ambigua y el valor asociado a la clave debe ser un booleano verdadero. Usaremos **n=2.**

  Por ejemplo, el vector de características `{'previous(have a)': True, 'next(time imagining)': True}` una de las claves del diccionario es `'previous(have a)'` y su valor es `True` lo que indica que antes de la palabra ambigua se encuentran las palabras «have a». En este caso, al tener secuencias de dos palabras (***n=2***), se están considerando bigramas y la ventana tendría tamaño cinco (2n+1). Por lo tanto, si la palabra ambigua es «*hard*» en el contexto guardado en el campo `context` de la instancia, aparece la siguiente parte de la frase «*have a hard time imagining*».

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	



- Debes mostrar el vector de características resultante para una de las instancias del corpus.

- **Importante:** Debes tener en cuenta los posibles casos en los que la palabra ambigua aparezca al principio o final de la frase, ya que en esas instancias no vas a poder obtener una secuencia de palabras de longitud n. Por ejemplo, para la instancia cuyo contexto es: `[('some', 'DT'), ('hard', 'JJ'), ('choices', 'NNS'), ('had', 'VBD'), ('to', 'TO'), ('be', 'VB'), ('made', 'VBN'), …]` si *n=2* deberías obtener el siguiente vector de características: `{'previous(some)': True, 'next(choices had)': True}`.

- **Nota**: aunque no es imprescindible para realizar esta actividad de laboratorio, puedes utilizar las funcionalidades para trabajar con n-gramas que ofrece NLTK. Estas se pueden importar utilizando el siguiente comando:


```
from nltk import ngrams
```

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

Extracción de características con otro criterio

Propón un tercer conjunto de características que puedas obtener del corpus Senseval 2. Razona porque crees que puede ser bueno el conjunto de características que tú propones, no es necesario que lo implementes en Python.

Parte 3: Entrenamiento de clasificadores

Para los entrenamientos usamos

Debes entrenar diferentes clasificadores que permitan desambiguar las palabras ambiguas en inglés «*hard*» y «*serve*». Además, vas a tener que evaluar el desempeño de los clasificadores creados. Por lo tanto, debes crear el código en Python que te permita entrenar estos clasificadores y evaluarlos.

El tipo de clasificador que vas a utilizar en este laboratorio es el *Naive Bayes*. Para importar el clasificador y el paquete que te permita evaluar su rendimiento debes utilizar el siguiente comando:

```
from nltk.classify import accuracy, NaiveBayesClassifier
```

Una vez hayas importado los paquetes anteriores, para entrenar un clasificador *Naïve Bayes* puedes usar el comando `NaiveBayesClassifier.train()` y para evaluarlo `accuracy()`. Además, puedes utilizar el clasificador entrenado para clasificar una instancia utilizando su método `classify()`. Por último, puedes obtener la matriz de confusión utilizando el comando `nltk.ConfusionMatrix()`.

Para realizar esta parte del laboratorio debes seguir los siguientes pasos:

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

1. **Entrenamiento de dos clasificadores para la palabra «hard».** Debes entrenar dos clasificadores que permitan desambiguar la palabra «*hard*», es decir, que debes entrenar los clasificadores utilizando las instancias disponibles en el corpus Senseval 2 para esta palabra ambigua.

   - **Conjuntos de entrenamiento y test**. Para entrenar y validar divide las instancias disponibles en una proporción del 80-20 % y recuerda que en el conjunto de datos de entrenamiento deben aparecer instancias de todas las clases.

   - **Clasificador basado en las palabras vecinas.** Con el conjunto de datos de entrenamiento, entrena un clasificador para «*hard*» que use como características el conjunto basado en las **palabras vecinas** cuyo código has implementado en la parte 2 de este laboratorio. Para definir el vocabulario utiliza las **250** palabras más frecuentes (***m=250***).

   - **Clasificador basado en características de colocación**. Con el conjunto de datos de entrenamiento, entrena un clasificador para «*hard*» que use como conjunto de características las de colocación cuyo código has implementado en la parte 2 de este laboratorio. Para definir la **ventana de contexto** utiliza la secuencia de dos palabras que ocurren antes de la palabra ambigua y la secuencia de dos palabras que ocurren después de esta **(n=2).**

2. **Validación de los clasificadores para la palabra «hard».** Utilizando el conjunto de datos de test que has generado previamente, obtén la exactitud (*accuracy*) y la matriz de confusión para cada uno de los dos clasificadores que permiten desambiguar el sentido de la palabra «*hard*».

- Debes mostrar la exactitud (*accuracy*) y la matriz de confusión resultantes de la validación de cada uno de los dos clasificadores.

Debes comparar y analizar los resultados de rendimiento de los clasificadores. Para ello responde a las siguientes preguntas:

- ¿Cuál es el conjunto de características que aporta mejores resultados? ¿Por qué?
- ¿Cuál es el sentido más difícil de identificar? ¿Por qué?
- ¿Qué posibles mejoras se podrían aplicar para mejorar el rendimiento de los clasificadores creados? No es necesario que las implementes, solo que las comentes.

3. **Instancias clasificadas incorrectamente para «*hard*».** Para el clasificador que permite desambiguar la palabra «*hard*» y que utiliza las características de colocación, obtén las instancias que pertenecen al sentido `'HARD1'` y que se han clasificado incorrectamente.

   - Presenta en el informe la oración en la que aparece la palabra ambigua (el contexto) para cada una de esas instancias y la etiqueta en la que han sido erróneamente clasificadas.

4. **Entrenamiento y validación de dos clasificadores para la palabra «serve».** Repite el proceso anterior para entrenar y validar dos clasificadores que permitan desambiguar la palabra «*serve*». Puedes aprovechar el código que has generado anteriormente.

- Crea los conjuntos de entrenamiento y test para las instancias donde la palabra ambigua es «*serve*». Mantén la proporción del 80-20 % para la creación de los conjuntos de entrenamiento y de test.
- Entrena un clasificador para «*serve*» que use como características el conjunto basado en las **palabras vecinas**. Para definir el vocabulario utiliza las **250** palabras más frecuentes (***m=250***).

- Entrena un clasificador para «*serve*» que use como conjunto **de características las de colocación**. Para definir la **ventana de contexto** utiliza la secuencia de dos palabras que ocurren antes de la palabra ambigua y la secuencia de dos palabras que ocurren después de esta **(n=2).**

- Obtén el valor la exactitud (*accuracy*) para cada uno de los dos clasificadores que permiten desambiguar el sentido de la palabra «*sense*».

5. **Análisis de resultados del rendimiento de los clasificadores**. Presenta en el informe una tabla resumen con los valores de exactitud para cada uno de los 4 clasificadores (dos para cada palabra ambigua) que has entrenado previamente y responde a las siguientes preguntas:

- ¿Por qué no es justo comparar directamente la exactitud aportada por los clasificadores que han aprendido diferentes palabras ambiguas?
- ¿Cómo podrías hacerlo para que la comparación entre clasificadores que desambiguan palabras diferentes tenga sentido?
- Compara la exactitud de los clasificadores con la que proporcionaría un clasificador que asignara el sentido de forma aleatoria. ¿Cuál sería el mejor clasificador tomando como referencia (*baseline*), el clasificador aleatorio?

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

Parte 4: Conclusiones sobre el uso de aprendizaje automático supervisado para desambiguar el sentido de las palabras

Una vez hayas implementado diferentes clasificadores para desambiguar el sentido de diferentes palabras y analizado su desempeño, reflexiona sobre el uso de algoritmos basados en aprendizaje automático supervisado para resolver la tarea de desambiguación del sentido de las palabras. Para ello responde de forma razonada a las siguientes preguntas:

- ¿Cuáles son las limitaciones de los clasificadores que has creado para la desambiguación del sentido de las palabras?

El aprendizaje automático supervisado, como lo hemos aplicado con el clasificador Naive Bayes, es una herramienta poderosa para la desambiguación del sentido de las palabras (WSD). Ponemos este ejemplo queremos enseñarle a un programa a entender qué quiere decir la palabra "duro" en diferentes frases. Por ejemplo, no es lo mismo "un problema duro" (difícil) que "una piedra dura" (resistente).

- ¿Qué alternativas propondrías para superar esas limitaciones y obtener algoritmo que resuelva mejor el problema de la desambiguación del sentido de las palabras?

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

Reflexiones sobre el entrenamiento de clasificadores Limitaciones de nuestra enseñanza actual:

1. **"Olvido" de lo que aprendió antes:** Nuestro programa (el clasificador Naive Bayes) es un poco "ingenuo". Cuando mira una palabra cerca de "duro" (como "problema"), asume que la aparición de "difícil" no influye en si aparece "examen". Pero en la vida real, ¡claro que sí! Si ves "problema" es más probable que también veas "examen". Esta suposición simplifica demasiado el lenguaje.
2. **Visión de túnel:** El programa solo mira las 2 palabras antes y 2 después de "duro", o un vocabulario de las 250 palabras más frecuentes. Si la clave para entender "duro" está en una palabra que apareció mucho antes en la frase, ¡se la pierde! Es como intentar entender una broma viendo solo las últimas dos palabras.
3. **No entiende significados profundos:** El programa no sabe que "examen" y "problema" se parecen semánticamente, ni que "piedra" y "madera" son cosas sólidas. Solo sabe si una palabra *está* o *no está*.

Reflexiones sobre el entrenamiento de clasificadores ahora pensemos Cómo podríamos enseñarle mejor

1. **Es necesario darle más pistas**
2. **Usar diccionarios "inteligentes"**
3. **"Pedirle" que lea más**
4. **Usa cerebros más complejos (Redes Neuronales)**

Al final tenemos un micro cerebro artificial que pueda "leer" toda la frase, recordar lo que pasó al principio, y entender las relaciones entre palabras, incluso si están lejos.

Tenemos Modelos como BERT que literal son cerebros más grandes son como tener a un lector súper avanzado que entiende el contexto completo de una palabra antes de decidir su significado.

Y por ultimo tenemos Modelos más avanzados como deepseek R1 y los nuevos modelos de GEMINI FLASH y PRO o OPENAI que son mucho más fuertes en este aspecto y nos permiten generar mucho más procesamiento

Conclusiones sobre la desambiguación

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

Entender el sentido exacto de una palabra según el contexto en el que aparece —lo que se conoce como desambiguación del sentido de las palabras— es un reto clave en el procesamiento del lenguaje. Esto es algo que influye directamente en tareas tan comunes como traducir textos, buscar información en línea o incluso hacer que una máquina comprenda lo que alguien está diciendo o escribiendo.

Durante mucho tiempo, se ha confiado en métodos de aprendizaje automático supervisado para resolver este problema. Es decir, se entrena un modelo con muchos ejemplos ya clasificados, para que aprenda a distinguir qué sentido tiene una palabra en diferentes situaciones. Cuando hay suficientes datos buenos y bien etiquetados, estos modelos funcionan bastante bien.

El problema es que no siempre se cuenta con ese tipo de datos. Anotar textos a mano, marcando el significado exacto de cada palabra según el contexto, lleva mucho tiempo y recursos. Además, hay palabras con sentidos muy raros o que solo aparecen en contextos muy específicos, por lo que es difícil que el modelo aprenda a manejarlos correctamente.

Frente a estas limitaciones, han surgido otras formas de abordar el problema. Una de ellas es usar métodos no supervisados o semi-supervisados, que no dependen tanto de datos anotados a mano. También se están usando cada vez más modelos de lenguaje avanzados —como BERT o GPT— que han sido entrenados con enormes cantidades de texto y pueden captar bastante bien los matices del lenguaje sin necesidad de ejemplos explícitos.

Otra opción interesante es la desambiguación basada en conocimientos, que se apoya en bases de datos lingüísticas como WordNet para deducir el significado más probable según el contexto.

En resumen, aunque los métodos supervisados han sido muy útiles, no son suficientes por sí solos. El futuro de esta tarea parece estar en una combinación inteligente de enfoques: modelos potentes, grandes volúmenes de texto, y recursos lingüísticos que aporten estructura y significado.

Criterios de evaluación

- Debe resolver la actividad en el Jupyter Notebook proporcionado junto con este enunciado. Si no se usa el archivo proporcionado la actividad será calificada con cero puntos.

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

- Para la evaluación de la actividad se debe entregar el **Jupyter Notebook** que contenga:
  - El código en Python que permita resolver la actividad, debidamente comentado.
  - Todos los resultados que se solicita en el enunciado y que genera el código implementado.
  - Las respuestas a las preguntas planteadas en el enunciado.
- Se valorarán positivamente, los comentarios del código claros y oportunos, que permitan entender las decisiones de implementación realizadas. Si el código no está comentado de forma suficiente, se penalizará en la calificación de la actividad.
- Se valorará positivamente, la respuesta clara, breve y bien argumentada a las preguntas.
- Si se presenta el código, pero no se muestran los resultados que éste genera, no se considerarán esos resultados en la calificación de la actividad.
- Si se detecta **plagio** de cualquier fuente (internet…), ya sea en el código, en los comentarios o en las respuestas, el alumno obtendrá una calificación para la actividad de cero puntos.
- Si se detecta **copia** entre alumnos en el código, en los comentarios o en las respuestas, todos los alumnos involucrados obtendrán una calificación para la actividad de cero puntos.

Extensión máxima: no hay restricciones en la extensión.

Laboratorio: Desambiguación del sentido de las palabras (valor real: 5 puntos)	Descripción	Puntuación máxima (puntos)	Peso %
Análisis del corpus	El código para analizar el corpus es correcto y las respuestas a las preguntas sobre el corpus son correctas.	2	20%
Extracción de	El código para extraer tres conjuntos	3	30%

Asignatura	Datos del alumno	Fecha
Procesamiento del Lenguaje Natural	Apellidos: G,S,B,H	16/06/2025
	Nombre: D,O,J,A	

características	diferentes de características a es correcto.		
Código para el entrenamiento clasificadores	El código para entrenar los clasificadores, evaluar su rendimiento y presentar los errores de clasificación es correcto.	2	20%
Reflexiones sobre el entrenamiento de clasificadores	Las respuestas a las preguntas sobre el rendimiento de los diferentes clasificadores entrenados para desambiguar son correctas.	2	20%
Conclusiones sobre la desambiguación	Las respuestas a las preguntas sobre el uso de aprendizaje automático supervisado para desambiguar el sentido de las palabras, sus limitaciones y las posibles alternativas, son correctas.	1	10%
		10	**100 %**