PRÁCTICA 1 Minería de Textos

Alumno: Alex Herrerías Ramírez

8 de diciembre de 2025

Índice

1. Descripción de la tarea

El objetivo de la práctica consiste en la detección y clasificación de nombres propios en texto libre, para ello crearé un script en Python que procese el archivo de entrada y junto al modelo Spacy genere unas predicciones que se compararan con el script adjuntado en el ejercicio que usa el corpus estándar *CoNLL-2002*.

1.1. Tipos de entidades

El sistema debe ser capaz de identificar y clasificar las siguientes cuatro categorías:

- **PER**: Personas (nombres propios de individuos reales o ficticios).
- **ORG**: Organizaciones (empresas, instituciones públicas, partidos políticos, etc.).
- **LOC**: Localizaciones (países, ciudades, montañas, ríos, etc.).
- **MISC**: Miscelánea. Esta categoría engloba entidades que no encajan en las anteriores, como nacionalidades, eventos, títulos de obras o productos comerciales.

1.2. Formato de los datos

El corpus utiliza el sistema de etiquetado **IOB**. Este formato nos permite representar entidades compuestas por varias palabras usando prefijos en las etiquetas:

- **B-ETIQUETA (Begin):** Indica que la palabra es el inicio de una entidad.
- **I-ETIQUETA (Inside):** Indica que la palabra forma parte de la entidad iniciada anteriormente.
- **O (Outside):** Indica que la palabra no pertenece a ninguna entidad nombrada.

Token	**Etiqueta**
El	O
Banco	**B-ORG**
de	**I-ORG**
España	**I-ORG**
anunció	O
ayer	O
...	...

Cuadro 1: Ejemplo del formato IOB

2. Desarrollo e implementación

2.1. Herramientas utilizadas

Para el desarrollo de la práctica se ha usado Python como lenguaje, la librería Spacy para tener acceso a los modelos `es_core_news_sm` y `es_core_news_lg`, y como script para la evaluación se utilizó el archivo `conlleval.py` adjuntado en la practica.

2.2. Descripción del código

El script principal se ha desarrollado en el archivo `practica_base.py` con la siguiente estructura:

1. **Importación de librerías**

   Primero se carga la librería Spacy para cargar el modelo y procesar las oraciones.

```
import spacy
from spacy.tokens import Doc
import os
```

2. **Función principal**

   La función `procesar_archivo_conll` recibe como entrada el archivo con el corpus (`esp.testb.txt`) y el archivo donde se guardarán los resultados.

```
def procesar_archivo_conll(archivo_entrada, archivo_salida):
```

3. **Carga del modelo de Spacy**

   Se intenta cargar el modelo `es_core_news_sm`, si no esta instalado o hay algun error se indica y se para el proceso.

```
try:
    nlp = spacy.load("es_core_news_sm")
    print("modelo␣cargado")
except OSError:
    print("error␣al␣cargar␣el␣modelo")
    return
```

4. **Comprobación del archivo de entrada**

   Antes de procesar, se verifica que el archivo `esp.testb.txt` está disponible.

```
if not os.path.exists(archivo_entrada):
    print("error␣al␣cargar␣el␣archivo␣testb")
    return
```

5. **Estructuras de almacenamiento**

   Se inicializan las listas necesarias que se usaran posteriormente para guardar las oraciones, sus palabras y sus etiquetas reales.

```
oraciones_palabras = []
oraciones_gold = []

palabras_actuales = []
etiquetas_actuales = []
```

6. **Lectura del archivo CoNLL**

   Se recorre el archivo línea por línea. Cada línea se basa en una palabra y su etiqueta, con un salto de línea vacío al final de la oración.

```
with open(archivo_entrada, 'r', encoding='utf-8',
errors='ignore') as f:
    for linea in f:
        linea = linea.strip()

        if not linea:
            if palabras_actuales:
                oraciones_palabras.append(palabras_actuales)
                oraciones_gold.append(etiquetas_actuales)
                palabras_actuales = []
                etiquetas_actuales = []
            continue

        partes = linea.split()
        if len(partes) >= 2:
            palabra = partes[0]
            etiqueta = partes[-1]

            palabras_actuales.append(palabra)
            etiquetas_actuales.append(etiqueta)

    if palabras_actuales:
        oraciones_palabras.append(palabras_actuales)
        oraciones_gold.append(etiquetas_actuales)
```

   Al procesar todas las lineas se generan dos listas sincronizadas, una con las palabras de cada oración y otra con sus etiquetas Gold.

7. **Procesamiento del archivo de salida**

   Se abre el archivo donde se escribirán las predicciones del modelo.

```
with open(archivo_salida, 'w', encoding='utf-8') as salida:
```

   Por cada oración almacenada se recuperan las palabras con sus etiquetas reales y se crea un objeto `Doc` de forma manual para que la tokenización coincida con la del corpus.

```
for i in range(len(oraciones_palabras)):
    palabras = oraciones_palabras[i]
    etiquetas_gold = oraciones_gold[i]

    doc = Doc(nlp.vocab, words=palabras)
```

8. **Aplicación manual del pipeline de Spacy**

   En lugar de usar `nlp(doc)` procesamos los componentes uno a uno evitando que Spacy retokenice los datos.

```
for nombre, proceso in nlp.pipeline:
    doc = proceso(doc)
```

9. **Construcción de las etiquetas predichas**

   Para cada token del documento se genera la etiqueta IOB correspondiente mencionadas en el apartado 1: `B-TIPO`, `I-TIPO` u `O`.

```
etiquetas_pred = []
for token in doc:
    if token.ent_iob_ == 'O':
        etiquetas_pred.append('O')
    else:
        etiquetas_pred.append(f"{token.ent_iob_}-
␣␣␣␣␣␣␣␣{token.ent_type_}")
```

10. **Escritura de las predicciones en el archivo de salida**

    Para cada token de la oración se escribe en el archivo de salida la palabra original con su etiqueta Gold y la etiqueta que ha predecido Spacy:

```
for palabra, etiqueta_gold, etiqueta_pred in zip(
        palabras, etiquetas_gold, etiquetas_pred):
    salida.write(f"{palabra}␣{etiqueta_gold}
␣␣␣␣{etiqueta_pred}\n")

salida.write("\n")
```

11. **Ejecución del script**

    Cargamos el nombre del archivo y el nombre que queramos que tenga el archivo de salida y llamamos al script principal

```
if __name__ == "__main__":
    archivo_entrada = "esp.testb.txt"
    archivo_salida = "salida_es_core_news_sm.txt"
    procesar_archivo_conll(archivo_entrada, archivo_salida)
```

3. Resultados

Primero comentaremos los resultados obtenidos por el modelo base (`es_core_news_sm`), mas adelante lo compararemos con el modelo superior.

> Nota: Estos resultados se guardan en el archivo **resultados_sm** obtenidos al ejecutar `Get-Content salida_es_core_news_sm.txt | python conlleval.py`.

3.1. Métricas globales

Se ha conseguido detectar un total de 3.709 entidades, de las cuales 2.029 son correctas.

Métrica	**Precisión**	**Cobertura (Recall)**	**Medida-F (FB1)**
Global	54.70 %	57.01 %	55.83 %
Exactitud (Accuracy)	93.57 %		

Cuadro 2: Resultados globales de la evaluación (Modelo SM)

3.2. Resultados por categoría

Si nos fijamos en el acierto por cada tipo de entidad observamos un comportamiento desigual del modelo dependiendo de la categoría semántica:

- **PER (Personas):** Categoría con mejor desempeño.
  - Precisión: 64.07 %
  - Recall: 78.37 %
  - **FB1: 70.50 %**
- **ORG (Organizaciones):** Categoría con alta precisión pero baja cobertura.
  - Precisión: 73.60 %
  - Recall: 43.21 %
  - **FB1: 54.46 %**
- **LOC (Localizaciones):** Categoría con alta cobertura, precisión media.
  - Precisión: 50.39 %
  - Recall: 70.85 %
  - **FB1: 58.90 %**
- **MISC (Miscelánea):** Categoría con malos resultados, esto nos puede indicar una discrepancia en la definición de esta categoría entre el modelo y el corpus usado que usaremos más adelante para mejorar los resultados.
  - Precisión: 17.24 %
  - Recall: 23.53 %
  - **FB1: 19.90 %**

4. Análisis de los errores

Al examinarse el fichero de salida superficialmente he identificado algunos errores que se repiten de forma constante:

1. **Bajo rendimiento en MISC:** La categoría MISC tiene un FB1 muy bajo. Esto es posible que se deba a que el corpus incluye en MISC entidades como "Internet"o "Copa del Rey" que Spacy no etiqueta o los etiqueta de una forma diferente.
    - Por ejemplo, "viajesydestinos.com" se predice como O pero esta etiquedada como B-MIST, o "Internautas"que sufre el mismo error de predicción
2. **Confusión PER vs ORG:** Spacy confunde a veces organizaciones que llevan nombres propios, etiquetándolas como personas.
    - Por ejemplo "Savia Amadeus"se predice como nombre de persona en lugar de una empresa
3. **Falsos Positivos en LOC:** La precisión en LOC es baja, lo que nos puede indicar que Spacy etiqueta muchas palabras como lugares que en el corpus no lo son.

4.1. Conclusiones preliminares

Usando el modelo preentrenado `es_core_news_sm` de Spacy sobre el corpus nos da un rendimiento moderado de sobre el 55.83 %.

- El modelo es bueno al identificar personas (PER), alcanzando un 70 % en F-Measure, lo que nos indica que tanto el test y el corpus usan estructuras de nombres propios parecidas y no debería ser el foco principal a la hora de mejorar el modelo.
- El modelo no es bueno al intentar predecir la categoria MISC, es posible que se deba a que definiciones sean mas ambiguas.
- Si quisieramos mejorar los resultados deberiamos re-entrenar el modelo para que aprenda mejor a distingir entre ORG y LOG, y que aprendiese a predecir MISC correctamente.

5. Mejoras y modificaciones en el etiquetado

Tras analizar los resultados iniciales, se implementaran varias mejoras para optimizar el rendimiento del sistema, se irán realizando diferentes iterraciones hasta llegar a un resultado con el que me encuentre satisfecho.

> Nota: A partir de aqui se usara el archivo python *practica_mejorada* ya que experimentare con diferentes soluciones que modificarán lo que se pedia en la primera parte del ejercicio, el codigo final puede diferir con lo mostrado en cada iteracción, ya que en cada una se modifica/borra codigo.

5.1. Fase 1: Actualización del modelo

La primera medida ha consistido en sustituir el modelo básico `es_core_news_sm` por el modelo `es_core_news_lg`. Este modelo incluye vectores de palabras y ha sido entrenado con un volumen de datos mayor.

5.2. Fase 2: Implementación de reglas heurísticas

A pesar de utilizar el modelo *large*, Spacy tiende a etiquetar erróneamente determinantes y preposiciones (como "La", "El", "En") como entidades de tipo **LOC** u **ORG** cuando aparecen con mayúscula al inicio de una frase.

Para intentar solucionar este problema, se ha modificado el código introduciendo un filtro heurístico antes de asignar la etiqueta final.

5.2.1. Implementación del filtro

Primero he definido un conjunto de palabras, luego durante la iteración sobre los tokens si una palabra pertenece a esta lista y ha sido etiquetada como entidad, se fuerza su etiqueta a 'O'.

```
# listado de palabras para el filtro
STOP_WORDS = {'El', 'La', 'Los', 'Las', 'Un', 'Una',
'En', 'De', 'Por', 'Para'}

etiquetas_pred = []
for token in doc:
    # filtro heuristico
    if token.text in STOP_WORDS and token.ent_iob_ != 'O':
        etiquetas_pred.append('O')

    elif token.ent_iob_ == 'O':
        etiquetas_pred.append('O')
    else:
        etiquetas_pred.append(
        f"{token.ent_iob_}-{token.ent_type_}")
```

Listing 1: Filtro heurístico añadido en practica1.py

5.2.2. Resultados obtenidos (Iteración 1)

Tras ejecutar el script de evaluación se obtuvieron los siguientes resultados:

Configuración	Precisión	Recall	F1-Score
Modelo Base (sm)	54.70 %	57.01 %	55.83 %
Modelo (lg) + Filtro Agresivo	**58.61 %**	**64.06 %**	**61.22 %**

Cuadro 3: Resultados tras la primera fase de mejoras

5.2.3. Análisis de errores

Aunque el uso del modelo *large* mejoro la cobertura y el filtro mejoró la precisión general, al analizar el fichero de salida he descubierto que la regla heurísta es demasiado agresiva dejando nuevos errores:

- **Rotura de Entidades Compuestas:** Al filtrar palabras como "La"o "El", se rompieron entidades que comienzan con algúno de los artículos.
- Como ejemplo, la entidad **"La Coruña"** fue etiquetada incorrectamente porque el script forzó el token "La", dejando "Coruña"como una entidad incompleta.
- **Falsos Positivos en Fechas:** Además, he observado que el modelo seguía confundiendo fechas (ej. "23 may") con nombres de personas.

5.3. Segunda iteración: mejora del filtro

Para solucionar estos problemas y aumentar tanto la precisión como la cobertura, se implementó una segunda versión con las siguientes modificaciones lógicas:

1. **Modificación de la lista de Stop Words:** Se eliminaron los artículos de la lista de filtrado para que las entidades compuestas se vuelvan a clasificar de forma correcta.
2. **Filtro de fechas y números:** Se añadio otro filtro para forzar la etiqueta `O` si el token es un dígito o pertenece a la lista definida en el codigo.

5.3.1. Implementación de las mejoras

```
# 1. filtro heuristico acortado
STOP_WORDS = {'En', 'De', 'Por', 'Para', 'Con', 'Sobre'}

# 2. filtro para fechas y números
FECHAS_CONFLICTIVAS = {'enero', 'febrero', 'marzo', ...}

# identificar si el token necesita ser filtrado
es_numero = token.text.isdigit()
es_fecha = token.text.lower() in FECHAS_CONFLICTIVAS
es_stop = token.text in STOP_WORDS
```

```
# si es el caso, se fuerza como O
if (es_numero or es_fecha or es_stop) and token.ent_iob_ != 'O':
    etiquetas_pred.append('O')

elif token.ent_iob_ == 'O':
    etiquetas_pred.append('O')
else:
    etiquetas_pred.append(f"{token.ent_iob_}-{token.ent_type_}")
```

Listing 2: Lógica de filtrado refinada (Iteración 2)

5.4. Resultados tras la segunda mejora

Tras la aplicación de los nuevos filtros, explorando los resultados se puede ver que se han corregido algunos errores que se crearon en la primera versión del filtro (entidades como "La Coruña"), el filtro para el ruido en las fechas no ha dado buenos resultados, por lo que en la siguiente versión sera eliminado.

Configuración	**Precisión**	**Recall**	**F1-Score**
Modelo Base (sm)	54.70 %	57.01 %	55.83 %
Iteración 1 (lg + filtro)	58.61 %	64.06 %	61.22 %
Iteración 2 (lg + filtro mejorado)	**59.38 %**	**64.12 %**	**61.66 %**

Cuadro 4: Comparativa de rendimiento tras la segunda mejora

5.5. Tercera iteración: ajuste fino con lógica condicional

Después de aplicar las dos primeras mejoras aunque el rendimiento general ha aumentado, la categoría **ORG** continua con errores y **MISC** sigue mostrando un rendimiento bajo (F1 $\approx$ 27 %). Al revisar más ejemplos del corpus con ayuda para poder profundizar en el archivo de texto, observe que el corpus CoNLL utiliza criterios muy concretos para anotar entidades, mientras que Spacy trabaja con reglas más generales.

1. **Instituciones genéricas:** El corpus etiqueta sistemáticamente palabras como "Gobierno", "Ministerio"o "Ayuntamiento"como `ORG`. Sin embargo, Spacy muchas veces las interpreta como palabras comunes (`O`) o incluso como `LOC`.

2. **Problemas al aplicar filtros de forma global:** En la iteración anterior, los filtros se aplicaban siempre que coincidía una palabra, sin tener en cuenta su posición o el flujo IOB. Esto provocaba que algunas entidades se rompieran. Por ejemplo, al eliminar "de" en *Banco de España*, Spacy terminaba produciendo entidades incompletas o mal segmentadas.

A partir de estos problemas, he cambiado como se aplican los filtros; en vez de aplicarse de forma global, se aplicarán solo cuando se cumplan ciertas condiciones.

- **Filtro de Stop Words solo al inicio:** Solo se eliminarán artículos y preposiciones si el token esta al inicio de una entidad. Con esto busco evitar casos como en "**La** Coruña"(donde "La" no forma parte del nombre), pero si existen preposiciones de forma interna como en "Ministerio **de** Hacienda"se conserven.

- **Detección y Propagación de Instituciones Genéricas:** Al detectar una palabra clave como "Ministerio.º "Gobierno.en posición de inicio, fuerzo la etiqueta `B-ORG` y activo una bandera para que los tokens siguientes reciban automáticamente `I-ORG`. Esto ayuda a unificar entidades que el modelo de Spacy suele fragmentar o etiquetar mal.

```
ORGANIZACIONES = {'Gobierno', 'Ministerio', ...}
STOP_WORDS = {'El', 'La', 'En', 'De', ...}

correccion_activa = None

if iob == 'B':
    if token.text in ORGANIZACIONES:
        etiquetas_pred.append('B-ORG')
        correccion_activa = 'ORG'

    elif token.text in STOP_WORDS:
        etiquetas_pred.append('O')
        correccion_activa = None

    else:
        etiquetas_pred.append(f"B-{tipo}")
        correccion_activa = None

elif iob == 'I':
    if correccion_activa:
        etiquetas_pred.append(f"I-{correccion_activa}")
    else:
        etiquetas_pred.append(f"I-{tipo}")
```

Listing 3: Lógica condicional de la iteración 3

5.6. Resultados Finales

Al aplicar las reglas se ha conseguido borrar una cantidad moderada de falsos negativos, mejorando el *Recall* y equilibrando un poco las categorías.

Fase	**Precisión**	**Recall**	**F1 Global**	**F1 MISC**
Modelo Base (sm)	54.70 %	57.01 %	55.83 %	19.90 %
Iteración 1 (lg)	58.61 %	64.06 %	61.22 %	29.29 %
Iteración 2 (Reglas)	59.78 %	65.24 %	62.39 %	27.27 %
Iteración 3 (Final)	**60.49 %**	**66.09 %**	**63.17 %**	**29.52 %**

Cuadro 5: Resultado final de las mejoras.

6. Parte opcional: Anotación manual

Para realizar la parte opcional de la práctica, he anotado manualmente un texto corto para poder evaluar el comportamiento en un contexto diferente al del corpus que se nos entrego.

> Nota: El texto podría a llegar a ser demasiado corto para poder obtener unos resultados realmente relevantes, pero debido a la falta de tiempo no se ha podido realizar la anotación IOB en un texto mas largo.

6.1. Selección del texto y metodología

Para realizarlo se ha seleccionado un texto corto relativo a la adquisición de la empresa DeepMind por parte de Google. El texto es corto pero contiene una alta densidad de entidades nombradas que es el requisito del ejercicio.

El proceso seguido ha sido el siguiente:

1. **Anotación manual:** Se ha dividido el texto en tokens y les he asignado manualmente su etiqueta IOB correspondiente.

2. **Procesamiento:** Se ha utilizado el mismo script de Python desarrollado en la parte obligatoria con una pequeña modificación para que en vez de saltos de linea, usara un punto para terminar la frase.

```
for palabra, etiqueta in DATOS_MANUALES:
        if palabra == ".":
            palabras_actuales.append(palabra)
            gold_actuales.append(etiqueta)

            oraciones_palabras.append(palabras_actuales)
            oraciones_gold.append(gold_actuales)

            palabras_actuales = []
            gold_actuales = []
        else:
            palabras_actuales.append(palabra)
            gold_actuales.append(etiqueta)
    ]
```

Listing 4: Modificación del Script principal

3. **Evaluación:** Como en la parte obligatoria se ha usado el Script `conlleval.py` para evaluar el rendimiento del modelo.

6.2. Código y datos

La modificación del Script se ha guardado en `parte_opcional.py`, que contiene los datos manuales anotados y la nueva lógica para poder procesar el texto.

```
DATOS_MANUALES = [
    ("Google", "B-ORG"), ("ha", "O"), ("comprado", "O"),
    ("la", "O"), ("empresa", "O"), ("de", "O"),
    ("inteligencia", "O"), ("artificial", "O"),
    ("DeepMind", "B-ORG"), ("por", "O"), ("400", "O"),
    ("millones", "O"), ("de", "O"), ("libras", "O"),
    (".", "O"), ...
]
```

Listing 5: Datos definidos manualmente

6.3. Resultados de la parte opcional

Al ser un texto formateado correctamente, el modelo ha tenido un buen desempeño, aunque con algunos errores al clasificar tipos concretos.

Métrica	**Precisión**	**Recall**	**Medida-F (FB1)**
Global	71.43 %	71.43 %	71.43 %
Exactitud (Accuracy)	90.74 %		

Cuadro 6: Resultados en el texto opcional

- **LOC y PER (100 %):** El modelo identificó correctamente "Londres" (LOC) y "Larry Page" (PER). Esto nos confirma lo que sabiamos anteriormente de la parte obligatoria, que para entidades el modelo funciona correctamente.
- **ORG (75 %):** Detecto correctamente "Google", "DeepMind" y "Microsoft", pero hubo confusión con otras entidades.
- **MISC (0 %):** Como en la parte obligatoria, esta categoría falló completamente debido a discrepancias de criterio con el modelo.

6.4. Análisis de errores y comparativa

Si analizamos el archivo de salida `salida_opcional.txt`, podemos identificar algunos errores que causan la bajada en la precisión:

1. **Confusión en ORG vs MISC:**
   - **Caso 1:** En el texto manual, "The New York Times" se anoto como **MISC**, pero Spacy lo etiquetó como **ORG**. En este caso el error se podría deber a como anote los datos, ya que se puede considerar tanto una ORG como un periódico.
   - **Caso 2:** El modelo etiquetó "Facebook" como **MISC**, mientras que es una **ORG**.