**Tarea 1 – Detección de Entidades Nombradas.**

### Adrián José Zapater Reig

## Objetivos:

* Análisis de la tarea CoNLL 2002.
* Solución planteada por el estudiante.
* Presentación del texto de prueba.
* Análisis del código fuente.
* Análisis de los resultados de evaluación.
* Análisis de los errores de etiquetado y sus causas.
* Mejoras propuestas.
* Anexos:
  + Anexo I - Jupyter Notebook con la solución del estudiante.
  + Anexo II - conlleval.py

## Análisis de la tarea CoNLL 2002:

El objetivo de la tarea [CoNLL 2002](https://www.aclweb.org/anthology/W02-2024/) es plantear un problema de identificación de entidades (NER) multi-lenguaje y un marco común de evaluación con el que medir las soluciones planteadas.

La tarea proporciona un total de 6 ficheros, 3 para cada idioma: Español y Holandés, compuestos por un set de entrenamiento, un set de desarrollo y un set de prueba. Los ficheros están anotados con una variante del esquema IOB que categoriza las entidades en 4 tipos: Personas (PER), Organizaciones (ORG), Lugares (LOC) y Miscelánea (MISC).

El marco de evaluación se compone de un baseline y un set de prueba. Las soluciones al problema se ejecutan sobre el set de prueba para obtener la precisión, cobertura y medida-F. Estas 3 medidas se comparan contra el baseline para comprobar si son una solución real (si una solución obtiene peor resultado que el baseline no es realmente una solución al problema) o no.

Los 12 sistemas solución al problema planteados en el [artículo](https://www.aclweb.org/anthology/W02-2024.pdf) obtienen una mejora sobre baseline.

## Solución planteada por el estudiante:

Se ha desarrollado un notebook de Jupyter con Kernel de Python3 que, haciendo uso de un modelo pre-entrenado de [spaCy](https://spacy.io/) y su api, genere anotaciones del esquema IOB sobre el set de prueba [esp.testb](https://www.clips.uantwerpen.be/conll2002/ner/data/esp.testb) y posteriormente compare los resultados obtenidos contra las anotaciones del set de prueba calculando la precisión, cobertura y medida-F con el script conlleval.py.

La solución planteada se compone de 5 fases:

### 1 – Preparación de entorno de trabajo:

En esta fase se descargan los ficheros de datos correspondientes, se descarga el modelo de spaCy y se inicializa el módulo de spaCy.

### 2 – Preparación de datos de entrada:

El modelo de spaCy espera recibir una cadena de texto en una única línea con espacios entre cada palabra. En primer lugar, se debe parsear el fichero de set de prueba y procesarlo para generar la entrada que espera el modelo de spaCy.

### 3 – Ejecución del modelo:

Una vez que tenemos la entrada preparada, se debe alimentar el modelo y recuperar los resultados.

### 4 – Preparación de fichero para comparar:

El script conlleval.py espera recibir una línea por palabra (o carácter) con 3 columnas separadas por un espacio: *palabra, anotación de referencia, anotación generada por nuestro modelo*. Para poder lanzar el script necesitamos procesar la salida del modelo junto con el set de prueba y generar el formato esperado.

### 5 – Cálculo de métricas de evaluación:

Por último, se debe pasar la salida al script colleval.py a través de la salida estándar.

## Presentación del texto de prueba:

El texto viene anotado con una variante del esquema IOB en el que se categoriza cada palabra como O (fuera de la entidad), B(Inicio de la entidad) o I(Palabra intermedia de la Entidad). Además, cada entidad está tipificada como una de 4 tipos: Personas [PER], Organizaciones [ORG], Lugares [LOC] y Miscelánea [MISC]. Sólo las entidades (B o I) están tipificadas, las palabras que están fuera de una entidad (O) no tienen tipo.

Podemos encontrar el set de prueba en el siguiente [link](https://www.clips.uantwerpen.be/conll2002/ner/data/esp.testb). Analicemos las primeras 10 líneas:

Tabla 1

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | La B-LOC  Coruña I-LOC  , O  23 O  may O  ( O  EFECOM B-ORG  ) O  . O  - O |

Las primeras 2 línea identifican la entidad “La Coruña” de tipo Lugar [LOC]. “La” está categorizada como inicio de entidad (B) y “Coruña” como palabra intermedia de entidad (I).

Las líneas 3 y 4 contienen 2 palabras que no son ni pertenecen a una entidad, por lo que están categorizadas como fuera de entidad (O) y no tienen tipo (LOC, PER, ORG y MISC).

La línea 9 es diferente al resto, se trata de una línea vacía sin categoría ni tipo. Esta línea en blanco representa el final de una frase.

## Análisis del código fuente:

El Anexo I contiene el código fuente con la solución del estudiante. También podemos encontrar una copia ejecutable en el ZIP adjunto a la tarea.

### 1 – Preparación de entorno de trabajo:

En primer lugar, se debe descargar el fichero con el set de prueba y depositarlo en la carpeta “src” del espacio de trabajo. Esta carpeta alojará la entrada, salida, log y script de evaluación.

Para descargar el fichero se ha hecho uso de la herramienta *wget*.

En segundo lugar, es necesario descargar [el modelo en español de spaCy](https://spacy.io/models/es) e instalarlo como paquete en Python. Puesto que este notebook corre sobre un Kernel de Python emulado por anaconda, no basta con ejecutar “*python -m spacy download es*”, se debe hacer uso la api *cli* de spaCy:

Tabla 2

|  |
| --- |
| import spacy.cli  spacy.cli.download("es") |

Una vez ejecutado esto, se descargará e instalará el modelo en Español en el python de nuestra Kernel. Podemos proceder a cargar el modelo y así terminar de inicializar spaCy:

Tabla 3

|  |
| --- |
| es\_model = spacy.load('es') |

### 2 – Preparación de datos de entrada:

El set de prueba está descargado en la carpeta *src*, por lo que abriremos el fichero e iremos leyéndolo línea a línea.

Tabla 4

|  |
| --- |
| esp\_test\_raw\_data = open('./src/esp.testb', 'r')  for i, line in enumerate(esp\_test\_raw\_data):  # Procesamiento de cada línea del fichero. |

Cada línea contendrá 2 columnas separadas por un espacio (*palabra“ “anotación*) por lo que podemos hacer uso de una expresión regular para separar cada línea en una palabra y su anotación y guardar en una variable (*text)* las palabras separadas por espacios:

Tabla 5

|  |
| --- |
| # Módulo de expresiones regulares.  import re  anotated\_word = re.split('\s', line)  # Palabra  word = anotated\_word[0]  # Anotación  anotation = anotated\_word[1] # Anotación  # Resultado  text += word + ' ' |

En la fase 4 nos hará falta tener disponible la información del fichero de referencia junto con sus anotaciones, para facilitar esta tarea, vamos a almacenar cada palabra y anotación en un diccionario (*anotated\_reference*) utilizando la posición de la palabra en el fichero como clave:

Tabla 6

|  |
| --- |
| anotated\_reference = {}  for i, line in enumerate(esp\_test\_raw\_data):  anotated\_word = re.split('\s', line)  word = anotated\_word[0]  anotation = anotated\_word[1]  # Save original Solution  anotated\_reference[i] = (word, anotation)  # Prepare text input for spaCy's NER model.  text += word + ' '  # Ya no necesitamos utilizar el fichero mas, es buena práctica cerrarlo para liberar memoria.  esp\_test\_raw\_data.close() |

### 3 – Ejecución del modelo:

La api de spaCy es muy fácil de utilizar, para alimentar el modelo sólo debemos pasar la variable “text” como parámetro al modelo inicializado en la fase 1:

Tabla 7

|  |
| --- |
| anotated\_doc\_es = es\_model(text) |

El resultado (*anotated\_doc\_es*) contiene el texto anotado con el esquema IOB. Guardaremos la salida del modelo en un diccionario (*anotated\_output*) con el siguiente formato:

Clave: id del token (posición de la palabra)

Valor: Tupla (Palabra, Anotación)

Spacy guarda por separado la categoría de entidad y el tipo en los atributos *ent\_iob\_* y *ent\_type\_* respectivamente, por lo que la *anotación* se compondrá por “*ent\_iob\_-ent\_type\_*” si se trata de una entidad B o I y sólo *ent\_iob\_* para las entidades O.

Tabla 8

|  |
| --- |
| anotated\_output = {}  for i, token in enumerate(anotated\_doc\_es):  anotated\_output[i] = (token.text, token.ent\_iob\_ + "-" + token.ent\_type\_ if token.ent\_iob\_ != 'O' else token.ent\_iob\_) |

### 4 – Preparación de fichero para comparar:

La entrada que espera el script colleval.py debe tener 3 columnas separadas por un espacio en cada línea:

Palabra - Anotación de Referencia – Anotación a Evaluar

Para construir esta estructura haremos uso de los 2 diccionarios que hemos ido construyendo durante el proceso: *anotated\_reference* y *anotated\_output*. Se deben recorrer los dos diccionarios a la vez almacenando las anotaciones de cada diccionario y la palabra para construir cada línea cómo: “$palabra $anot\_ref $anot\_out \n”.

Tabla 9

|  |
| --- |
| out\_file = open("./src/esp\_out.testb", "w")  for idx in anotated\_reference:  word = anotated\_reference[idx][0]  ref = anotated\_reference[idx][1]  asig = anotated\_output[idx][1]  out\_file.write('{} {} {}\n'.format(word, ref, asig))  out\_file.close() |

¿Pero qué pasa cuando las anotaciones no coinciden porque el modelo de spaCy ha procesado una palabra como 2 o ha juntado 2 palabras en 1? Para solucionar estos problemas, se ha decidido utilizar un contador separado para cada diccionario (idx y j) y si descubrimos que la palabra no coincide en ambos diccionarios, avanzamos una posición en uno de los diccionarios y comprobamos de nuevo. Para evitar avanzar infinitamente utilizamos una variable (*max\_error*) como límite de avaces que se pueden hacer para un diccioinario. El código queda así:

Tabla 10

|  |
| --- |
| j = 0  max\_errors = 10  for idx in anotated\_reference:  word = anotated\_reference[idx][0]  ref = anotated\_reference[idx][1]    # If there is a miss match, we try with the next elements of anotated\_output.  while(word.strip() != anotated\_output[j][0].strip()):  log\_file.write('Word missmatch: ref:{} asig: {}\n'.format(word, anotated\_output[j][0]))  j += 1  # If there are more  if j - idx > max\_errors:  log\_file.write('Max Errors reached ({})\n'.format(max\_errors))  j = idx  break    asig = anotated\_output[j][1]  out\_file.write('{} {} {}\n'.format(word, ref, asig))  j += 1 |

Nota: Hacemos un strip() de la palabra para evitar falsos negativos al comparar 2 palabras iguales pero donde una de las 2 tiene un espacio al inicio o final.

Aun queda un caso mas que cubrir: Los espacios de final de oración. El modelo de spaCy no parece que haya interpretado los espacios como final de oración y les ha asignado una categoría: O. Para no generar una salida errónea, debemos controlar este escenario. Para ellos comprobaremos si la palabra es vacía o está formada por espacios y la dejaremos en blanco si esto sucede:

Tabla 11

|  |
| --- |
| out\_file = open("./src/esp\_out.testb", "w")  j = 0  max\_errors = 10  for idx in anotated\_reference:  word = anotated\_reference[idx][0]  ref = anotated\_reference[idx][1]    # If there is a miss match, we try with the next elements of anotated\_output.  while(word.strip() != anotated\_output[j][0].strip()):  log\_file.write('Word missmatch: ref:{} asig: {}\n'.format(word, anotated\_output[j][0]))  j += 1  # If there are more  if j - idx > max\_errors:  log\_file.write('Max Errors reached ({})\n'.format(max\_errors))  j = idx  break    asig = '' if word.strip() == '' else anotated\_output[j][1]  out\_file.write('{} {} {}\n'.format(word, ref, asig))  j += 1    out\_file.close() |

### 5 – Cálculo de métricas de evaluación:

Una vez que hemos generado la salida correcta y la hemos almacenado en el fichero *esp\_out.testb*, pasaremos el contenido del mismo al script conlleval.py a través de la salida estándar.

Tabla 12

|  |
| --- |
| ! python ./src/conlleval.py < ./src/esp\_out.testb |

## Resultados de evaluación:

Tabla 13

|  |
| --- |
| processed 53049 tokens with 3559 phrases; found: 3802 phrases; correct: 316.  accuracy: 8.90%; (non-O)  accuracy: 80.70%; precision: 8.31%; recall: 8.88%; FB1: 8.59  LOC: precision: 7.52%; recall: 10.06%; FB1: 8.60 1450  MISC: precision: 2.26%; recall: 4.41%; FB1: 2.99 663  ORG: precision: 12.31%; recall: 7.50%; FB1: 9.32 853  PER: precision: 10.41%; recall: 11.84%; FB1: 11.08 836 |

Los resultados son muy malos, pero era de esperar teniendo en cuenta que hemos utilizado un modelo “out of the box”. Los resultados nos dicen que tenemos una accuracy del 80.7 % sobre el total de palabras (I,B y O), que a primera vista parece que es un buen resultado, per tras ver que tenemos una accuracy del 8.9% de las palabras que si son entidades (I y B), sacamos como conclusión que nuestro modelo sólo es bueno para detectar las palabras que están fuera de las entidades (O) y no para detectar y clasificar entidades.

## Anexos:

### Anexo I







### Anexo II:

Tabla 14

|  |
| --- |
| """  This script applies to IOB2 or IOBES tagging scheme.  If you are using a different scheme, please convert to IOB2 or IOBES.  IOB2:  - B = begin,  - I = inside but not the first,  - O = outside  e.g.  John lives in New York City .  B-PER O O B-LOC I-LOC I-LOC O  IOBES:  - B = begin,  - E = end,  - S = singleton,  - I = inside but not the first or the last,  - O = outside  e.g.  John lives in New York City .  S-PER O O B-LOC I-LOC E-LOC O  prefix: IOBES  chunk\_type: PER, LOC, etc.  """  from \_\_future\_\_ import division, print\_function, unicode\_literals  import sys  from collections import defaultdict  def split\_tag(chunk\_tag):  """  split chunk tag into IOBES prefix and chunk\_type  e.g.  B-PER -> (B, PER)  O -> (O, None)  """  if chunk\_tag == 'O':  return ('O', None)  #return chunk\_tag.split('-', maxsplit=1)  return chunk\_tag.split('-')  def is\_chunk\_end(prev\_tag, tag):  """  check if the previous chunk ended between the previous and current word  e.g.  (B-PER, I-PER) -> False  (B-LOC, O) -> True  Note: in case of contradicting tags, e.g. (B-PER, I-LOC)  this is considered as (B-PER, B-LOC)  """  prefix1, chunk\_type1 = split\_tag(prev\_tag)  prefix2, chunk\_type2 = split\_tag(tag)  if prefix1 == 'O':  return False  if prefix2 == 'O':  return prefix1 != 'O'  if chunk\_type1 != chunk\_type2:  return True  return prefix2 in ['B', 'S'] or prefix1 in ['E', 'S']  def is\_chunk\_start(prev\_tag, tag):  """  check if a new chunk started between the previous and current word  """  prefix1, chunk\_type1 = split\_tag(prev\_tag)  prefix2, chunk\_type2 = split\_tag(tag)  if prefix2 == 'O':  return False  if prefix1 == 'O':  return prefix2 != 'O'  if chunk\_type1 != chunk\_type2:  return True  return prefix2 in ['B', 'S'] or prefix1 in ['E', 'S']  def calc\_metrics(tp, p, t, percent=True):  """  compute overall precision, recall and FB1 (default values are 0.0)  if percent is True, return 100 \* original decimal value  """  precision = tp / p if p else 0  recall = tp / t if t else 0  fb1 = 2 \* precision \* recall / (precision + recall) if precision + recall else 0  if percent:  return 100 \* precision, 100 \* recall, 100 \* fb1  else:  return precision, recall, fb1  def count\_chunks(true\_seqs, pred\_seqs):  """  true\_seqs: a list of true tags  pred\_seqs: a list of predicted tags  return:  correct\_chunks: a dict (counter),  key = chunk types,  value = number of correctly identified chunks per type  true\_chunks: a dict, number of true chunks per type  pred\_chunks: a dict, number of identified chunks per type  correct\_counts, true\_counts, pred\_counts: similar to above, but for tags  """  correct\_chunks = defaultdict(int)  true\_chunks = defaultdict(int)  pred\_chunks = defaultdict(int)  correct\_counts = defaultdict(int)  true\_counts = defaultdict(int)  pred\_counts = defaultdict(int)  prev\_true\_tag, prev\_pred\_tag = 'O', 'O'  correct\_chunk = None  for true\_tag, pred\_tag in zip(true\_seqs, pred\_seqs):  if true\_tag == pred\_tag:  correct\_counts[true\_tag] += 1  true\_counts[true\_tag] += 1  pred\_counts[pred\_tag] += 1  \_, true\_type = split\_tag(true\_tag)  \_, pred\_type = split\_tag(pred\_tag)  if correct\_chunk is not None:  true\_end = is\_chunk\_end(prev\_true\_tag, true\_tag)  pred\_end = is\_chunk\_end(prev\_pred\_tag, pred\_tag)  if pred\_end and true\_end:  correct\_chunks[correct\_chunk] += 1  correct\_chunk = None  elif pred\_end != true\_end or true\_type != pred\_type:  correct\_chunk = None  true\_start = is\_chunk\_start(prev\_true\_tag, true\_tag)  pred\_start = is\_chunk\_start(prev\_pred\_tag, pred\_tag)  if true\_start and pred\_start and true\_type == pred\_type:  correct\_chunk = true\_type  if true\_start:  true\_chunks[true\_type] += 1  if pred\_start:  pred\_chunks[pred\_type] += 1  prev\_true\_tag, prev\_pred\_tag = true\_tag, pred\_tag  if correct\_chunk is not None:  correct\_chunks[correct\_chunk] += 1  return (correct\_chunks, true\_chunks, pred\_chunks,  correct\_counts, true\_counts, pred\_counts)  def get\_result(correct\_chunks, true\_chunks, pred\_chunks,  correct\_counts, true\_counts, pred\_counts, verbose=True):  """  if verbose, print overall performance, as well as preformance per chunk type;  otherwise, simply return overall prec, rec, f1 scores  """  # sum counts  sum\_correct\_chunks = sum(correct\_chunks.values())  sum\_true\_chunks = sum(true\_chunks.values())  sum\_pred\_chunks = sum(pred\_chunks.values())  sum\_correct\_counts = sum(correct\_counts.values())  sum\_true\_counts = sum(true\_counts.values())  nonO\_correct\_counts = sum(v for k, v in correct\_counts.items() if k != 'O')  nonO\_true\_counts = sum(v for k, v in true\_counts.items() if k != 'O')  chunk\_types = sorted(list(set(list(true\_chunks) + list(pred\_chunks))))  # compute overall precision, recall and FB1 (default values are 0.0)  prec, rec, f1 = calc\_metrics(sum\_correct\_chunks, sum\_pred\_chunks, sum\_true\_chunks)  res = (prec, rec, f1)  if not verbose:  return res  # print overall performance, and performance per chunk type    print("processed %i tokens with %i phrases; " % (sum\_true\_counts, sum\_true\_chunks), end='')  print("found: %i phrases; correct: %i.\n" % (sum\_pred\_chunks, sum\_correct\_chunks), end='')    print("accuracy: %6.2f%%; (non-O)" % (100\*nonO\_correct\_counts/nonO\_true\_counts))  print("accuracy: %6.2f%%; " % (100\*sum\_correct\_counts/sum\_true\_counts), end='')  print("precision: %6.2f%%; recall: %6.2f%%; FB1: %6.2f" % (prec, rec, f1))  # for each chunk type, compute precision, recall and FB1 (default values are 0.0)  for t in chunk\_types:  prec, rec, f1 = calc\_metrics(correct\_chunks[t], pred\_chunks[t], true\_chunks[t])  print("%17s: " %t , end='')  print("precision: %6.2f%%; recall: %6.2f%%; FB1: %6.2f" %  (prec, rec, f1), end='')  print(" %d" % pred\_chunks[t])  return res  # you can generate LaTeX output for tables like in  # http://cnts.uia.ac.be/conll2003/ner/example.tex  # but I'm not implementing this  def evaluate(true\_seqs, pred\_seqs, verbose=True):  (correct\_chunks, true\_chunks, pred\_chunks,  correct\_counts, true\_counts, pred\_counts) = count\_chunks(true\_seqs, pred\_seqs)  result = get\_result(correct\_chunks, true\_chunks, pred\_chunks,  correct\_counts, true\_counts, pred\_counts, verbose=verbose)  return result  def evaluate\_conll\_file(fileIterator):  true\_seqs, pred\_seqs = [], []    for line in fileIterator:  cols = line.strip().split()  # each non-empty line must contain >= 3 columns  if not cols:  true\_seqs.append('O')  pred\_seqs.append('O')  elif len(cols) < 3:  raise IOError("conlleval: too few columns in line %s\n" % line)  else:  # extract tags from last 2 columns  true\_seqs.append(cols[-2])  pred\_seqs.append(cols[-1])  return evaluate(true\_seqs, pred\_seqs)  if \_\_name\_\_ == '\_\_main\_\_':  """  usage: conlleval < file  """  evaluate\_conll\_file(sys.stdin) |