PROCESAMIENTO DE LENGUAJE NATURAL

PROYECTO 2020-2021

Objetivos

En este proyecto vamos a abordar la similitud entre textos, siguiendo la tarea internacional competitiva planteada desde 2012 a 2017. Supone plantearnos un reto de inicio a la investigación en donde poner a prueba los conocimientos y habilidades que hemos ido adquiriendo a lo largo del curso, y es una ocasión también para ver desde una perspectiva profesional las actividades de investigación en el ámbito internacional.

Estimar la similitud entre dos textos es un elemento clave en múltiples problemas de acceso a la información textual. Por ejemplo, la semejanza entre una consulta y los documentos potencialmente relevantes en un buscador, la comparación de fragmentos de diferentes documentos en tareas de resumen automático, sistemas de detección de copias, la identificación de *trending topics* en redes sociales, y un largo etcétera. La tesis de Aitor González-Aguirre (Universidad del Pais Vasco) ofrece una perspectiva global del problema, sus aplicaciones, las aproximaciones existentes y datos para su evaluación.

Básicamente, podemos distinguir dos tipos de aproximaciones: distribucional y ontológica. En las soluciones basadas en representación distribucional, se entrenan modelos de representación semántica estudiando las distribuciones de palabras y sus co-ocurrencias en un corpus. En este proyecto compararemos aproximaciones basadas en ambos paradigmas. Dentro del paradigma ontológico, emplearemos Wordnet como recurso léxico-semántico para estudiar la proximidad entre palabras y extenderla a similitud entre frases. Como aproximación distribucional, emplearemos la representación lexico-distribucional Word2Vec.

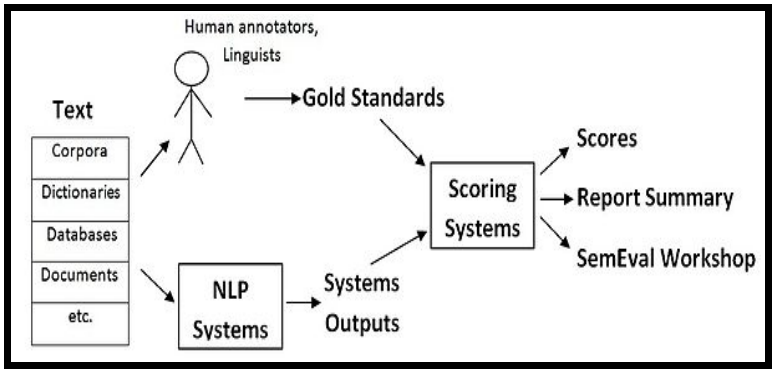
Enunciado del proyecto

Este proyecto consiste en diseñar e implementar una batería de métodos que calculen la similitud entre dos frases. Para ello, partiremos de los diferentes métodos de similitud léxica basada en wordnet disponibles en las librerías de NLTK, su extensión a similitud entre frases, y su efectividad evaluada en términos de correlación Pearson sobre un corpus de pares de frases cuya similitud ha sido anotada manualmente.

En este documento se dan una serie de directrices para la descarga del corpus, necesidades de preprocesamiento de texto con las herramientas disponibles en NLTK, aproximaciones de similitud léxica a implementar y métodos que hemos de aplicar para su extensión a nivel de frase. También se dan una serie de indicaciones sobre los aspectos a analizar y evaluar empíricamente, así como el material a entregar.

Paso 1: Descargando el corpus

Para la realización de nuestros experimentos, emplearemos el corpus de evaluación de Semantic Textual Similarity. Desde este enlace puede descargarse el corpus, consistente en 8628 pares de textos cuya similitud ha sido anotada manualmente. Las instancias están categorizadas según diferentes tipos de problema: comparación entre noticias, descriptores de imágenes, etc. Estos datos han sido empleados a lo largo de diferentes campañas de evaluación denominadas *Semantic Textual Similarity* en el contexto de las conferencias *Semeval*, donde grupos de investigación de todo el mundo han comparado sus aproximaciones. En este enlace está disponible un artículo que describe la campaña realizada en 2017.



Paso 2: Preprocesamiento

El segundo paso a realizar consiste en preprocesar los textos. En este proyecto emplearemos de nuevo Python y las librerías de NLTK. En el capítulo 3 del libro de NLTK se describen las herramientas necesarias para leer un fichero, y procesarlo. El formato de los ficheros del corpus consiste en columnas separadas por tabulador, que incluyen el subconjunto al que pertenece la muestra (por ejemplo, images, MSRvid, etc), la anotación de similitud realizada por humanos, y el texto de ambas frases. El primer paso será capturar cada uno de estos elementos. Al final esta etapa deberíamos tener un programa en Python que lea los archivos del corpus y que almacenara los string.

3 Processing Raw Text

The most important source of texts is undoubtedly the Web. It's convenient to have existing text collections to explore, such as the corpora we saw in the previous chapters. However, you probably have your own text sources in mind, and need to learn how to access them.

The goal of this chapter is to answer the following questions:

1. How can we write programs to access text from local files and from the web, in order to get hold of an unlimited range of language material?
2. How can we split documents up into individual words and punctuation symbols, so we can carry out the same kinds of analysis we did with text corpora in earlier chapters?
3. How can we write programs to produce formatted output and save it in a file?

In order to address these questions, we will be covering key concepts in NLP, including tokenization and stemming. Along the way you will consolidate your Python knowledge and learn about strings, files, and regular expressions. Since so much text on the web is in HTML format, we will also see how to dispense with markup.

Note

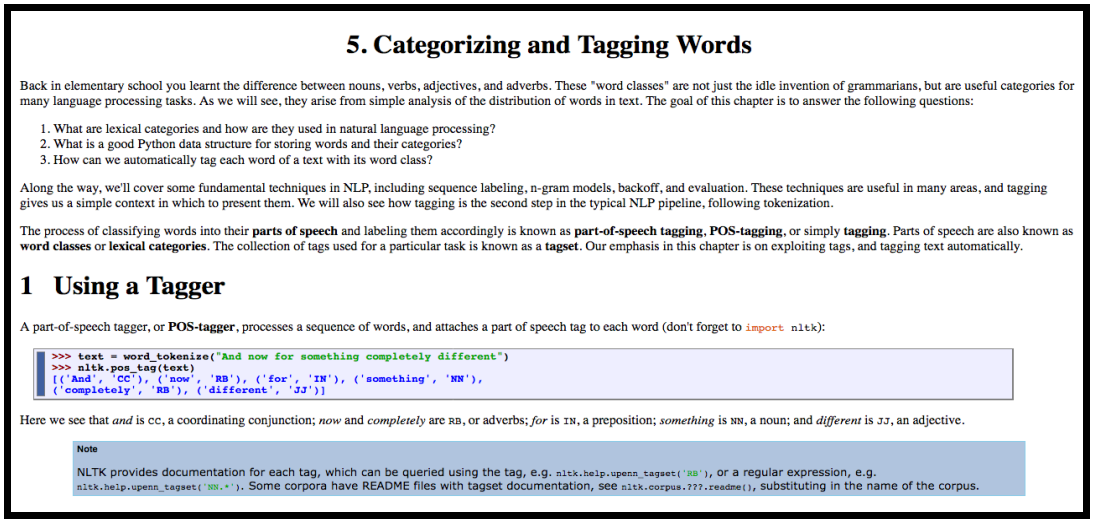
Important: From this chapter onwards, our program samples will assume you begin your interactive session or your program with the following import statements:

```
>>> from __future__ import division  # Python 2 users only
>>> import nltk, re, pprint
>>> from nltk import word_tokenize
```

El siguiente paso será *tokenizar* los textos, separando las palabras entre si, eliminando signos de puntuación, etc. Para ello, emplearemos patrones al igual que en la realización de la primera práctica. También deberemos aplicar otros procesos como la eliminación de mayúsculas, símbolos extraños, etc. Al final de este proceso nuestro programa debería almacenar en dos variables dos listas de términos por cada par de frases del corpus.

Paso 3: Filtrado gramatical

A la hora de comparar semánticamente dos frases, no todas las palabras tienen la misma carga. Por ejemplo, términos con ciertas etiquetas gramaticales como nombre o verbos ('NOUN' y 'VERB' según el etiquetador en inglés de NLTK) deberían tener más peso que otros términos de la frase. En esta práctica, usaremos el POS-tagger de NLTK para reconocer categorías gramaticales y *jugar* con diferentes filtros comparando los resultados. Es decir, eliminaremos diferentes palabras según su categoria gramatical en el cálculo de la similitud. En el capítulo 5 del libro de NLTK aparecen ejemplos de cómo ejecutar un analizador gramatical.



5. Categorizing and Tagging Words

Back in elementary school you learnt the difference between nouns, verbs, adjectives, and adverbs. These "word classes" are not just the idle invention of grammarians, but are useful categories for many language processing tasks. As we will see, they arise from simple analysis of the distribution of words in text. The goal of this chapter is to answer the following questions:

1. What are lexical categories and how are they used in natural language processing?
2. What is a good Python data structure for storing words and their categories?
3. How can we automatically tag each word of a text with its word class?

Along the way, we'll cover some fundamental techniques in NLP, including sequence labeling, n-gram models, backoff, and evaluation. These techniques are useful in many areas, and tagging gives us a simple context in which to present them. We will also see how tagging is the second step in the typical NLP pipeline, following tokenization.

The process of classifying words into their **parts of speech** and labeling them accordingly is known as **part-of-speech tagging**, **POS-tagging**, or simply **tagging**. Parts of speech are also known as **word classes** or **lexical categories**. The collection of tags used for a particular task is known as a **tagset**. Our emphasis in this chapter is on exploiting tags, and tagging text automatically.

1 Using a Tagger

A part-of-speech tagger, or **POS-tagger**, processes a sequence of words, and attaches a part of speech tag to each word (don't forget to `import nltk`):

```
>>> text = word_tokenize("And now for something completely different")
>>> nltk.pos_tag(text)
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'),
('completely', 'RB'), ('different', 'JJ')]
```

Here we see that *and* is CC, a coordinating conjunction; *now* and *completely* are RB, or adverbs; *for* is IN, a preposition; *something* is NN, a noun; and *different* is JJ, an adjective.

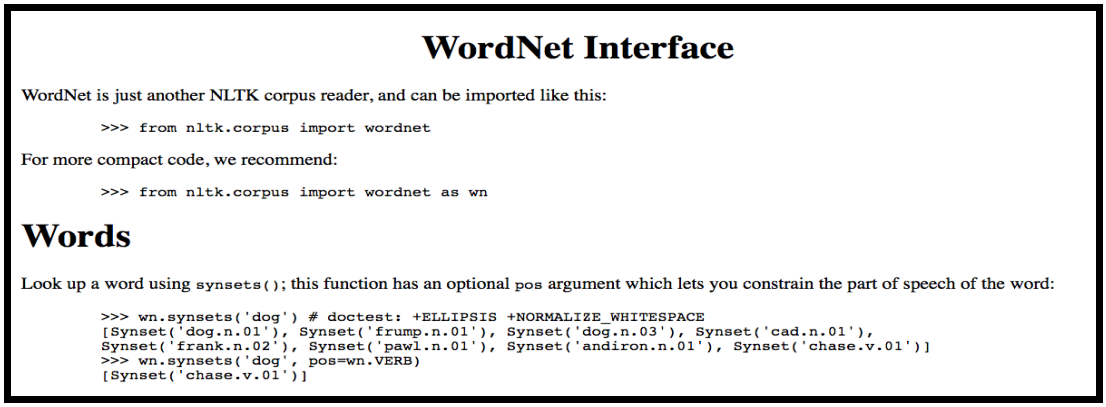
Note

NLTK provides documentation for each tag, which can be queried using the tag, e.g. `nltk.help.upenn_tagset('RB')`, or a regular expression, e.g. `nltk.help.upenn_tagset('NN.*')`. Some corpora have README files with tagset documentation, see `nltk.corpus.???.readme()`, substituting in the name of the corpus.

Al final de este proceso nuestro programa debería almacenar en dos variables dos listas de términos por cada par de frases del corpus, con diferentes filtros según categorías gramaticales.

Paso 4: Similitud léxica basada en ontologías.

En este proyecto partiremos de la similitud entre palabras para luego estimar a partir de esta información la similitud entre frases. Existen múltiples técnicas para estudiar la similitud semántica a nivel léxico. Dentro del paradigma basado en ontologías, emplearemos herramientas disponibles en NLTK que calculan la distancia entre palabras en términos de longitud de caminos posibles entre nodos de la ontología Wordnet. En el siguiente enlace se describe la interfaz de NLTK para el uso de wordnet, así como funciones que estiman la similitud léxica. Éstas incluyen: longitud del camino más corto, Leacock-Chodorow, Wu-Palmer, Resnik, Jiang-Conrath, y la similitud de Lin.



WordNet Interface

WordNet is just another NLTK corpus reader, and can be imported like this:

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

For more compact code, we recommend:

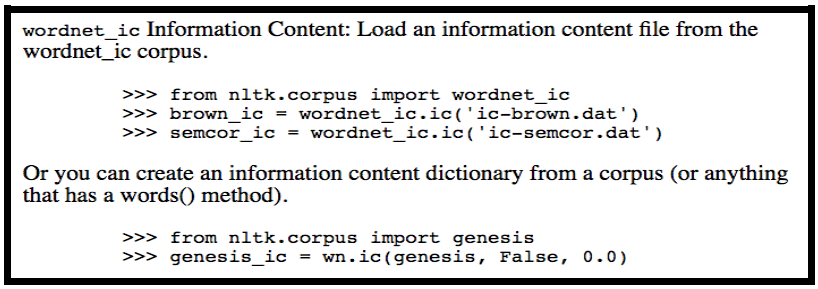
```
>>> from nltk.corpus import wordnet as wn
```

Words

Look up a word using `synsets()`; this function has an optional `pos` argument which lets you constrain the part of speech of the word:

```
>>> wn.synsets('dog') # doctest: +ELLIPSIS +NORMALIZE_WHITESPACE
[Synset('dog.n.01'), Synset('frump.n.01'), Synset('dog.n.03'), Synset('cad.n.01'),
Synset('frank.n.02'), Synset('pawl.n.01'), Synset('andiron.n.01'), Synset('chase.v.01')]
>>> wn.synsets('dog', pos=wn.VERB)
[Synset('chase.v.01')]
```

Algunas de estas funciones de similitud se basan en el concepto de *Information Content* (IC), relacionado con la frecuencia de cada nodo de wordnet en un corpus de texto. NLTK permite "entrenar" IC a partir de diferentes corpora. Parte de este proyecto consistirá en estudiar los resultados cuando se aplican diferentes corpus para entrenar.



`wordnet_ic` Information Content: Load an information content file from the wordnet_ic corpus.

```
>>> from nltk.corpus import wordnet_ic
>>> brown_ic = wordnet_ic.ic('ic-brown.dat')
>>> semcor_ic = wordnet_ic.ic('ic-semcor.dat')
```

Or you can create an information content dictionary from a corpus (or anything that has a words() method).

```
>>> from nltk.corpus import genesis
>>> genesis_ic = wn.ic(genesis, False, 0.0)
```

Al final de este paso, nuestro programa debería ser capaz de calcular la similitud entre palabras de una y otra frase según diferentes criterios de similitud.

Paso 5: Similitud léxica basada en semántica distribucional.

Implementaremos en esta práctica una aproximación basada en modelos distribucionales. En capítulo 6 de la tercera edición de "Speech and Language Processing" (borrador disponible en internet) puede encontrarse una descripción de los mecanismos de representación semántico-léxica sobre la base de modelos distribucionales. Básicamente, las palabras quedan representadas como puntos en un espacio multidimensional, según sus propiedades distribucionales en grandes muestras de textos. Se fundamenta en la llamada hipótesis distribucional: «elementos lingüísticos con distribuciones similares tienen significados similares».

En este proyecto, haremos uso de representaciones ya entrenadas, es decir, un recurso con pares palabra-vector. En este enlace puede descargarse las representaciones distribucionales basadas en el modelo Word2Vec, consistente en 3 millones de palabras y sintagmas representados en vectores de 300 dimensiones.

Las medidas de similitud estandar entre dos términos en el espacio distribucional son, la similitud coseno, la distancia euclidea y el producto escalar de ambos vectores. La distancia coseno considera únicamente el ángulo entre los vectores. El producto escalar en Word2Vec aproxima la información mutua estadística entre pares de palabras.

Al final de este paso, nuestro programa debería de ser capaz de calcular estas tres distancias entre palabras de ambos textos.

Paso 6: Similitud a nivel de texto.

El paso siguiente es estimar la proximidad entre frases. Emplearemos para ello dos aproximaciones sencillas. La primera es el solapamiento de palabras descrito en la página 26 de la tesis de Aitor González-Aguirre. Esta similitud depende del número de términos de cada frase y del número de términos que se pueden alinear entre frases. El alineamiento básico es la coincidencia de palabras. A partir de esto, podemos relajar la condición de alineamiento imponiendo un umbral de similitud, empleando las medidas de similitud léxica desarrolladas en los anteriores apartados.

Para cada medida de similitud léxica, podemos fijar dichos umbrales de diferentes formas. Una de ellas es seleccionando un conjunto de pares de palabras que consideremos que tienen la similitud mínima, calcular su similitud según cada medida y establecer esto como umbral. La otra opción es realizar una estadística de similitudes y poner umbrales porcentuales. Por ejemplo, la similitud mínima para estar entre el 20% de términos más parecidos entre sí. Entonces podremos aplicar dicho umbral porcentual a cada una de las medidas.

La segunda medida de similitud textual que usaremos es la de Milhacea descrita también en la tesis de Aitor González-Aguirre. Esta medida de similitud requiere el cálculo de idf del término, que se corresponde con el menos logaritmo de la frecuencia de la palabra (https://es.wikipedia.org/wiki/Tf-idf) Este número lo podemos aproximar consultando su frecuencia en un corpus de NLTK y calculando el logaritmo del tamaño del corpus dividido por la frecuencia de la palabra. En el capítulo 2 del libro de NLTK se muestran funciones para realizar este tipo de operaciones sobre un corpus.

Paso 7: Evaluación y análisis de resultados.

Para evaluar los resultados emplearemos correlación Pearson, que es la métrica oficial en las campañas de Semantic Textual Similarity. Es decir, la correlación lineal entre los valores de similitud de pares de frases estimados por el programa frente a la similitud real anotada por humanos en el corpus. Las variantes a estudiar son:

1. Eliminación de palabras según categorías gramaticales, por ejemplo, preposiciones, artículos, etc.
2. Diferentes medidas de similitud léxica basadas en Wordnet, así como corpus de entrenamiento de IC.
3. Diferentes funciones de similitud léxica entre representaciones distribucionales de palabras (vectores).
4. Las dos funciones de similitud propuestas para la similitud a nivel de frase, así como diferentes umbrales de similitud.
5. Comparativa de resultados para diferentes subconjuntos de muestras de STS.

Se valorará la cantidad de variantes cubiertas y la calidad del análisis y explicación de los resultados. Dado que se trata de un trabajo de investigación, es importante analizar la relación existente entre los datos, las aproximaciones y los resultados. Por ejemplo, recomendamos inspeccionar y estudiar de forma cualitativa las características de los textos según el subconjunto al que pertenece: su longitud, el nivel al que conectan semánticamente (nivel lógico, temático, etc.), el tipo de lenguaje y vocabulario empleado, etc. Esto debería tener conexión con la efectividad de las diferentes aproximaciones.

Como puede verse en la documentación de la competición, una parte muy importante es comparar los resultados propios con los obtenidos por otros sistemas, por ello habrá una primera entrega del trabajo de cada uno, y una segunda entrega, en donde hay que añadir al informe un estudio comparativo con los trabajos de los compañeros.

Planificación

El coste estimado de las diferentes actividades dentro de este proyecto es:

1. Familiarización con el problema, lectura de artículos de referencia y documentación (5 horas)
2. Descarga de datos y lectura de ficheros (5 horas).
3. Preprocesado de textos. (10 horas).
4. Similitud léxica: (5 horas).
5. Implementación y ejecución de similitud a nivel de frase (20).
6. Evaluación y análisis de resultados (20 horas).
7. Análisis comparativo con el trabajo de los compañeros (5 horas)

El periodo para desarrollar el proyecto es del 8 de marzo al 4 de mayo. Se aconseja empezar cuanto antes y no dejarlo para el ultimo momento, ya que ademas de las incidencias que puedan surgir en la programación, se requiere tiempo de reflexión y análisis pausado. Habrá una segunda entrega el 18 de mayo incluyendo la comparación de los resultados con los compañeros.

Documentación a entregar

La documentación a entregar consistirá en un informe que contenga:

1. Una descripción del sistema desarrollado, tecnologías empleadas. modularización, etc.
2. Problemas encontrados durante el desarrollo y soluciones empleadas.
3. Análisis comparativo y justificación de los resultados obtenidos.
4. Análisis comparativo en relación a los trabajos realizados por el resto de compañeros. (En una segunda entrega)
5. Conclusiones.

Téngase en cuenta que este informe se hará público para el resto de alumnos. Tiene que estar redactado de forma completa y legible para que se pueda entender y utilizar como fuente de información para la comparación entre los diferentes trabajos presentados.