

Procesamiento del Lenguaje Natural

Introducción al procesamiento del lenguaje natural

Índice

Esquema	3
Ideas clave	4
1.1. Introducción y objetivos	4
1.2. Procesamiento del lenguaje natural	5
1.3. Aplicaciones del PLN	6
1.4. Historia del procesamiento del lenguaje natural	10
1.5. Conocimiento del lenguaje utilizado en el PLN	21
1.6. Referencias bibliográficas	24
A fondo	29
Test	30

Esquema



1.1. Introducción y objetivos

Comenzaremos introduciendo de manera general el campo de la inteligencia artificial denominado **procesamiento del lenguaje natural (PLN)**. La idea de que las máquinas sean capaces de poseer habilidades de comunicación y lenguaje es tan antigua como la propia aparición de los primeros ordenadores.

Se describirán diferentes **aplicaciones** que utilizan **técnicas basadas en el procesado del lenguaje natural**. Por ejemplo: los sistemas de búsqueda de respuestas, el filtrado de mails, la traducción automática, los agentes conversacionales, el análisis de redes sociales, los correctores gramaticales, el análisis de encuestas y, finalmente, la utilidad del PLN para algunos casos de uso concretos.

Algunas de estas aplicaciones, como los agentes conversacionales o la traducción automática, entre otras, se verán con detalle en el futuro. Junto con ello. En los próximos apartados, se presentarán la historia del procesamiento del lenguaje natural y se puede observar cómo ha evolucionado dicho campo y los diferentes enfoques que ha ido tomando a lo largo de la historia. Por último, analizaremos cómo el conocimiento del lenguaje es imprescindible para el correcto funcionamiento de los sistemas de procesamiento de lenguaje natural, describiendo los diferentes tipos de conocimiento del lenguaje.

Objetivos

- ▶ Definir el concepto de procesamiento del lenguaje natural.
- ▶ Identificar distintas aplicaciones del mundo real donde se usa el PLN como parte fundamental.

- ▶ Narrar la historia del PLN.
- ▶ Definir los diferentes tipos de conocimiento del lenguaje que es necesario para correcto funcionamiento de los sistemas de PLN.

1.2. Procesamiento del lenguaje natural

El procesamiento del lenguaje y del habla ha sido tratado históricamente de forma muy diferente en la informática, la ingeniería, la lingüística, la psicología o la ciencia cognitiva. Hoy en día se concibe el **procesamiento del lenguaje natural como área que abarca varios campos diferentes y diversos pero superpuestos.**

Por ello, el **procesamiento del lenguaje natural es un campo interdisciplinario que une a informáticos, ingenieros electrónicos y de telecomunicaciones con lingüistas, sociólogos y psicólogos.**

El **reconocimiento de la voz**, que incluye tareas del procesamiento de la señal, se ha tratado tradicionalmente en la **ingeniería electrónica y de telecomunicaciones**. El **análisis sintáctico y la interpretación semántica de las palabras** y frases son áreas tradicionales del procesamiento del lenguaje natural que se estudian en el campo de la **informática**. La **morfología, la fonología y la pragmática** son tareas de investigación en la **lingüística computacional**. Psicolingüistas y sociolingüistas estudian respectivamente los mecanismos cognitivos para la adquisición del lenguaje y cómo la sociedad influye en el uso de la lengua.

El ancho espectro que abarca el campo del procesamiento del lenguaje natural hace que se conozca con diferentes nombres debido a las diferentes vertientes involucradas. Algunos de estos nombres, que provienen de estas diferentes facetas, serían procesamiento del lenguaje y del habla, tecnología del lenguaje, procesamiento del lenguaje natural, lingüística computacional o reconocimiento y síntesis del habla.

En la inteligencia artificial, el **procesamiento del lenguaje natural** es un campo que tiene como **objetivo que las máquinas sean capaces de realizar tareas que involucren el lenguaje humano.**

Algunas de las tareas que debe realizar una máquina para ser capaz de procesar el lenguaje natural incluyen funcionalidades tales como la de habilitar a la máquina de habilidades para comunicarse con personas, la de mejorar la comunicación entre humanos o, simplemente, la de procesar un texto o el habla.

1.3. Aplicaciones del PLN

El PLN es una de las áreas más importantes dentro del campo de la IA y de la Ciencia de Datos. Por este motivo, actualmente existen multitud de aplicaciones basadas en IA de las que el PLN es una **pieza clave**. Entre estas aplicaciones destacan algunas como las siguientes:



Figura 1. Aplicaciones del procesamiento del lenguaje natural. Fuente: elaboración propia.

Búsqueda de respuestas y autocompletada

En inglés, **Question Answering (QA)**. Es un tipo de recuperación de la información basado en el lenguaje natural. Por lo tanto, es una **extensión de una búsqueda simple de información en la web**, pero en lugar de solo escribir palabras clave, un usuario puede hacer preguntas completas. Los motores de búsqueda pueden responder preguntas sobre fechas y ubicaciones sin necesidad de aplicar procesamiento del lenguaje natural. Sin embargo, para responder a preguntas más complejas se requiere alguno de estos aspectos:

- ▶ La extracción de información o de un fragmento de texto en una página web.
- ▶ Hacer inferencia, es decir, sacar conclusiones basadas en hechos conocidos.
- ▶ Sintetizar y resumir información de múltiples fuentes o páginas web.

Los sistemas de búsqueda de respuestas, que requieren la comprensión de la información, **se componen de** diferentes elementos tales como **un módulo de extracción de información, un módulo para resumir de forma automática o un módulo de desambiguación del sentido de las palabras**.

Junto a la búsqueda de respuestas, el PLN se usa también para el autocompletado de textos, de manera que a medida que un usuario vaya escribiendo una secuencia de palabras, el **sistema tratará de inferir las palabras que vendrían después**.

Filtrado de mails

Uno de los sistemas de filtrado de mails más conocidos es el **filtro de spam**, que en muchas ocasiones **se basa en el uso de técnicas de PLN junto con modelos de aprendizaje automático**. Con ello, en función de determinados elementos del correo (remitente, asunto, contenido, etc.), se podrán filtrar de manera automática aquellos mails que son potencialmente spam.

Ahora bien, **esta aproximación no sirve sólo para la detección de spam, sino que también sirve para clasificar mails en otro tipo de categorías** (promociones, asuntos importantes, etc.).

Traducción automática

Es otra tarea relacionada con el procesamiento del lenguaje natural. El **objetivo** de la traducción automática es **traducir automáticamente un documento de un idioma a otro**. Además, se incluye en este ámbito **no solo** la traducción de documentos de **texto, sino** también la traducción de forma automática del **habla** de un lenguaje o idioma a otro.

Los mejores resultados se obtienen utilizando métodos estadísticos basados en frases. En estos métodos se utiliza básicamente el aprendizaje automático para analizar grandes conjuntos de datos y realizar traducciones que no contemplen las cuestiones gramaticales. En esta se requieren también herramientas para solventar la ambigüedad de las palabras como serían los algoritmos de desambiguación.

Agentes conversacionales

También llamados sistemas de diálogo. Son **programas que conversan con las personas a través del lenguaje natural**. Los **chatbots** son uno de los tipos más avanzados de los agentes conversacionales porque permiten mantener conversaciones no estructuradas, una característica de las conversaciones entre personas. Los agentes conversacionales pueden interactuar con el humano ya sea a través de la voz (hablando con el usuario), de texto, en el caso que la conversación se lleve a cabo a través de un chat, o utilizando ambas modalidades a la vez.

Los agentes conversacionales se caracterizan por ser sistemas que toman turnos para conversar, por lo que aparte de tener que analizar el lenguaje natural durante la conversación, deben tener en cuenta el turno de palabra. Por lo tanto, los agentes

conversacionales, además de tratar con tareas básicas del procesamiento del lenguaje natural como son el reconocimiento de palabras y frases o la semántica de estas, deben mantener el estado de la conversación y ser capaces de generar nuevas frases que continúen la conversación que mantienen con la persona.

Análisis de redes sociales

Gran parte de la población usa algún tipo de red social, de manera que estas recogen mucha información de usuarios de distintas partes del mundo. De esta manera, por ejemplo, si una empresa lanza un nuevo producto al mercado, las redes sociales tendrán información muy valiosa de la opinión de gran parte de los consumidores sobre ese nuevo producto. Ahora bien, esta información suele quedar reflejada en las redes sociales como textos (ej., un tweet). Para poder analizar grandes volúmenes de información de este tipo de manera automática se usan precisamente técnicas de PLN, muchas veces en combinación con **algoritmos de aprendizaje automático**. Por ejemplo, podemos analizar el sentimiento de los usuarios hacia una marca o producto concreto en un momento dado para ver si este es positivo, negativo o neutro, utilizando técnicas de PLN que extraigan información relevante de los textos, junto con modelos de aprendizaje automático que clasifiquen esa información en una de las tres categorías.

Correctores gramaticales

Así como el PLN sirve para tareas como el autocompletado de texto (sugiriendo posibles maneras de continuación de una frase), también **es un elemento clave en los correctores gramaticales**. Teniendo en cuenta las palabras previas escritas, el conocimiento de la lengua, la información sintáctica y semántica. Se pueden tanto corregir errores ortográficos, así como proponer maneras más correctas de escribir una frase.

Análisis de encuestas

Es habitual que muchas empresas y organismos realicen encuestas para conocer la opinión de los usuarios respecto de distintos temas. Ahora bien, puede ocurrir que estas encuestas contengan *feedback* de los usuarios expresado en texto libre, de manera que, si el número de encuestas es muy elevado, puede ser muy laborioso hacer un análisis manual de todas ellas. Aquí también entran las técnicas de PLN para poder procesar esos textos y extraer la información relevante que allí se contiene.

PLN para casos de uso concretos

A modo de ejemplo, otro de los sectores donde el PLN está siendo de gran utilidad es en el ámbito de Recursos Humanos (RR. HH.) para tareas como, por ejemplo, la criba curricular. Si la cantidad de CV que recibe la empresa es muy elevada, y si además los formatos de estos no son homogéneos, el uso de técnicas de PLN puede ayudar a ordenarlos según cómo estén de alineados con la descripción del puesto de trabajo. Esto también se puede utilizar para ver si un candidato está más alineado para otra vacante disponible en lugar de a la que había aplicado.

Este es un ejemplo, pero hay otros casos de uso específicos para distintos dominios (ej., analizar automáticamente historias clínicas en el ámbito médico).

1.4. Historia del procesamiento del lenguaje

natural

La historia del procesamiento del lenguaje natural se puede dividir en diferentes etapas. Desde la aparición de las bases o paradigmas fundacionales en la década de 1940 hasta la explotación de los paradigmas más modernos para desarrollar hoy en día aplicaciones más inteligentes.

1940-1950	Paradigmas fundacionales
1957-1970	Paradigma simbólico y estocástico
1970-1983	Cuatro paradigmas de investigación
1983-1993	Revivir del empirismo y los modelos de estados finitos
1994-1999	Unión de las diferentes vertientes
2000	Auge del aprendizaje automático

Tabla 1. Cronología de las diferentes etapas de la historia del PLN. Fuente: elaboración propia.

Paradigmas fundacionales: década de 1940 y 1950

El principio del PLN data del período justo después de la Segunda Guerra Mundial, cuando se dio el origen del ordenador. En este período, desde la década de 1940 hasta el final de la década de 1950, se trabajó intensamente en dos paradigmas fundacionales:

- ▶ Autómatas.
- ▶ Modelos probabilísticos o de teoría de la información.

Autómatas

El autómata surgió en la década de 1950 a partir del famosísimo estudio publicado por Turing (1936), Los números computables, con una aplicación al Entscheidungsproblem, y que se considera como la base de la informática moderna. El trabajo de Turing condujo primero a un modelo simplificado de la neurona como elemento de computación que podría describirse en términos de lógica proposicional (McCulloch y Pitts, 1943) y luego a la definición de los autómatas finitos y las expresiones regulares (Kleene, 1951).

Shannon aplicó las cadenas de Markov, un modelo de proceso estocástico discreto en el que la probabilidad de ocurrencia de un evento depende solo del evento

inmediatamente anterior, a los autómatas para el lenguaje (Shannon, 1948). Aprovechando las ideas del trabajo de Shannon, Chomsky fue el primero en considerar las máquinas de estados finitos como una forma de caracterizar una gramática y definió el lenguaje de estados finitos como un lenguaje generado por una gramática de estados finitos (Chomsky, 1956).

Estos primeros modelos llevaron a la aparición de la teoría del lenguaje formal, que utilizó el álgebra y la teoría de conjuntos para definir los lenguajes formales como secuencias de símbolos. Esta teoría incluye las gramáticas libres de contexto, un concepto que Chomsky definió en 1956 para las lenguas naturales, pero que también fue descubierto de forma independiente por Backus y Naur en su descripción del lenguaje de programación ALGOL (Backus, 1959) (Naur et al., 1960).

Teoría de la información

El segundo paradigma fundamental de este período fue el desarrollo de algoritmos probabilísticos para el procesamiento del lenguaje y del habla. Esta idea proviene de la otra gran contribución de Shannon, el teorema de codificación de canal en la teoría de la información, y que muestra que es posible la transmisión y decodificación del lenguaje a través de un canal de comunicación ruidoso.

Shannon tomó prestado el concepto de entropía de la termodinámica como una forma de medir la capacidad de información de un canal o el contenido de información de un idioma, y realizó la primera medida de la entropía del inglés utilizando técnicas probabilísticas.

También durante este período inicial se desarrolló el espectrógrafo de sonido y se realizó investigación fundamental en la fonética instrumental, lo que sentó las bases para el reconocimiento de la voz. La primera máquina capaz de realizar el reconocimiento de la voz apareció a principios de los años cincuenta. En 1952 investigadores de Bell Labs construyeron un sistema estadístico basado en correlación que podía reconocer con un 97-99 % de precisión cualquiera de los diez

primeros números a partir de unos patrones grabados con los sonidos de las vocales de un único hablante (Davis, Biddulph y Balashek, 1952).

Paradigma simbólico y estocástico: 1957-1970

A fines de la década de 1950 y comienzos de la década de 1960, el procesamiento del habla y el lenguaje se había dividido muy claramente en dos paradigmas:

- ▶ Simbólico.
- ▶ Estocástico.

Paradigma simbólico

Apareció de dos líneas de investigación: la teoría del lenguaje formal y la inteligencia artificial.

La primera línea se basaba en el trabajo que realizaron Chomsky y sus colaboradores en la **teoría del lenguaje formal y la sintaxis generativa**, además de en el trabajo de muchos lingüistas e informáticos que estudiaban los algoritmos de análisis, inicialmente de arriba hacia abajo (*top-down*) y de abajo hacia arriba (*bottom-up*) y luego a través de la programación dinámica. Uno de los primeros sistemas de análisis fue TDAP (Transformations and Discourse Analysis Project) que implementó Zelig Harris entre junio de 1958 y julio de 1959 en la Universidad de Pennsylvania.

La segunda línea de investigación del paradigma simbólico fue el nuevo campo de la **inteligencia artificial**. En el verano de 1956, John McCarthy, Marvin Minsky, Claude Shannon y Nathaniel Rochester congregaron durante dos meses a un grupo de investigadores para realizar un simposio sobre lo que decidieron llamar «inteligencia artificial».

Aunque el incipiente ámbito de la inteligencia artificial incluía una minoría de investigadores centrados en algoritmos estocásticos y estadísticos (incluidos los

modelos probabilísticos y las redes neuronales), este nuevo campo se enfocó básicamente en el **razonamiento y la lógica**, representados por el trabajo de Newell y Simon en los programas de ordenador Logic Theorist y General Problem Solver (Newell, Shaw y Simon, 1959).

En los principios de la inteligencia artificial fue cuando se construyeron los primeros **sistemas de comprensión del lenguaje natural**. Estos sistemas simples estaban diseñados para trabajar en un dominio concreto y basaban su funcionamiento en la búsqueda de patrones y heurística de palabras clave para realizar el razonamiento y la búsqueda de respuestas. Fue a finales de la década de 1960 cuando se desarrollaron algunos sistemas lógicos más formales.

Paradigma estocástico

Se desarrolló principalmente por parte de investigadores de los departamentos de estadística y de ingeniería electrónica. A fines de la década de 1950 comenzó a aplicarse el método bayesiano al problema del reconocimiento óptico de caracteres. Por ejemplo, **Bledsoe y Browning construyeron un sistema bayesiano de reconocimiento de texto que calculaba la probabilidad de una secuencia de letras dadas las palabras de un diccionario**.

En la década de 1960 aparecieron también los primeros **modelos psicológicos para el PLN basados en gramáticas transformacionales y los primeros corpus disponibles online**. Un ejemplo es el Brown Corpus, un corpus del inglés americano desarrollado en 1963 por la Brown University y que contenía una colección de un millón de palabras extraídas de 500 textos de diferentes géneros: periódicos, novelas, no ficción, académico, etc. (Kucera y Francis, 1967).

Cuatro paradigmas de investigación: 1970-1983

En el siguiente período, en la década de 1970 y a principios de la década de 1980, se produce una explosión de la investigación en el procesamiento de lenguaje y del

habla. Es en esta época cuando se desarrollan una serie de paradigmas de investigación que todavía hoy dominan el campo:

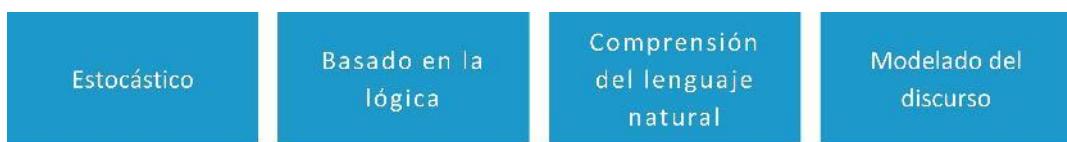


Figura 2. Paradigmas de investigación dominantes. Fuente: elaboración propia.

El **paradigma estocástico** jugó un papel muy importante en el desarrollo de algoritmos de reconocimiento de voz para los que se utilizaban modelos ocultos de Markov (HMM) y el teorema de codificación de canal de Shannon. Algunos de los investigadores que trabajaron en este ámbito fueron Jelinek, Bahl, Mercer y sus socios del Thomas J. Watson Research Center de IBM, Baker en la Carnegie Mellon University, e investigadores de los Bell Laboratories de AT&T.

El **paradigma basado en lógica** surgió del trabajo de Alain Colmerauer y sus colaboradores en la década de 1970 que desarrollaron Q-system, un analizador *bottom-up* basado en una serie de reglas con variables lógicas y que permitían la traducción del inglés al francés (Colmerauer, 1978). En este paradigma basado en lógica se engloba también la gramática de cláusulas definidas (Definite Clause Grammar, DCG), una forma de expresar la gramática en un lenguaje de programación lógico como por ejemplo Prolog (Pereira y Warren, 1980).

De forma independiente, apareció también el trabajo de Kay (1979) sobre la gramática funcional y, poco después, el trabajo de Joan Bresnan y Ronald Kaplan (1982) sobre la gramática léxico funcional (Lexical functional grammar, LFG), una gramática generativa que se centra en la investigación de la sintaxis del lenguaje natural.

El **paradigma de la comprensión del lenguaje natural** apareció durante la década de 1970 con la aparición del sistema SHRDLU (Winograd, 1972). Este sistema simulaba un robot que movía bloques y era capaz de recibir comandos del lenguaje natural en

formato de texto, como por ejemplo «mueve el bloque rojo que se encuentra en la parte superior del verde más pequeño». Este sistema, complejo y sofisticado para su época, también fue el primero en construir una gramática relativamente extensa del inglés.

Los avances en el modelo de análisis del lenguaje de Winograd dejó claro que la investigación debía comenzar a enfocarse en la semántica y los modelos de discurso. Roger Schank y sus colaboradores, conocidos como la Escuela de Yale, construyeron una serie de programas de comprensión del lenguaje que se centraron en el conocimiento humano, por ejemplo, en planes y objetivos, y la organización de la memoria humana (Schank y Riesbeck, 1981). Estos trabajos, por ejemplo, el realizado por R. F. Simmons (1973), usaban una representación del conocimiento en forma de red, lo que se conoce como redes semánticas (Quillian, 1968), y comenzaron a incorporar en sus representaciones la idea de gramática de casos (Fillmore, 1968), por la cual se establece una relación entre un verbo y múltiples papeles temáticos que serían los sintagmas nominales.

El paradigma basado en lógica y el paradigma de la comprensión del lenguaje natural se unificaron en sistemas que usaban la lógica de predicados como una representación semántica, como el sistema de búsqueda de respuestas LUNAR (Woods, 1967).

El **paradigma del modelado del discurso** se centró en las cuatro áreas clave del discurso. Grosz y sus colaboradores introdujeron el **estudio de la estructura del discurso y el enfoque del discurso** (Grosz, 1977) (Grosz y Sidner, 1986). Una serie de investigadores comenzaron a trabajar en la resolución de forma automática de referencias en el discurso (Hobbs, 1978 y 1979). Por último, se desarrolló el modelo BDI (creencias-deseos-intenciones en inglés), un marco de trabajo basado en la lógica de actos de habla (Perrault y Allen, 1980) (Cohen y Perrault, 1979).

Revivir del empirismo y los modelos de estados finitos: 1983-1993

A partir de 1983 volvieron dos clases de modelos que habían perdido popularidad a finales de la década de 1950 y principios de la década de 1960 debido a los argumentos teóricos en su contra (Chomsky, 1959).



Figura 3. Modelos de estados finitos y modelos empíricos. Fuente: elaboración propia.

Los **modelos de estados finitos** revivieron en esta época y comenzaron a recibir atención porque se usaron en la fonología y la morfología (Kaplan y Kay, 1981), y en la sintaxis (Church, 1980).

La segunda tendencia en este período fue lo que se ha llamado el **retorno del empirismo**, marcada por el aumento de los **modelos probabilísticos** para el **procesamiento del lenguaje y del habla**. Cabe destacar la influencia del trabajo de los investigadores del Thomas J. Watson Research Center de IBM sobre **modelos probabilísticos** en el reconocimiento de voz. Estos métodos probabilísticos y otros **enfoques basados en los datos** se extendieron al **etiquetado morfosintáctico** (POS tagging), al análisis y resolución de ambigüedades y a la semántica.

Este paradigma empírico vino acompañado por el enfoque de la evaluación de los modelos basado en los datos. Entonces en este período se desarrollaron métricas cuantitativas para la evaluación y se enfatizó en la comparación del rendimiento de estas métricas con los resultados de las investigaciones previas. Además, en este período, se trabajó considerablemente en la generación de lenguaje natural.

Unión de las diferentes vertientes: 1994-1999

En los últimos cinco años del pasado milenio, el campo del procesamiento del lenguaje natural sufrió grandes cambios.

En primer lugar, los modelos probabilísticos y los modelos basados en datos se volvieron estándares para el procesamiento del lenguaje natural. Los algoritmos de análisis, de etiquetado morfosintáctico, de resolución de referencias y de procesamiento del discurso empezaron a incorporar probabilidades y adoptaron metodologías de evaluación provenientes de los ámbitos del reconocimiento de la voz y la recuperación de información.

En segundo lugar, el aumento en la velocidad y la memoria de los ordenadores permitió la explotación comercial de varias áreas del procesamiento del lenguaje y del habla, en particular el reconocimiento de la voz y la revisión de la ortografía y la gramática. Además, los algoritmos de procesamiento del lenguaje y del habla comenzaron a aplicarse a la comunicación aumentativa para ayudar a personas con algún tipo de discapacidad. Por último, el aumento de la web enfatizó la necesidad de la recuperación y la extracción de información basada en el lenguaje natural.

Auge del aprendizaje automático: 2000

Las tendencias empíricas que marcaron la última parte de la década de 1990 se aceleraron a un ritmo asombroso en el nuevo milenio. Esta aceleración fue impulsada en gran parte por el auge del aprendizaje automático.



Figura 4. Aprendizaje automático. Fuente: elaboración propia.

Grandes cantidades de material hablado y escrito se pusieron a disposición de los investigadores a través de organizaciones tipo el Linguistic Data Consortium (LDC). Estos materiales eran fuentes de texto estándar que venían anotados con información sintáctica, semántica y pragmática. Entre estos materiales caben destacar las primeras colecciones anotadas: el Penn Treebank, el Prague Dependency Treebank, que anota la estructura de las dependencias, y las anotaciones semánticas del PropBank.

La existencia de estos recursos anotados promovió la tendencia de atacar los problemas más complejos del procesamiento del lenguaje natural, tipo el análisis sintáctico y semántico, como problemas de aprendizaje automático supervisado. Por ejemplo, se empezaron a aplicar las máquinas de vectores de soporte (SVM), el principio de máxima entropía, la regresión logística multinomial y los modelos bayesianos en la lingüística computacional.

Entonces, este mayor enfoque en el aprendizaje automático llevó a una interacción más seria de los lingüistas computacionales con la comunidad estadística.

El coste y la dificultad de producir corpus anotados se convirtió en un factor limitante del uso de los enfoques supervisados para muchos problemas del procesamiento del lenguaje. Por lo que a partir de 2005 aparece una nueva tendencia hacia el uso de técnicas de **aprendizaje no supervisado** en el procesamiento del lenguaje natural.

Entonces se empezaron a construir algunas aplicaciones lingüísticas a partir de datos sin anotación alguna, por ejemplo, para la traducción automática o para el modelado de temas.

Los algoritmos de aprendizaje no supervisado se han usado en el etiquetado morfosintáctico (POS tagging) para agrupar palabras en las correspondientes partes del lenguaje (Goldwater y Griffiths, 2007) (Sirts, Eisenstein, Elsner y Goldwater, 2014). Además, las técnicas del aprendizaje no supervisado se han usado también para el etiquetado semántico, donde se han creado conjuntos de roles semánticos a partir de las características sintácticas (Titov y Klementiev, 2012) (Lang y Lapata, 2014).

En 2006, Geoffrey Hinton acuña el término **deep learning** (aprendizaje profundo). Con el auge de este tipo de redes neuronales en la década de 2010, estas redes de neuronas artificiales profundas se empezaron a usar en diferentes ámbitos del procesamiento del lenguaje natural. Las redes neuronales recurrentes se están utilizando como una alternativa a los modelos ocultos de Markov (HMM) en análisis morfosintáctico y en el análisis sintáctico (Chen y Manning, 2014) (Dozat, Qi, y Manning, 2017). Además, las redes neuronales profundas también se están utilizando para el etiquetado semántico (Collobert et al., 2011) (Foland Jr. y Martin, 2015). De hecho, el **deep learning** es la base de los modelos de secuencia a secuencia (seq2seq) que se utilizan en los agentes conversacionales y *chatbots* actuales.

En el vídeo *Aplicaciones del procesamiento del lenguaje natural* se realizará un análisis de la importancia del PLN en el mundo actual, y cómo es un elemento clave en muchas aplicaciones.



Accede al vídeo

En el vídeo *Historia del procesamiento del lenguaje natural* se resume la historia de este, al tiempo que destaca los hitos más importantes de esta.



Accede al vídeo

1.5. Conocimiento del lenguaje utilizado en el PLN

Lo que distingue a las aplicaciones de procesamiento de lenguaje de otros sistemas de procesamiento de datos es **su uso del conocimiento del lenguaje**. Por ejemplo, un programa que cuenta el número de bytes, palabras y líneas en un archivo de texto podemos considerarlo como una aplicación de procesamiento de datos ordinaria.

Sin embargo, si para contar las palabras en el archivo de texto se requiere conocimiento sobre lo que significa ser una palabra porque se quieren contar el número de pronombres que aparecen en el archivo, ese programa se convierte en un sistema de procesamiento de lenguaje natural.

Por supuesto, este sistema de procesamiento de lenguaje es extremadamente simple en comparación con los agentes conversacionales, los sistemas de traducción automática y los sistemas de búsqueda de respuestas, que requieren un conocimiento mucho más amplio y profundo del lenguaje.



Figura 5. Conocimientos necesarios para tareas complejas de PLN. Fuente: elaboración propia.

Un agente conversacional necesita reconocer las palabras de una señal de audio y generar una señal de audio de una secuencia de palabras. Estas tareas de reconocimiento de la voz y del habla son tareas de síntesis que requieren conocimiento sobre **fonética y fonología**. Es decir, conocimiento de cómo se pronuncian las palabras a partir de la secuencia de sonidos y cómo se generan cada uno de estos sonidos acústicamente.

Fonética y fonología: conocimiento sobre los sonidos lingüísticos

El agente conversacional también necesita **reconocer y saber producir variaciones de las palabras**. Por ejemplo, reconocer que «puertas» es el plural de una palabra y saber generar una frase en la que esta palabra se utilice en plural. Entonces, estas tareas requieren conocimiento sobre morfología, es decir, la forma en que las palabras se descomponen en partes que tienen un significado como puede ser la raíz de la palabra y una terminación que indique que es un plural.

Morfología: conocimiento de los componentes significativos de las palabras

Si vamos más allá de las palabras como elementos aislados y entendidas de forma individual, un agente conversacional debe usar conocimiento estructural para encadenar las palabras que constituyen su respuesta. El agente debe saber identificar que una secuencia de palabras no tiene sentido a pesar de que el conjunto de

palabras original pudiera tener sentido si se ordenaran las palabras de otra forma. El conocimiento necesario para ordenar y agrupar palabras se llama sintaxis.

Sintaxis: conocimiento de las relaciones estructurales entre palabras

Para responder a una pregunta, el agente conversacional puede necesitar saber algo sobre la semántica léxica, es decir, sobre el significado de cada una de las palabras, así como sobre semántica composicional o el significado de varias palabras que se utilizan de forma conjunta en una combinación de palabras.

Semántica: conocimiento del significado

Además de comprender el significado de las palabras, el agente conversacional puede necesitar saber algo sobre la relación de las palabras con la estructura sintáctica: si un sintagma es un complemento circunstancial de tiempo o un complemento del nombre.

Por lo tanto, el conocimiento sobre la sintaxis y el conocimiento sobre la semántica se van a utilizar de forma conjunta en el procesado del lenguaje natural.

El agente conversacional necesita determinar también el tipo de expresión que le ha interpuesto el usuario. Necesita saber si la expresión con la que este le acaba de interesar es una pregunta a la que debe dar una respuesta hablada, una solicitud para que realice una acción o un simple enunciado o declaración sobre un hecho. Además, el agente puede determinar usar expresiones más formales y educadas en función de su interlocutor y cómo este le haya preguntado. Entonces, el agente necesita conocimiento sobre el diálogo o la pragmática para poder identificar la intención que tiene el usuario al interesarle y dar una respuesta acorde.

Pragmática: conocimiento de la relación del significado con los objetivos y las intenciones

Por último, el agente conversacional necesita interpretar palabras o expresiones que hacen referencia a términos que han aparecido anteriormente en la conversación. Sería el caso de pronombres o sintagmas nominales con determinantes que se refieren a partes previas del discurso. Es por eso por lo que el agente examina las preguntas anteriores que se formularon previamente y utiliza el conocimiento sobre el discurso previo para resolver las referencias cruzadas.

Discurso: conocimiento sobre unidades lingüísticas más grandes que un solo enunciado.

En conclusión, para realizar tareas complejas de PLN se necesitan **diferentes tipos de conocimiento del lenguaje**, concretamente conocimiento sobre la fonética y fonología, la morfología, la sintaxis, la semántica, la pragmática y el discurso.

1.6. Referencias bibliográficas

Backus, J. W. (1959). *The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference*. International Business Machines Corp., New York, USA.

Bresnan, J. y Kaplan, R. M. (1982). Introduction: Grammars as mental representations of language. En Bresnan, J. (Ed.), *The Mental Representation of Grammatical Relations*. MIT Press.

Chen, D. and Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. En *Proceedings of the 2014 Conference on Empirical Methods in*

Natural Language Processing (EMNLP) (pp. 740-750). Doha, Catar: Association for Computational Linguistics.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2(3), 113-124.

Chomsky, N. (1959). A review of B. F. Skinner's «Verbal Behavior». *Language*, 35, 26-58.

Church, K. W. (1980). *On memory limitations in natural language processing* (Tesis de maestría, Massachusetts Institute of Technology).

https://www.researchgate.net/publication/230875927_On_Memory_Limitations_in_Natural_Language_Processing

Cohen, P. R. y Perrault, C. R. (1979). Elements of a planbased theory of speech acts. *Cognitive Science*, 3(3), 177–212.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. y Kuksa, P. (2011). Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12, 2493–2537.

Colmerauer, A. (1978). Metamorphosis grammars. En L. Bolc (Ed.), *Natural Language Communication with Computers, Lecture Notes in Computer Science 63* (pp. 133-189). Springer Verlag.

Davis, K. H., Biddulph, R. y Balashek, S. (1952). Automatic recognition of spoken digits. *Journal of the Acoustical Society of America*, 24(6), 637-642.

Dozat, T., Qi, P., y Manning, C. D. (2017). Stanford's graph-based neural dependency parser at the CoNLL 2017 shared task. En *Proceedings of the CoNLL 2017 Shared Task* (pp. 20-30). Vancouver, Canadá: Association for Computational Linguistics.

Fillmore, C. J. (1968). The case for case. En E. W. Bach y R. T. Harms, (Eds.), *Universals in Linguistic Theory* (pp. 1-88). Nueva York, Estados Unidos: Holt, Rinehart & Winston.

Foland Jr., W. R. y Martin, J. H. (2015). Dependency-based semantic role labeling using convolutional neural networks. En *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics (*SEM 2015)* (pp. 279-289). Denver, Estados Unidos: *SEM 2015 Organizing Committee.

Goldwater, S. y Griffiths, T. L. (2007). A fully Bayesian approach to unsupervised part-of-speech tagging. *Annual Meeting of the Association of Computational Linguistics ACL-07*, 744-751. <http://aclweb.org/anthology/P07-1094>

Grosz, B. J. (1977). The representation and use of focus in a system for understanding dialogs. *International Joint Conference on Artificial Intelligence*, 67-76.

Grosz, B. J. y Sidner, C. L. (1986). *Attention, intentions, and the structure of discourse. Computational Linguistics*, 12(3), 175-204.

Hobbs, J. R. (1978). Resolving pronoun references. *Lingua*, 44, 311-338.

Hobbs, J. R. (1979). *Coherence and coreference. Cognitive Science*, 3, 67-90.

Kaplan, R. M. y Kay, M. (1981). Phonological rules and finite-state transducers. *Annual meeting of the Linguistics Society of America*. Linguistic Society of America.

Kay, M. (1979). *Functional grammar. Proceedings of the Annual Meeting of the Berkeley Linguistics Society*, Estados Unidos. http://linguistics.berkeley.edu/bls/previous_proceedings/bls5.pdf

Kleene, S. C. (1951). *Representation of events in nerve nets and finite automata*. Santa Mónica, Estados Unidos: RAND Corporation. https://www.rand.org/pubs/research_memoranda/RM704.html

Kucera, H. y Francis, W. N. (1967). *Computational analysis of present-day American English*. Brown University Press.

Lang, J. y Lapata, M. (2014). Similarity-driven semantic role induction via graph partitioning. *Computational Linguistics*, 40(3), 633-669.

McCulloch, W. S. y Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 115-133.

Naur, P., Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., van Wijnagaarden, A. y Woodger, M. (1960). Report on the algorithmic language ALGOL 60. *Communications of the ACM*, 3(5), 299–314.

Newell, A., Shaw, J. C. y Simon, H. A. (1959). Report on a general problem-solving program. En UNESCO (Ed.), *Information processing: proceedings of the International Conference on Information Processing* (pp. 256-264). Oldenbourg Verlag.

Pereira, F. C. N. y Warren, D. H. D. (1980). Definite clause grammars for language analysis: a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13(3), 231-278.

Perrault, C. R. and Allen, J. (1980). A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics*, 6(3-4), 167-182.

Quillian, M. R. (1968). Semantic memory. En M. Minsky (Ed.), *Semantic Information Processing* (pp. 227–270). MIT Press.

Schank, R. C. y Riesbeck, C. K. (Eds.). (1981). *Inside computer understanding: five programs plus miniatures*. Lawrence Erlbaum.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3), 379-423.

Simmons, R. F. (1973). Semantic networks: Their computation and use for understanding English sentences. En R. C. Schank y K. M. Colby (Eds.), *Computer Models of Thought and Language* (pp. 61-113). San Francisco, Estados Unidos: W.H. Freeman and Co.

Sirts, K., Eisenstein, J., Elsner, M., y Goldwater, S. (2014). POS induction with distributional and morphological information using a distance-dependent Chinese restaurant process. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 265-271. <http://www.aclweb.org/anthology/P14-2044>

Titov, I. y Klementiev, A. (2012). A Bayesian approach to unsupervised semantic role induction. *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, 12-22. <https://dl.acm.org/citation.cfm?id=2380821>

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1), 230-265.

Winograd, T. (1972). *Understanding natural language*. Cognitive Psychology, 3(1), 1-191.

Woods, W. A. (1967). *Semantics for a Question-Answering System*. Garland Publishing.

A fondo

Sistemas de diálogo

Mooc Málaga. (2015). *Sistemas de diálogo.*

<https://www.youtube.com/watch?v=9tfhNWqsx-8>

Este vídeo muestra una breve introducción a los sistemas de diálogo hablado que son una de las aplicaciones del lenguaje natural que más interés suscitan.

Búsqueda de respuestas

Mooc Málaga. (2017). *Búsqueda de respuestas.*

https://www.youtube.com/watch?v=49_KA89mPHg

Este vídeo muestra una breve introducción a los sistemas de búsqueda de respuestas donde el PLN permite que estos sistemas vayan más allá que los sistemas de recuperación de información.

Test

1. Indica las afirmaciones correctas sobre el procesamiento del lenguaje natural:

 - A. También se llama reconocimiento de la voz.
 - B. Es un campo de la inteligencia artificial que tiene como objetivo que las máquinas realicen tareas que involucren el lenguaje humano.
 - C. Es un campo interdisciplinar que involucra disciplinas tan diversas como el procesamiento de la señal, el análisis sintáctico y semántico, la morfología, fonología y pragmática, la psicolingüística y la sociolingüística.
 - D. Es un campo en el que trabajan informáticos, ingenieros, lingüistas, sociólogos y psiquiatras.
2. Indica las afirmaciones correctas sobre el procesamiento del lenguaje natural en las décadas de 1940 y 1950:

 - A. El procesamiento del lenguaje natural aparece justo antes de la Segunda Guerra Mundial.
 - B. Uno de los paradigmas fundacionales del procesamiento del lenguaje natural se basa en uso de autómatas finitos y más concretamente cadenas de Markov.
 - C. Uno de los paradigmas fundacionales del procesamiento del lenguaje natural se basa en las ideas de Shannon descritas en la teoría de la información.
 - D. En esa época aparece la primera máquina capaz de reconocer la voz.

- 3.** Indica las afirmaciones correctas sobre el paradigma simbólico en el período de 1957 a 1970:
- A. Una de sus líneas de investigación se basa en la teoría del lenguaje formal y la sintaxis generativa.
 - B. Una de sus líneas de investigación se basa en la inteligencia artificial.
 - C. Aplicaba el método bayesiano.
 - D. Utilizaba los primeros corpus *online*.
- 4.** Indica las afirmaciones correctas sobre los cuatro paradigmas de investigación que se dieron en el período de 1970 a 1983:
- A. El paradigma estocástico utilizaba modelos ocultos de Markov para el reconocimiento de la voz.
 - B. El paradigma del modelado del discurso se basó en la estructura y el enfoque del discurso.
 - C. El paradigma basado en lógica utilizaba la lógica de predicados.
 - D. En el paradigma de la comprensión del lenguaje natural se utilizaban las redes semánticas.
- 5.** Indica las afirmaciones correctas sobre los cambios que sufrió el campo del procesamiento del lenguaje natural a finales del pasado milenio:
- A. Se recuperaron los modelos de estados finitos y el empirismo
 - B. Se volvieron populares los modelos probabilísticos.
 - C. Se volvieron estándares los modelos basados en datos.
 - D. Se empezaron a comercializar algunos productos con tecnologías del procesamiento del lenguaje natural.

- 6.** Indica las afirmaciones correctas sobre como el auge del aprendizaje automático ha cambiado el procesamiento del lenguaje natural desde el año 2000:
- A. El *deep learning* es un ámbito por explotar que se va a estudiar en los próximos años.
 - B. El aprendizaje supervisado se ha usado en el análisis semántico.
 - C. El aprendizaje no supervisado se ha usado en el análisis sintáctico y semántico.
 - D. El aprendizaje no supervisado es una técnica más eficiente que el aprendizaje supervisado al no requerir anotaciones de los corpus.
- 7.** Indica cuales de las siguientes aplicaciones utilizan técnicas del procesamiento del lenguaje natural:
- A. Los agentes conversacionales.
 - B. La corrección ortográfica.
 - C. La búsqueda de respuestas.
 - D. La traducción automática.
- 8.** Si un agente conversacional tiene que crear una respuesta en la conversación y para ello encadena diferentes palabras, ¿qué tipo de conocimiento necesita para realizar esta tarea? Indica las respuestas correctas:
- A. Conocimiento sobre el significado de las palabras.
 - B. Conocimiento sobre las relaciones estructurales entre palabras.
 - C. Conocimiento fonético.
 - D. Conocimiento sintáctico.

9. Si un agente conversacional necesita interpretar si el usuario le ha hecho una pregunta o simplemente le ha contado un hecho, ¿qué tipo de conocimiento necesita para realizar esta tarea? Indica las respuestas correctas:

- A. Conocimiento pragmático.
- B. Conocimiento sobre el diálogo.
- C. Conocimiento sobre el discurso.
- D. Conocimiento sobre conocimiento sobre la relación del significado con los objetivos y las intenciones.

10. Si un agente conversacional necesita generar una frase en la que el número del nombre en el sujeto debe ser un plural para que concuerde con el verbo, ¿qué tipo de conocimiento necesita para realizar esta tarea? Indica las respuestas correctas:

- A. Conocimiento morfológico.
- B. Conocimiento semántico.
- C. Conocimiento sintáctico.
- D. Conocimiento sobre los componentes significativos de las palabras.

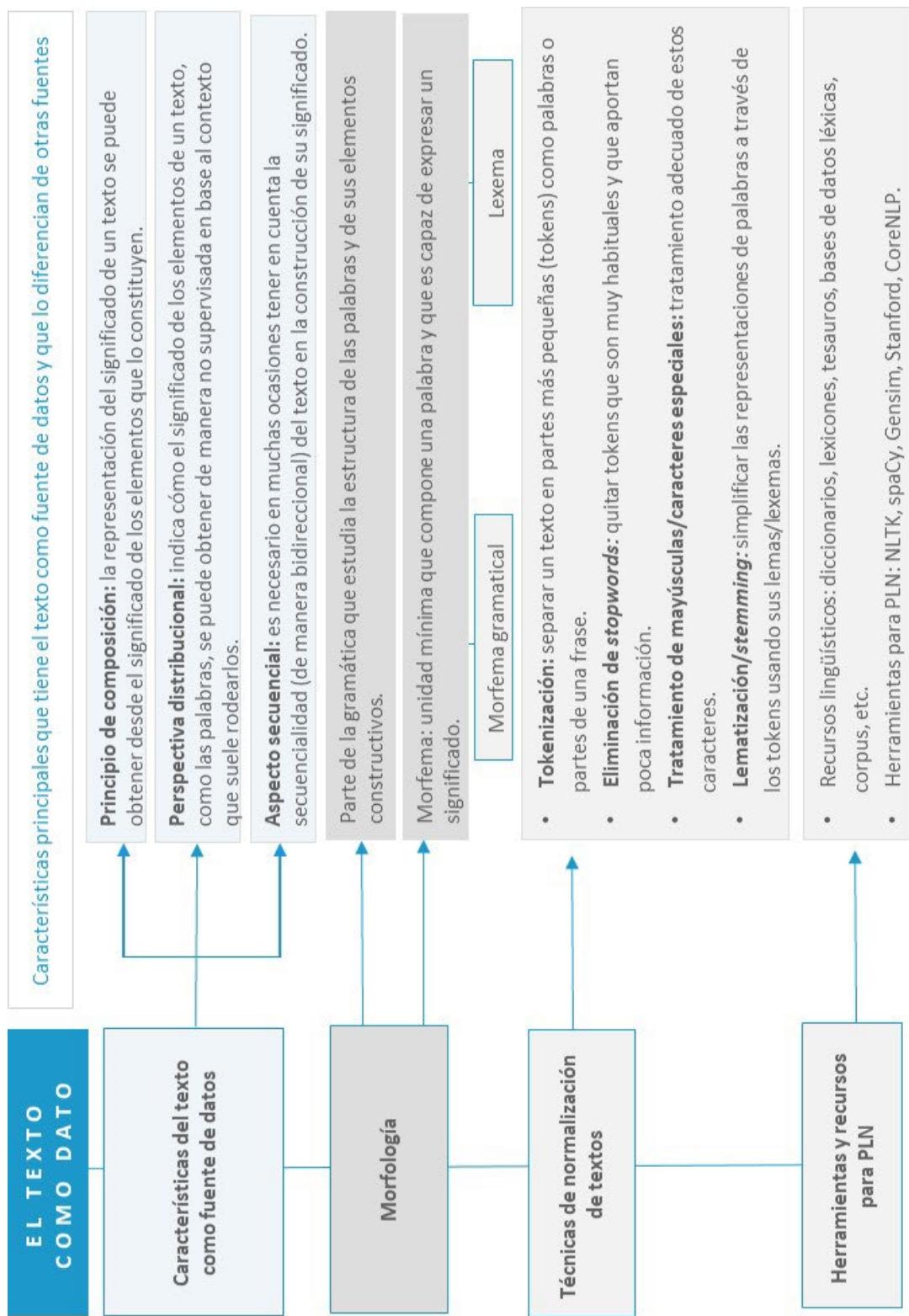
Procesamiento del Lenguaje Natural

El texto como dato

Índice

Esquema	3
Ideas clave	4
2.1. Introducción y objetivos	4
2.2. Características del texto como fuente de datos	5
2.3. Morfología	9
2.4. Técnicas de normalización de textos	12
2.5. Recursos lingüísticos	18
2.6. Corpus en español	22
2.7. Herramientas y librerías para el PLN	24
2.8. Referencias bibliográficas	27
A fondo	29
Test	31

Esquema



2.1. Introducción y objetivos

A continuación, se presentarán las características principales que tiene el texto como fuente de datos, y qué lo diferencia de otras fuentes. Por otro lado, se define el concepto de morfología y qué aspectos incluye. Esto servirá como base para explicar distintas técnicas de normalización de textos, muy usadas en aplicaciones de PLN, y cómo se relacionan dentro de lo que se conoce como flujo (*pipeline*) de normalización. Finalmente, se explicarán los distintos recursos lingüísticos, se darán referencias a distintos corpus en español que se pueden usar para tareas de PLN, junto con referencias a algunas de las herramientas de software más usadas hoy en día para ello.

Objetivos

- ▶ Entender las características concretas que tienen los textos como fuentes de datos, y qué aspectos hay que tener en cuenta para poder trabajar con ellos.
- ▶ Describir los conceptos fundamentales de la morfología en español.
- ▶ Conocer algunas de las técnicas más habituales de normalización de textos, y cómo poder incluirlas en un flujo de normalización.
- ▶ Describir los diferentes tipos de recursos lingüísticos necesarios para implementar tareas de procesamiento del lenguaje natural.
- ▶ Conocer referencias a corpus en español, así como a herramientas para poder trabajar en PLN

2.2. Características del texto como fuente de datos

Actualmente, en la era del Big Data, se generan grandes volúmenes de datos muy **heterogéneos** (imágenes, textos, datos de dispositivos IoT, etc.). Por este motivo, **para poder trabajar adecuadamente con una fuente específica de datos, es necesario conocer bien qué la caracteriza.** Esto ocurre con los textos en el ámbito del PLN. Los textos son una fuente más de datos disponible que contiene, como ya se vio en el tema previo, información muy relevante para **distintos casos de uso.**

Una de las primeras características asociadas al texto como dato es que el lenguaje **es composicional** y se puede tratar de manera discreta. Esto hace referencia a que el **significado general de un texto** (ej., una frase) **se puede componer desde el significado de sus elementos** (ej., palabras). Por ejemplo, la frase:

- ▶ He aprobado el examen.

El significado general de esta frase se constituye en base a las palabras individuales que lo componen, donde «He aprobado» hace referencia al sujeto (el que ha aprobado), mientras que «el examen» hace referencia al predicado. Combinando estos dos elementos se construye el significado de la frase original.

Análogamente, se pueden combinar textos para componer el significado de un texto más largo. Por ejemplo, la frase:

- ▶ He aprobado el examen tras mucho estudio.

Parte del significado de dicha frase se reúne en «He aprobado el examen», como hemos visto antes. Para obtener el significado de la frase completa, se construye con base en sus constituyentes que son «He aprobado el examen» junto con «tras mucho

estudio», de manera que el significado de la primera parte de la oración se combina con el de la segunda parte. El principio de composición se puede llevar a textos más amplios, como frases completas o incluso a nivel de documentos completos.

El principio de composición indica que la representación del significado de un texto se puede obtener desde el significado de los elementos que lo constituyen.

Este principio también aparece a nivel de palabras. Por ejemplo, la palabra *médicos* se compone del elemento raíz *médico* combinado con la letra *s* para especificar el significado final de la palabra, que en este caso es el significado de esa palabra raíz, pero en plural. El aspecto composicional a nivel de palabras está relacionado con el ámbito de la **morfología**, que se estudiará en el punto siguiente.

La aproximación de utilizar el principio de composición para construir el significado de un texto con base en sus componentes no es suficiente en algunas ocasiones. Esto ocurre en casos como cuando hay **ambigüedad léxica** (donde una palabra puede tener varios significados) o cuando se tienen **frases hechas** (donde el significado de la frase no se interpreta directamente desde sus palabras aisladas). Por ejemplo, la frase:

- ▶ Me he sentado en el banco del parque a ver el lago.

En ella aparece la palabra *banco*, que puede referirse tanto a una entidad financiera, como a un conjunto de peces o a un asiento. En este caso, el sentido es el de asiento y la forma por la que esto se sabe es por las palabras que aparecen en el contexto de la frase.

Esto está relacionado con la **perspectiva distribucional** de los textos, con la que se puede construir el significado de los elementos de un texto en base a las palabras que suelen aparecer en su contexto en otros textos.

Esta aproximación es no supervisada, ya que no es necesario que un anotador haga un trabajo previo para que se pueda deducir el significado de una palabra concreta. Sólo es necesario otros textos para que se pueda deducir ese significado gracias a esta **aproximación distribucional**.

A modo de ejemplo, textos como los siguientes servirían para generar el contexto necesario para esta perspectiva de la palabra *banco*:

- ▶ ... fue al banco a contratar una hipoteca.
- ▶ ... en el mar pude ver un banco de peces rojos.
- ▶ ... ellos se sentaron en un banco de madera.

Esto no sirve para explicar por qué esas palabras aparecen en un contexto determinado de la palabra *banco*, pero sí para identificar el significado de esta palabra en un texto concreto.

La perspectiva distribucional de los textos indica cómo el significado de sus elementos, como las palabras, se puede obtener de manera no supervisada con base al contexto que suele rodearlos.

Ahora bien, aunque los textos se pueden discretizar y analizar desde el punto de vista del principio de composición, complementando el análisis con la perspectiva distribucional, se deben tener en cuenta aspectos como la creación de palabras nuevas en una lengua (ej., neologismos), o que algunas palabras son mucho más frecuentes en textos que otras (por ejemplo, las palabras que son determinantes o preposiciones frente a verbos y sustantivos) (Zipf, 1949). Como consecuencia de esto, los algoritmos de PLN deben tener en cuenta estos aspectos para poder trabajar adecuadamente con los textos.

Además de la perspectiva distribucional para poder construir el significado de los elementos de un texto de manera no supervisada, **existen otras maneras de ver las relaciones entre las palabras de manera supervisada. Este es el caso de las ontologías**

semánticas como WORDNET (Fellbaum, 2010), donde se recogen las relaciones que existen entre palabras y otras unidades de texto. Por ejemplo, WORDNET serviría para saber que una *rueda* es una parte de un *coche*.

Los textos tienen otra característica que los diferencian de otros conjuntos de datos: el **aspecto secuencial**. En muchas ocasiones, el principio de composición y la perspectiva distribucional no son suficientes para construir el significado de una frase, como ocurre en el caso siguiente:

- ▶ Quiero información de hipotecas, pero no seguros.
- ▶ Quiero información de seguros, pero no hipotecas.

En ambas frases las palabras son las mismas, pero el significado es distinto. Tratar de construir el significado de la frase desde las palabras individuales con lo visto hasta ahora llevaría en ambos casos al mismo resultado final. Es por esto por lo que en PLN es necesario también atender al **aspecto secuencial de los textos**, de manera que viendo las palabras previas (ej., el «*no*» delante de «*seguros*» o «*hipotecas*») se puede construir mejor el significado global desde los elementos individuales. Ocurre también que en otras ocasiones se debe tener en cuenta no sólo las palabras previas, sino las posteriores, como en:

- ▶ Al campo no he ido, he ido a la playa.
- ▶ A la playa no he ido, he ido al campo.

En estos casos la información para saber que no ha ido al «*campo*» o a la «*playa*» aparece después de estas palabras, no antes, con lo que el aspecto secuencial de los textos es bidireccional.

El aspecto secuencial de los textos pone de manifiesto cómo es necesario en muchas ocasiones tener en cuenta la secuencialidad (de manera bidireccional) del texto en la construcción de su significado.

En algunos casos la construcción del significado puede ser aún más compleja, como en el caso siguiente:

- ▶ Pedro ve un cuadro de su madre.

Esta frase se podría interpretar de dos maneras: el cuadro pertenece a la madre, o en el cuadro aparece la madre. La construcción del significado en estos casos debería tener en cuenta, además del aspecto composicional, distribucional y secuencial, la **relación sintáctica** entre las palabras dentro de la oración.

Aquí, por tanto, se ve otro tipo de ambigüedad que se puede encontrar en los textos: **la ambigüedad sintáctica, donde una misma frase puede tener distintas interpretaciones.**

En el vídeo *El texto como dato* se presentará qué caracteriza a los textos como dato y cómo poder trabajar con ellos a nivel computacional.



Accede al vídeo

2.3. Morfología

La Real Academia Española (RAE) en su diccionario de la lengua española define la palabra **morfología**: «*De morfo- y -logía. 2. f. Gram. Parte de la gramática que estudia la estructura de las palabras y de sus elementos constitutivos.*

Por tanto, la **morfología estudia la forma en que las palabras se descomponen en partes indivisibles, las cuales tienen un significado.**

A estas unidades mínimas se les llama **morfemas** y, por ejemplo, la palabra «mujeres» contiene dos morfemas: el primero es *mujer* y el segundo es *es*.

Un morfema es la unidad mínima que compone una palabra y que es capaz de expresar un significado.

De hecho, no se debe confundir esta definición de morfema con la de morfema gramatical, al que en algunos textos se le llama de la misma manera. Se entiende como **morfema gramatical** al que tiene un significado gramatical, es decir, significado sobre:

- ▶ El género (masculino o femenino).
- ▶ Número (singular o plural).
- ▶ Persona (p. ej. tercera persona del singular).
- ▶ Modo y tiempo (p. ej. modo indicativo y tiempo futuro).

En contraposición al morfema gramatical está el **lexema**.

Un lexema sería el morfema que conforma la raíz o parte invariante de la palabra y que es la mínima unidad con significado léxico, por lo que proporciona el significado principal de la palabra.

Para el ejemplo de la palabra «mujeres», el morfema *mujer* sería el lexema o raíz de la palabra con significado léxico y el morfema *es* sería el morfema gramatical que añade significado gramatical y expresa el número plural. El singular, que también sería en este caso «mujer», contiene a su vez dos morfemas: el lexema *mujer* y un morfema cero de número singular.

Un morfema cero es un morfema gramatical sin realización fonética, pero que tiene significado gramatical.

Una palabra como «cantábamos» contiene tres morfemas:

1. El primero es el lexema *cant*, que indica el significado léxico de la palabra, es decir, el acto de que una persona produzca con la voz sonidos melodiosos.
2. El segundo es el morfema *aba*, que proporciona el significado gramatical de modo indicativo y tiempo pasado.
3. El tercero es el morfema *mos*, que indica el significado gramatical de primera persona del plural.

En el procesamiento del lenguaje natural, el ámbito de la morfología computacional trata de reconocer de forma automática los morfemas que contiene una palabra.

El análisis morfológico es el proceso de análisis que se realiza para obtener la separación de la palabra en morfemas.

Conocer la morfología de las palabras es importante para, si se está buscando en un texto el verbo «pensar», que se reconozca también la fórmula *piénsalo*, aunque ambas no se compongan de la misma cadena de caracteres en la raíz. Además, a la hora de generar de forma automática frases o expresiones de texto se debe conocer la morfología de las palabras a utilizar para encajar, por ejemplo, el género y el número de un nombre con el adjetivo que la acompaña o la persona del verbo con el sujeto de la frase.

El análisis morfológico en inglés, lengua en la que se realiza la mayoría de la investigación en el ámbito de la morfología computacional, es más simple que el análisis morfológico en español. Esto se debe principalmente a que en el español se dan muchas variedades diferentes de las formas en las terminaciones de las palabras y también se dan más alteraciones en la raíz de estas.

En el vídeo Morfología se verá qué es y las bases para entenderlo a nivel lingüístico y podamos aplicarlo en el procesamiento del lenguaje natural a nivel computacional.



Accede al vídeo

2.4. Técnicas de normalización de textos

El principio de composición visto antes, aunque no siempre es suficiente para construir el significado de determinadas frases, es una aproximación suficiente para cubrir muchos tipos de textos y casos de uso. Dado que este sirve para obtener la construcción del significado de un texto desde sus componentes, sirve también de cara a transformar un texto en distintas **variables de entrada** para poder entrenar un **modelo de aprendizaje automático**.

Anteriormente, se vio como un modelo de aprendizaje automático puede ser usado en tareas como el análisis del sentimiento de un texto y saber si este es positivo, negativo o neutro. Para poder llevar a cabo esta tarea con, por ejemplo, un modelo supervisado, se necesitaría tener un conjunto de textos anotados con esas tres categorías de sentimientos, y, como entrada, el modelo recibiría una serie de variables extraídas desde el texto siguiendo el principio de composición (por ejemplo, qué palabras aparecen en él).

La transformación de un texto en los elementos que lo componen para poder trabajar con ellos posteriormente es una de las tareas habituales de PLN, y sigue un flujo (pipeline) de normalización en distintos pasos.

Primer paso

Se denomina **tokenización**. Consiste en **partir de una cadena de texto** (como una frase) **para descomponerla en sus términos o componentes** (ej., las palabras que la forman). Por ejemplo, para la siguiente frase:

- ▶ Despues de estar estudiando 2 horas, he decidido estudiar 2 horas más.

Su descomposición en *tokens* sería la siguiente:

«Después», «de», «estar», «estudiando», «2», «horas», «he», «decidido», «estudiar», «2», «horas», «más».

Este proceso de tokenización **ha seguido** en este caso **una de las aproximaciones más habituales**, en las que se separa el texto en palabras en base a los **espacios en blanco** que aparecen en la secuencia de texto. Previamente a esto, se han eliminado signos de puntuación, como la coma o el punto final. Así, como resultado, tenemos una lista ordenada con las palabras que aparecen en el texto.

No obstante, si se analiza la lista final tras el proceso de tokenización, se pueden ver algunos **aspectos a considerar**. Por ejemplo, hay **palabras que aparecen varias veces**, como «*horas*». También, ocurre que hay **palabras con mayúsculas**, como «*Después*» por ser la primera palabra de la frase. También se ve que **no todo son palabras**, sino que aparecen números, como el *2*. Adicionalmente, se ve que **algunas palabras similares aparecen expresadas de manera distinta**, como es el caso de «*estudiando*» y «*estudiar*» por sus conjugaciones verbales. Estos aspectos también se tienen en cuenta en el proceso del flujo de normalización.

El proceso de tokenización busca separar un texto en partes más pequeñas (*tokens*) como palabras o partes de una frase.

Un primer aspecto para comentar es que el proceso de tokenización que se ha seguido de separar palabras tras eliminar signos de puntuación, no es correcto para todos los casos. Por ejemplo, si se tienen palabras en inglés como *Dr.*, *o'clock*,

doctor's (el apóstrofe 's en inglés), o formatos de fechas como *2022-06-01* o de horas como *08:30*, no sería correcto eliminar sin más estos signos y luego separar por espacios en blanco. Es necesario por tanto en el flujo de normalización diferenciar entre comas y puntos respecto a otro tipo de signos. Esto se puede realizar de distintas maneras, como por ejemplo mediante el uso de reglas o heurísticas, o usando bases de datos que sirvan para estandarizar algunos de estos términos.

Otra de las limitaciones en la aproximación de la tokenización que se ha visto es **cómo tratar los nombres propios**. Por ejemplo, si en un texto aparece *Nueva York*, no tendría demasiado sentido separarlo en dos tokens distintos, uno para *Nueva* y otro para *York*. Esto también ocurre con palabras compuestas, como *Estado del Arte* o *a priori*. En muchos casos se plantea en estos escenarios hacer una **tokenización en dos niveles**, de manera que primero se lleve a cabo una tokenización a bajo nivel, seguida de una a alto nivel que sirva para generar tokens que tengan más sentido desde un punto de vista semántico. Con ello, *Nueva York* pasaría a ser un único token expresado como *Nueva York*. Esto se puede llevar a cabo de distintas maneras, como por ejemplo mediante aproximaciones estadísticas que tienen en cuenta palabras que suelen ser coocurrentes.

Desde la perspectiva del principio de composición, según la tarea para la que se quiera usar el texto, se puede ver que no todos los tokens son igual de importantes. Si, por ejemplo, queremos identificar el sentimiento positivo, negativo o neutro de la frase

- ▶ Estoy contento de haber comprado este libro.

Las palabras como «de» y «este» no aportan mucha información para la tarea que se quiere llevar a cabo. Por este motivo, se presenta el siguiente paso.

Segundo paso

Es la **eliminación de stopwords**, lo que suele incluir palabras como artículos, conjunciones o preposiciones, e incluso pronombres en algunas ocasiones. Una manera de **detectar estas palabras** es mediante el **uso de diccionarios específicos** para cada lengua, que contengan la lista de palabras a eliminar. La eliminación de **stopwords** es una técnica que no siempre se va a emplear, sino que **dependerá de cada caso de uso concreto**. Para algunas tareas de PLN puede ser necesaria la información de esas palabras.

La eliminación de **stopwords** se usa para quitar tokens que son muy habituales y que aportan poca información, como es el caso de artículos, conjunciones o preposiciones cuando esos tokens representan palabras.

Tercer paso

Es el **tratamiento de las mayúsculas**. En una frase como:

- ▶ Días y días pasaron sin noticias nuevas.

Aparece la palabra «*días*» tanto en mayúscula como en minúscula. Si no se tratan las mayúsculas, «*Días y días*» serían dos tokens distintos a todos los efectos. Para estos casos, se puede llevar a cabo un paso de **normalización que elimine las mayúsculas de la frase**, de manera que ambas palabras queden representadas por el mismo token. **El tratamiento de las mayúsculas no siempre pasa por eliminarlas**. Por ejemplo, en la frase:

- ▶ Su amiga Estrella estaba de vacaciones, y vio una estrella en el cielo.

En este caso, «*Estrella*» hace referencia a un nombre de persona (nombre propio), mientras que «*estrella*» hace referencia a un nombre común. En este caso, si se quitase la mayúscula de «*Estrella*», se representarían ambas palabras con el mismo token, lo que no sería correcto. **El objetivo es por tanto eliminar las mayúsculas, pero**

solo cuando sea necesario. Esto se puede hacer mediante el uso de reglas (por ejemplo, eliminar las mayúsculas sólo de la primera palabra de la frase), o mediante soluciones más sofisticadas, como los algoritmos de **reconocimiento de entidades nombradas (NER, Named Entity Recognition)** que identificarían que tokens hacen referencia a personas y organizaciones de cara a dejar las mayúsculas en esos casos.

El tratamiento de las mayúsculas busca tratar adecuadamente las mayúsculas en los tokens, eliminándolas cuando tanto un token con mayúsculas como otro con minúsculas se refieren a la misma palabra.

Existen otro tipo de tratamientos que se pueden considerar para homogeneizar las representaciones de los tokens. Por ejemplo, en el primer texto se veía que «dos» se representaba por su número, «2». Puede ocurrir que en un documento coexistan ambas representaciones, de manera que sería necesario representar ambas de la misma manera.

Cuarto paso

También ocurre en algunos tipos de textos (por ejemplo, tweets), que aparecen caracteres no alfanuméricos (*hashtags*, @, etc.), de manera que en el flujo de normalización también se debe tener en cuenta qué hacer con estos caracteres a la hora de representar un texto por sus tokens normalizados. En algunas ocasiones puede servir eliminarlos (por ejemplo, eliminar algunos signos de exclamación que tenga un texto), pero dependerá de cada caso de uso y de si esto supone una pérdida relevante de información o no. Este paso se suele denominar **tratamiento de caracteres especiales**.

El orden de los pasos del proceso de normalización no siempre tiene porque ser el mismo. El proceso de tokenización es el primer paso, pero, según el caso concreto, el tratamiento de mayúsculas y de caracteres especiales puede ir antes de la eliminación de *stopwords*.

Por ejemplo, «Principado de Asturias» se detectaría como una entidad nombrada de un lugar concreto, y se podría expresar con un token como «Principado» «de» «Asturias» antes de eliminar la *stopword* «de».

Tras estos pasos, **una fase que incluye el proceso de normalización** para algunos casos de uso es la **obtención del lema o de la raíz** de las palabras. Para un texto como:

- ▶ Ayer jugaron al mismo deporte que han jugado hoy.

Se tienen dos verbos, «jugaron» y «jugado» en dos conjugaciones verbales distintas. Con la obtención del lema o de la raíz de estas palabras se busca representar ambas con un mismo token. **Para el caso de la obtención del lema se lleva a cabo el proceso de lematización**, con el que estas palabras quedan representadas por su lema, que sería *jugar*. Para esto, se pueden usar diccionarios donde se tengan las equivalencias entre distintas palabras y lemas, entre otras técnicas.

El proceso de obtención de la raíz, conocido como **stemming**, es distinto. Con él se busca **obtener el lexema eliminando el resto de los morfemas**. Por ejemplo, para estas dos mismas palabras sería *jug*.

Existen distintos algoritmos para llevar a cabo esta tarea, como por ejemplo el algoritmo Snowball (Porter, 2001).

Uno de los problemas de la aproximación de **stemming** frente a la de **lematizar** es que hay ocasiones en las que dos palabras que deberían tener **una misma representación no la tienen**.

Por ejemplo, para casos como «jugaron» y «yo juego», ambas palabras se representarían como *jugar* en el caso de la lematización, pero se representarían como *jug* y *jueg* respectivamente con el **stemming**. Otro aspecto importante que remarcar es que no en todos los casos de uso es aconsejable llevar a cabo un proceso

de lematización o *stemming*, ya que con este se puede perder información relevante de un texto. Para un texto como:

- ▶ El número de ingenieras egresadas ha aumentado en los últimos años.

Si se llevase a cabo un proceso de lematización, «ingenieras» y «egresadas» pasaría a ser «ingeniero» y «egresado», de manera que se estaría perdiendo una parte importante de la información que está expresando el texto.

En los procesos de lematización y de stemming se busca simplificar las representaciones de las palabras a través de los tokens usando sus lemas o sus lexemas respectivamente.

Como aspecto final, estos flujos de normalización son **útiles** para llevar a cabo la fase de **preprocesado de muchos textos** para luego utilizarlos en distintas tareas de PLN, principalmente cuando la fuente de origen es un texto en crudo. Existen otros formatos de datos que incluyen, junto con los textos, información relevante con la que no sería necesario en ocasiones llevar a cabo algunas de estas fases de normalización, como ocurre en muchos casos donde el texto está en **formato XML** (Extensible Markup Language).

2.5. Recursos lingüísticos

La implementación de diferentes tareas de procesamiento del lenguaje natural requiere de una serie de recursos lingüísticos. Entre las bases de conocimiento lingüístico destacan:

- ▶ Los diccionarios.
- ▶ Los lexicones.
- ▶ Los tesauros.

- **Las bases de datos de relaciones léxicas.**

Además, de estas bases de conocimiento lingüístico también cabe destacar las colecciones de textos o de recursos del habla llamados **corpus**.

A continuación, se presentan las definiciones de los diferentes tipos de recursos lingüísticos y, en las siguientes secciones, se presentan la base de datos de relaciones léxicas WordNet y algunos corpus en español.

Diccionario

Es el repertorio donde se agrupa, según un orden determinado, las **palabras de una lengua acompañadas de su definición o explicación**. Los diccionarios incluyen una descripción detallada y comprensible para un humano de los diferentes sentidos de las palabras.

Lexicón

Un diccionario que contiene información morfológica. Este diccionario morfológico es un repertorio donde se recogen una lista de morfemas y la información básica sobre estos.

Tesoro

También llamado *thesaurus*, *thesauri* o tesoro. Se refiere al diccionario que contiene una lista de significados de las palabras y las relaciones entre estos significados.

- En la literatura, un tesoro contiene una lista de palabras con sus sinónimos y sus antónimos.
- En lingüística, un tesoro reúne el conocimiento sobre las relaciones de hiperonimia/hiponimia y meronimia/holónimia representadas en la propia

estructura jerárquica del tesauro. Es decir que, la jerarquía del tesauro modela la relación *is-a* y la relación parte todo de los sentidos de las palabras. Además, un tesauro también puede agrupar conocimiento sobre otras relaciones semánticas como la sinonimia o antonimia.

Bases de datos de relaciones léxicas

Una generalización de los tesauros, es decir, un tipo de diccionario que recoge conocimiento sobre cualquier relación semántica entre sentidos de las palabras. Estas bases contienen un conjunto de lemas, cada uno de los cuales está anotado con el posible conjunto de sentidos de la palabra y las relaciones entre esos sentidos.

Corpus lingüístico

Es una colección de textos representativos de una lengua que se utilizan para el análisis lingüístico. Se puede distinguir entre:

- ▶ Los corpus textuales, que recogen extractos de libros, revistas, periódicos o cualquier otra fuente escrita.
- ▶ Los corpus orales que son colecciones de habla, es decir extractos de audios provenientes de fuentes como puede ser la radio o la televisión.

Los corpus pueden estar anotados o etiquetados de forma que las palabras que lo conforman presentan, además, algún tipo de información lingüística, por ejemplo, información sintáctica, semántica o pragmática, son los conocidos como **corpus etiquetados**.

Los primeros corpus disponibles *online* aparecieron en la década de 1960. Uno de los pioneros fue el **Brown Corpus**, un corpus del inglés americano desarrollado en 1963 por la Brown University y que contenía una colección de un millón de palabras extraídas de 500 textos de diferentes géneros: periódicos, novelas, no ficción, académico, etc. (Kucera y Francis, 1967). Al principio el corpus no estaba etiquetado, pero con el paso de los años se etiquetó con información sobre las categorías gramaticales (POS).

Hoy en día los corpus tienden a ser mucho más extensos que el Brown Corpus. Por ejemplo, el corpus etiquetado con información morfosintáctica **WSJ**, se trata de la colección del millón de palabras que se publicaron en los artículos del Wall Street Journal (WSJ) en el año 1989.

Para ampliar la primera versión del *Wall Street Journal* (**WSJ**) ingresa al siguiente enlace web: <https://catalog.ldc.upenn.edu/LDC2000T43>

2.6. Corpus en español

En las secciones anteriores se ha comentado sobre algunos corpus, esas colecciones de textos para el análisis lingüístico, utilizados en el procesamiento del lenguaje natural en inglés. En el ámbito hispánico son instituciones como la Real Academia Española, incluida en la Asociación de Academias de la Lengua Española, o el Instituto Cervantes las que se han ocupado de producir recursos lingüísticos que puedan ser utilizados en el procesamiento del lenguaje natural en español.

La Biblioteca Virtual Miguel de Cervantes reúne algunos corpus en español, por ejemplo:

- ▶ Corpus de sonetos del siglo de oro: <http://goldenage.cervantesvirtual.com/>
- ▶ Corpus diacrónico del español histórico IMPACT-es (documentación):
<http://data.cervantesvirtual.com/blog/documentacion-corpus-diacronico/>

El Corpus diacrónico del español histórico IMPACT-es (Sánchez-Martínez, Martínez-Sempere, Ivars-Ribes y Carrasco, 2013) contiene 86 obras pertenecientes a la Biblioteca Virtual Miguel de Cervantes e impresas entre los años 1482 y 1647. Este corpus viene acompañado por un lexicón que enlaza más de 10 000 lemas con las diferentes variantes de las palabras encontradas en los documentos. Además, las palabras más frecuentes del corpus están anotadas con su lema, categoría gramatical

y su forma moderna equivalente. Esto permite efectuar búsquedas de forma muy eficiente (Carrasco et al., 2015) a través del interfaz web.

- ▶ Accede al IMPACT-es: <http://data.cervantesvirtual.com/blog/diasearch/>
- ▶ Disponible para descarga: <https://www.digitisation.eu/tools-resources/language-resources/impact-es/>

Un ejemplo de búsqueda se presenta en la Figura 1 donde vemos el resultado de la búsqueda en el corpus diacrónico del español histórico (IMPACT-es) de obras literarias en las que aparece una expresión del tipo «hacer algo de», donde el verbo *hacer* puede aparecer en cualquiera de sus formas, como en español antiguo, y la palabra «algo» puede ser cualquier nombre, como por ejemplo *haz examen de o haciendo burla de*.

The screenshot shows the BVMC.Labs website interface. At the top, there is a navigation bar with links for 'DATOS ENLAZADOS', 'NOVEDADES', and 'LABS'. Below the navigation bar, there are language selection buttons for 'ESPAÑOL' and 'ENGLISH', and a search bar containing the query 'lemma#hacer pos#n de'. A red 'BUSCAR' button is located to the right of the search bar. The main content area displays search results under the heading '60 resultados para "lemma#hacer pos#n de"'. Each result entry includes a thumbnail image, the title of the work, the author, the edition information, and a snippet of text containing the search term in orange. The results are categorized into three sections: 'Corpus diacrónico', 'Comedias y Entremeses', and 'Otras obras'. The 'Corpus diacrónico' section contains entries such as 'Entremes de la elección de los alcaldes de Dagánco / Miguel de Cervantes Saavedra; edición publicada por Rodolfo Schevill y Adolfo Bonilla' and 'Novela del casamiento engañoso / Miguel de Cervantes Saavedra; edición publicada por Rodolfo Schevill y Adolfo Bonilla'. The 'Comedias y Entremeses' section contains entries such as 'Ocho comedias y ocho entremeses nuevos / Miguel de Cervantes Saavedra; edición publicada por Rodolfo Schevill y Adolfo Bonilla'. The 'Otras obras' section contains entries such as 'Cervantes Saavedra, Miguel de. Novelas del autor que no fueron publicadas en su vida'. The bottom of the page features a footer with the BVMC.Labs logo and a copyright notice: '© BVMC.Labs 2018'.

Figura 1. Resultado de la búsqueda en el corpus diacrónico del español histórico. Fuente: BVMC.Labs, s. f.

La Real Academia Española se fijó a finales del siglo pasado el objetivo de generar recursos lingüísticos para mostrar la evolución del español en distintas épocas y su funcionamiento actual en los diferentes países que conforman la comunidad lingüística hispánica. Es por eso por lo que, a partir de 1996, la RAE en colaboración con las otras academias de la Asociación de Academias de la Lengua Española crearon

el **Corpus de Referencia del Español Actual (CREA)** y el **Corpus del Español del Siglo XXI (CORPES XXI)**.

Para completar el estudio de esta sección puedes leer el capítulo 7.1 *CREA* y

CORPES, corpus de referencia del español del siguiente libro:

Sánchez, M. (2016). CREA y CORPES, corpus de referencia del español. *Tecnologías del lenguaje en España: comunicación inteligente entre personas y máquinas* (119-124). Fundación Telefónica y Ariel.

https://www.fundaciontelefonica.com/arte_cultura/publicaciones-listado/pagina-item-publicaciones/itempubli/565/

2.7. Herramientas y librerías para el PLN

Existen numerosas herramientas y librerías que facilitan el desarrollo e implementación de diferentes tareas relacionadas con el procesamiento del lenguaje natural. Una de las herramientas más utilizadas para la implementación en Python de aplicaciones para el procesamiento del lenguaje natural es Natural Language Toolkit (NLTK). NLTK está disponible para Windows, Mac OS X y Linux, es *open source* y su código se distribuye bajo la licencia Apache License Version 2.0.

NLTK es una plataforma que proporciona interfaces a más de cincuenta corpus y recursos léxicos, como por ejemplo WordNet, además de una serie de librerías para realizar tareas de clasificación, obtención de tokens, derivación, etiquetado morfosintáctico, análisis sintáctico y análisis semántico. NLTK ha sido desarrollado principalmente para el procesamiento del lenguaje natural en inglés. Sin embargo, esta plataforma tiene la flexibilidad necesaria para poder realizar procesamiento del lenguaje natural en otros idiomas como el español.

NLTK se puede utilizar para entrenar un etiquetador morfosintáctico en español a partir de un corpus etiquetado y también permite incorporar un etiquetador externo que sea capaz de analizar el español.

Accede a NLTK a través del aula virtual o desde la siguiente dirección web:

<http://www.nltk.org/>

Otra librería de Python para PLN es **spaCy**, muy usada tanto en el ámbito académico como en el industrial. SpaCy incorpora ya distintos flujos de normalización de textos (como los que se han visto previamente) de manera que se agilice todo el preprocesado de los textos para poder trabajar con ellos en las distintas tareas de PLN. Tiene soporte para más de sesenta idiomas, e incluye distintos modelos neuronales recientes para distintas tareas de PLN como, por ejemplo, el reconocimiento de entidades nombradas que se comentó previamente. Incluye también modelos preentrenados como BERT, que se verán con más detalle en otros capítulos de la asignatura. SpaCy es un software *open source* para uso comercial, con licencia MIT.

Accede a spaCy a través del aula virtual o desde la siguiente dirección web:

<https://spacy.io/>

También existe la librería **Gensim (Generate Similar)** de Python para trabajar con PLN. Con Gensim se pueden llevar a cabo tareas como *topic modelling*, indexado de documentos o búsquedas de documentos mediante el uso de técnicas como vector *embeddings*, los cuales se verán en detalle más adelante en la asignatura.

Accede a Gensim a través de la siguiente dirección web:

<https://radimrehurek.com/gensim/>

Además de estos recursos en Python, cabe destacar algunas librerías para el desarrollo en Java de aplicaciones en el ámbito del procesamiento del lenguaje natural:

- ▶ LingPipe: <http://alias-i.com/lingpipe/index.html>
- ▶ Apache OpenNLP: <https://opennlp.apache.org/docs/>
- ▶ Stanford NLP: <http://nlp.stanford.edu/software/>
- ▶ Stanford CoreNLP: <https://stanfordnlp.github.io/CoreNLP/>

Stanford CoreNLP incorpora modelos para realizar tareas de procesamiento del lenguaje natural en español. Por lo tanto, permite gestionar algunas características típicas de este idioma, como por ejemplo la separación de contracciones de palabras (del=de + el) durante el proceso de obtención de tokens o la identificación de los tiempos y modos de los verbos (presente de indicativo) en el etiquetado morfosintáctico. Stanford CoreNLP está implementado en Java, pero proporciona acceso por la línea de comandos lo que permite su integración en Python. Esta característica es muy útil si se quiere utilizar el etiquetado morfosintáctico para el español del Stanford CoreNLP en un código desarrollado en Python utilizando la plataforma NLTK.

En el caso de querer realizar una implementación de tareas de procesamiento del lenguaje natural en node.js se puede utilizar la librería Natural.

Accede a la guía de la librería Natural:

<https://github.com/NaturalNode/natural/blob/master/README.md>

2.8. Referencias bibliográficas

Aggarwal, C. C. (2018). Text Preparation and Similarity Computation. *Machine Learning for Text* (17-25). Springer International Publishing.

Carrasco, R. C., Martínez-Sempere, I., Mollá-Gandía, E., Sánchez-Martínez, F., Candela-Romero, G. y Escobar-Esteban M. P. (2015). Linguistically-Enhanced Search over an Open Diachronic Corpus. En A. Hanbury, G. Kazai, A. Rauber y N. Fuhr (Eds.), *Advances in Information Retrieval. Lecture Notes in Computer Science*, 9022. https://doi.org/10.1007/978-3-319-16354-3_89

Eisenstein, J. (2019). *Introduction to Natural Language Processing*. MIT press.

Poli, R., Healy, M. y Kameas, A. (2010). WordNet. Fellbaum, C. (A.), In *Theory and applications of ontology. Computer applications* (231-243). Springer.

Jurafsky, D. y Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall.

Kucera, H. y Francis, W. N. (1967). *Computational analysis of present-day American English*. Brown University Press.

Sánchez-Martínez, F., Martínez-Sempere, I., Ivars-Ribes, X. y Carrasco, R.C. (2013). An open diachronic corpus of historical Spanish. *Language Resources and Evaluation*, 47(4), 1327-1342.

Schütze, H., Manning, C. D., & Raghavan, P. (2008). *Introduction to information retrieval* (39, 234-65). Cambridge University Press.

Porter, M. F. (2001). *Snowball: A language for stemming algorithms*. Computer Science.

Zipf, G. K. (1949). *Human Behavior and The Principle of Least Effort*. Addison-Wesley Press.

A fondo

Machine Learning for Text

Aggarwal, C. C. (2018). Text Preparation and Similarity Computation. *Machine Learning for Text* (17-25). Springer International Publishing.

Se recomienda la lectura del segundo capítulo, donde se realiza una explicación detallada de qué caracteriza al texto como dato, así como de las fases que se siguen para normalizarlo y poder trabajar con él para llevar a cabo las distintas tareas de PLN.

Natural Language Processing with Python

Bird, S., Klein, E. y Loper, E. (2014). *Natural Language Processing with Python*. Sebastopol. O'Reilly. <https://www.nltk.org/book/>

Este libro proporciona una introducción práctica a la programación para el procesamiento del lenguaje natural. El libro está escrito por los creadores de NLTK y guía al lector a través de los fundamentos de escribir programas en Python, trabajar con corpus, categorizar texto, analizar estructuras lingüísticas. La versión en línea del libro se ha actualizado para Python 3 y NLTK 3. La versión del libro editada por O'Reilly Media en 2009 presentaba la versión original de NLTK basada en Python 2.

WordNet: An Electronic Lexical Database

Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database*. Cambridge. MIT Press.
<http://mitpress.mit.edu/books/wordnet>

Este libro tiene dos propósitos. Primero, comentar el diseño de WordNet y las motivaciones teóricas detrás de este. En segundo lugar, proporcionar un análisis de las aplicaciones de WordNet, incluida la identificación del sentido de las palabras, la recuperación de información, la selección de preferencias de los verbos y las cadenas léxicas.

Test

1. El principio de composición:

- A. Indica que la representación del significado de una frase se puede obtener desde la composición del significado de las palabras que la constituyen.
- B. Indica que la representación del significado de un documento se puede obtener desde la composición de las frases que lo forman.
- C. Indica que la representación del significado de una palabra está relacionada con su morfología.
- D. Ninguna de las anteriores.

2. Indica cuál de estas afirmaciones es correcta:

- A. El principio de composición es suficiente para representar el significado de cualquier texto.
- B. La perspectiva distribucional indica que todas las palabras tienen el mismo peso en la representación del significado de un texto.
- C. El aspecto secuencial muestra cómo es irrelevante el orden de las palabras para la construcción del significado de un texto.
- D. Ninguna de las anteriores

- 3.** Indica cuál de estas afirmaciones es cierta respecto a la normalización de textos:
- A. El proceso de tokenización suele ser la última fase del pipeline de normalización de los textos.
 - B. Siempre es aconsejable usar técnicas de *stemming* en el pipeline de normalización.
 - C. El proceso de lematización puede hacer que se pierda información importante para la construcción del significado de una frase.
 - D. La eliminación de stopwords se hace para quitar palabras poco habituales en una lengua.
- 4.** El proceso de *stemming* de la palabra «saltábamos» da como resultado:
- A. «Saltar».
 - B. «Salto».
 - C. «Salt».
 - D. «Saltaba».
- 5.** El proceso de lematización de la palabra «saltábamos» da como resultado:
- A. «Saltar».
 - B. «Salto».
 - C. «Salta».
 - D. «Saltaba».
- 6.** La mínima unidad que forma una palabra y tiene un significado es:
- A. El morfema gramatical.
 - B. El lexema.
 - C. El morfema.
 - D. El lema.

7. Indica las afirmaciones correctas sobre el lexema:

- A. Tienen un significado gramatical.
- B. Conforma la raíz de una palabra.
- C. Tienen un significado léxico.
- D. Proporciona significado a la palabra.

8. Indica las afirmaciones correctas sobre los recursos lingüísticos:

- A. Un lexicón es un tipo de diccionario que contiene una lista de lemas.
- B. El diccionario es el repertorio donde se recogen las palabras de una lengua acompañadas de su definición o explicación.
- C. El tesoro es un tipo de diccionario que contiene información sobre el léxico. aspecto secuencial muestra cómo es irrelevante el orden de las palabras para la construcción del significado de un texto.
- D. Las bases de datos de relaciones léxicas contienen un conjunto de lemas anotados con sus posibles sentidos.

9. Indica las afirmaciones correctas sobre un tesoro:

- A. Debe contener información sobre las relaciones de sinonimia.
- B. Los términos que conforman un tesoro se relacionan entre sí para mostrar las relaciones entre significados.
- C. Si la jerarquía del tesoro se basa en las relaciones de hiperonimia/hiponimia, cualquier término del tesoro es un descendiente del concepto de raíz.
- D. Debe contener información sobre las relaciones de meronimia.

10. Indica las afirmaciones correctas sobre Natural Language Toolkit (NLTK):

- A. Proporciona herramientas para analizar textos en español.
- B. Proporciona interfaces a diferentes corpus y otros recursos léxicos como por ejemplo WordNet.
- C. Permite integrar software de terceros por ejemplo la API de Stanford CoreNLP.
- D. Es la plataforma más utilizada para desarrollar aplicaciones de procesamiento del lenguaje natural en Python.

Procesamiento del Lenguaje Natural

Etiquetado morfosintáctico (POS *tagging*)

Índice

Esquema	3
Ideas clave	4
3.1. Introducción y objetivos	4
3.2. Categorías morfosintácticas o gramaticales	4
3.3. Funcionamiento y características del etiquetado morfosintáctico	7
3.4. Etiquetado morfosintáctico basado en modelos ocultos de Markov (HMM)	10
3.5. Etiquetado morfosintáctico basado en aprendizaje automático	22
3.6. <i>Named Entity Recognition</i>	24
3.7. Referencias bibliográficas	25
A fondo	28
Test	29

Esquema



3.1. Introducción y objetivos

A continuación, se estudiará el etiquetado morfosintáctico y cómo calcularlo, haciendo hincapié en los modelos ocultos de Markov (*Hidden Markov Models*).

Objetivos

- ▶ Identificar las diferentes categorías morfosintácticas o también llamadas gramaticales.
- ▶ Entender el funcionamiento y las características del etiquetado morfosintáctico.
- ▶ Aplicar un método estocástico basado en modelos ocultos de Markov (HMM) para realizar el etiquetado morfosintáctico.
- ▶ Describir diversos métodos basados en aprendizaje automático para realizar el etiquetado morfosintáctico.

3.2. Categorías morfosintácticas o gramaticales

La Real Academia Española (RAE, s. f.) en su diccionario de la lengua española define la palabra **morfosintaxis** como: «1. f. Ling. Parte de la gramática que integra la morfología y la sintaxis». Tal como se ha presentado en los temas anteriores:

La morfología estudia la estructura de las palabras y la sintaxis, el modo en que se combinan las palabras. Por lo tanto, la morfosintaxis aúna la morfología y la sintaxis para determinar las diferentes partes de la oración, llamadas *part-of-speech* (POS) en inglés.

Las **categorías morfosintácticas** del lenguaje, que en español también se llaman categorías **gramaticales**, proporcionan una clasificación de las diferentes partes de la oración, es decir, una clasificación de las palabras según su tipo.

Las categorías gramaticales del español, según la clasificación clásica, son nueve: sustantivo o nombre, determinante, adjetivo, pronombre, verbo, adverbio, preposición, conjunción e interjección. Las definiciones de las diferentes clases de palabras o categorías gramaticales se presentan en la siguiente tabla.

Categorías morfosintácticas o gramaticales	
Sustantivo o nombre	Clase de palabras cuyos elementos poseen género y número, forman sintagmas nominales con diversas funciones sintácticas y designan entidades de diferente naturaleza.
Determinante	Clase de palabras cuyos elementos determinan al sustantivo o al grupo nominal y se sitúan generalmente en posición prenominal. El artículo definido y los demostrativos son determinantes.
Adjetivo	Clase de palabras cuyos elementos modifican a un sustantivo o se predicen de él y denotan cualidades, propiedades y relaciones de diversa naturaleza. Ejemplos: inteligente, amplio, numérico, etc.
Pronombre	Clase de palabras cuyos elementos hacen las veces del sustantivo o del sintagma nominal y que se emplean para referirse a las personas, los animales o las cosas sin nombrarlos. Ej.: ella, esto, quién.
Verbo	Clase de palabras cuyos elementos pueden tener variación de persona, número, tiempo, modo y aspecto.
Adverbio	Clase de palabras cuyos elementos son invariables y tónicos, están dotados generalmente de significado léxico y modifican el significado de varias categorías, principalmente de un verbo, de un adjetivo, de una oración o de una palabra de la misma clase.
Preposición	Clase de palabras invariables cuyos elementos se caracterizan por introducir un término, generalmente nominal u oracional, con el que forman grupo sintáctico.
Conjunción	Clase de palabras invariables, generalmente átonas, cuyos elementos manifiestan relaciones de coordinación o subordinación entre palabras, grupos sintácticos u oraciones.
Interjección	Clase de palabras invariables, con cuyos elementos se forman enunciados exclamativos, que manifiestan impresiones, verbalizan sentimientos o realizan actos de habla apelativos.

Tabla 1. Definiciones de las diferentes categorías morfosintácticas. Fuente: elaboración propia adaptado de la Real Académica Española (RAE).

En el vídeo *Morfosintaxis y partes de la oración* se presentará qué es la morfosintaxis, parte de la gramática que integra la morfología y la sintaxis.



Accede al vídeo

Conocer las partes de la oración (categorías morfosintácticas o gramaticales) es útil debido a la gran cantidad de información que brindan sobre una palabra y sus vecinos.

- ▶ Saber si una palabra es un sustantivo o un verbo nos dice mucho acerca de las palabras vecinas, por ejemplo, los sustantivos pueden ir precedidos de determinantes o seguidos de adjetivos.
- ▶ Y también sobre la estructura sintáctica, por ejemplo, los sustantivos son generalmente parte de los sintagmas nominales.

Por estas razones, el etiquetado morfosintáctico es un componente muy importante del análisis sintáctico que se va a estudiar a continuación.

3.3. Funcionamiento y características del etiquetado morfosintáctico

El etiquetado morfosintáctico, llamado *POS tagging (part-of-speech tagging)* en inglés, es el proceso para identificar las diferentes partes de la oración y consiste en asignar una etiqueta (*tag*) sobre la categoría gramatical a cada una de las palabras de un texto de entrada.

La entrada del algoritmo de etiquetado morfosintáctico es una secuencia de palabras y la salida del algoritmo es una secuencia de pares formados por la palabra y la correspondiente etiqueta indicando la categoría gramatical a la que pertenece dicha palabra. Ante esto:

- ▶ Existen diferentes conjuntos de etiquetas que se pueden utilizar en el análisis morfosintáctico.

- ▶ Algunos etiquetadores morfosintácticos permiten indicar a su entrada el conjunto de etiquetas que:
 - Identifican cada categoría gramatical.
 - Se van a utilizar en el proceso de etiquetado de cada parte de la oración.

Hoy en día, la mayoría de los algoritmos de procesamiento del lenguaje natural que procesan palabras en inglés utilizan el **Penn Treebank** (Marcus, Santorini y Marcinkiewicz, 1993).

El Penn Treebank es un conjunto de 45 etiquetas que identifican las diferentes partes de la oración, tal como se muestra en la siguiente imagen.

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	<i>and, but, or</i>	SYM	symbol	+%, &
CD	cardinal number	<i>one, two</i>	TO	"to"	<i>to</i>
DT	determiner	<i>a, the</i>	UH	interjection	<i>ah, oops</i>
EX	existential 'there'	<i>there</i>	VB	verb base form	<i>eat</i>
FW	foreign word	<i>mea culpa</i>	VBD	verb past tense	<i>ate</i>
IN	preposition/sub-conj	<i>of, in, by</i>	VBG	verb gerund	<i>eating</i>
JJ	adjective	<i>yellow</i>	VBN	verb past participle	<i>eaten</i>
JJR	adj., comparative	<i>bigger</i>	VBP	verb non-3sg pres	<i>eat</i>
JJS	adj., superlative	<i>wildest</i>	VBZ	verb 3sg pres	<i>eats</i>
LS	list item marker	<i>1, 2, One</i>	WDT	wh-determiner	<i>which, that</i>
MD	modal	<i>can, should</i>	WP	wh-pronoun	<i>what, who</i>
NN	noun, sing. or mass	<i>llama</i>	WPS	possessive wh-	<i>whose</i>
NNS	noun, plural	<i>llamas</i>	WRB	wh-adverb	<i>how, where</i>
NNP	proper noun, sing.	<i>IBM</i>	\$	dollar sign	\$
NNPS	proper noun, plural	<i>Carolinas</i>	#	pound sign	#
PDT	predeterminer	<i>all, both</i>	"	left quote	' or "
POS	possessive ending	<i>'s</i>	"	right quote	' or "
PRP	personal pronoun	<i>I, you, he</i>	(left parenthesis	[, (, {, <
PRP\$	possessive pronoun	<i>your, one's</i>)	right parenthesis],), }, >
RB	adverb	<i>quickly, never</i>	,	comma	,
RBR	adverb, comparative	<i>faster</i>	.	sentence-final punc	. ! ?
RBS	adverb, superlative	<i>fastest</i>	:	mid-sentence punc	:: ... --
RP	particle	<i>up, off</i>			

Figura 1. Etiquetas para las categorías gramaticales en el Penn Treebank. Fuente: Jurafsky y Martin, 2009.

En el etiquetado morfosintáctico se identifican las diferentes partes de la oración y se asigna una etiqueta a cada una de las palabras, tal y como se ha comentado anteriormente.

La forma más habitual para representar la salida del etiquetador morfosintáctico es colocar, después de cada palabra, la etiqueta para la categoría gramatical separada por una barra.

Por ejemplo, si el etiquetador morfosintáctico analiza la frase:

- ▶ Bebo un vaso del vino tinto.

Suponiendo que se utilizan las etiquetas para las categorías gramaticales definidas en el Penn Treebank, la salida sería:

bebo/VBP un/DT vaso/NN de/IN el/DT vino/NN tinto/JJ

Para la frase:

- ▶ Vino de un lugar lejano.

El etiquetado morfosintáctico sería:

vino/VBZ de/IN un/DT lugar/NN lejano/JJ

En estos dos ejemplos de etiquetado morfosintáctico se observa que la misma palabra «vino» se etiqueta de forma distinta para las dos frases.

En el primer ejemplo, la palabra «vino» pertenece a la categoría gramatical de los sustantivos o nombres (NN). Mientras que, en el segundo ejemplo, pertenece a la categoría gramatical de los verbos (VBZ).

Así, el etiquetado morfosintáctico realiza durante su funcionamiento un proceso de desambiguación: una palabra, que es ambigua y puede pertenecer a más de una categoría gramatical, se etiqueta correctamente según el contexto de la frase analizada.

Además, es importante notar que el algoritmo que realiza el etiquetado morfosintáctico ha separado la palabra «*del*» (que aparece en la frase «*bebo un vaso del vino tinto*») en las palabras *de* y *el* antes de etiquetarlas respectivamente como una preposición (IN) y un determinante (DT).

Identificar que una palabra es una contracción y separarla en las dos que la constituyen forma parte del proceso previo de preprocessado de la oración, que permite separar la frase en las diferentes palabras.

La identificación de las palabras de una oración también es llamada proceso de obtención de los tokens, porque token es el nombre inglés para definir una cadena de caracteres que representa una palabra y se realiza siempre previamente al etiquetado morfosintáctico y a otras tareas de procesamiento del lenguaje natural.

3.4. Etiquetado morfosintáctico basado en modelos ocultos de Markov (HMM)

Una de las técnicas más utilizadas en el etiquetado morfosintáctico es los modelos ocultos de Markov o HMM (por sus siglas del inglés, *Hidden Markov Model*). Esta técnica consiste en construir un modelo de lenguaje estadístico que se utiliza para obtener, a partir de una frase de entrada, la secuencia de etiquetas gramaticales que tiene mayor probabilidad.

Un modelo oculto de Markov es un modelo estadístico que se puede representar como una máquina de estados finitos, pero donde las transacciones entre estados son probabilísticas y no determinísticas. El objetivo es determinar los parámetros desconocidos (ocultos) a partir de los parámetros observables.

Por ejemplo, si hemos etiquetado una palabra como determinante, la próxima palabra será un nombre con un 40 % de probabilidad, un adjetivo con otro 40 % y un número el 20 % restante. Conociendo esta información, un sistema puede decidir que la palabra «vino» en la frase «el vino» es más probable que sea un nombre a que sea un verbo.

Para el etiquetado morfosintáctico, los HMM son entrenados en un conjunto de datos totalmente etiquetados. Esto es un conjunto de frases con cada palabra anotada con una etiqueta describiendo su categoría gramatical. A partir de los datos de entrenamiento, los HMM fijan estimaciones de máxima verosimilitud para cada uno de los estados y determina las diferentes probabilidades que rigen el modelo.

Para llevar a cabo el proceso de estimación de probabilidades se usa el algoritmo de decodificación de Viterbi (Forney, 1973).

El objetivo de decodificación HMM es elegir la secuencia de etiquetas más probable dada la secuencia de observación n palabras w_1^n :

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n)$$

Esta ecuación la podemos reescribir mediante el uso de la regla de Bayes de la siguiente forma:

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)}$$

También podemos simplificar esta ecuación eliminando el denominador $P(w_1^n)$:

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(w_1^n | t_1^n) P(t_1^n)$$

Los etiquetadores HMM hacen dos suposiciones que permiten simplificar estas ecuaciones aún más:

1. La primera es que la probabilidad de aparición de una palabra depende solo de su propia etiqueta y es independiente de las palabras y etiquetas vecinas:

$$P(w_1^n | t_1^n) \approx \prod_{i=1}^n P(w_i^n | t_i^n)$$

2. La segunda suposición, también llamada bigrama o digrama, es que la probabilidad de una etiqueta solo depende de la etiqueta anterior, en lugar de toda la secuencia de etiquetas:

$$P(t_i^n) \approx \prod_{i=1}^n P(t_i | t_{i-1})$$

Aplicando estas suposiciones a las ecuaciones anteriores terminamos con la siguiente ecuación para la secuencia de etiquetas más probable de un etiquetador bigrama, las cuales corresponden a la probabilidad de emisión y la probabilidad de transición de un HMM:

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n) \approx \operatorname{argmax}_{t_1^n} \prod_{i=1}^n \overbrace{P(w_i^n | t_i^n)}^{\text{emission}} \overbrace{P(t_i | t_{i-1})}^{\text{transition}}$$

Ejemplos de probabilidad de transición y probabilidad de emisión

Veamos a través de un ejemplo cómo se calculan y se utilizan en una tarea de etiquetado estas probabilidades. En el etiquetado HMM, las probabilidades se estiman simplemente a partir de un corpus de entrenamiento etiquetado; para este ejemplo vamos a utilizar el **corpus etiquetado WSJ**, una colección de un millón de

palabras que se publicaron en los artículos del Wall Street Journal (WSJ) en 1989 y que están anotadas utilizando las etiquetas morfosintácticas del Penn Treebank.

Un corpus lingüístico es una colección de textos representativos de una lengua que se utilizan para el análisis lingüístico. Los corpus pueden estar anotados o etiquetados de forma que las palabras que lo conforman presentan, además, algún tipo de información lingüística.

Accede al corpus a través del aula virtual o desde la siguiente dirección web:

<https://catalog.ldc.upenn.edu/LDC2000T43>

Probabilidades de transición

Las probabilidades de transición de etiqueta $P(t_i|t_{i-1})$ representan la probabilidad de una etiqueta dada la etiqueta anterior. Por ejemplo, los verbos modales (etiqueta MD) como «can» (poder) son muy probablemente seguidos por un verbo en la forma base (etiqueta VB) como «run» (correr), por lo que espera que esta probabilidad sea alta.

La estimación de máxima verosimilitud de una probabilidad de transición se calcula por recuento de las veces que vemos la primera etiqueta en un corpus etiquetado y la frecuencia con que esta primera etiqueta es seguida por la segunda según la siguiente fórmula:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

En el corpus WSJ, por ejemplo, los MD aparecen 13 124 veces, de las cuales son seguidos por un VB 10 471 veces, lo cual resulta en una estimación de máxima probabilidad de:

$$P(VB|MD) = \frac{C(MD, VB)}{C(MD)} = \frac{10471}{13124} = .80$$

Probabilidades de emisión

Las probabilidades de emisión $P(w_1^n | t_1^n)$ representan la probabilidad de que, dada una etiqueta (digamos MD), esta se asocie con una palabra concreta (digamos «will»).

La estimación de **máxima verosimilitud** de la probabilidad de emisión en general se define como:

$$P(w_i | t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

De los 13 124 casos de MD en el corpus WSJ, estas se asocian o refieren a «will» en 4046 ocasiones, por tanto:

$$P(will|MD) = \frac{C(MD, will)}{C(MD)} = \frac{4046}{13124} = .31$$

Como aclaración, hemos de decir que esta probabilidad no se refiere a cuál es la etiqueta más probable para la palabra «will», ya que esta sería la probabilidad *a posteriori* $P(MD|will)$. En su lugar, $P(will|MD)$ responde a una pregunta ligeramente menos intuitiva, concretamente si vamos a generar una etiqueta MD, ¿qué probabilidades hay de que este MD sea «will»?

Los dos tipos de probabilidades mencionados anteriormente, la probabilidad de transición $P(VB|MD)$ y la probabilidad de emisión $P(will|MD)$, se corresponden al conjunto A de probabilidades de transición del HMM y al conjunto B de probabilidades de observación B del HMM.

La Figura 2 ilustra algunas de las probabilidades de transición A para tres estados en un etiquetador morfosintáctico HMM; el etiquetador completo tendría un estado

para cada etiqueta. En esta imagen, las probabilidades de transición A se utilizan para calcular la probabilidad *a priori*.

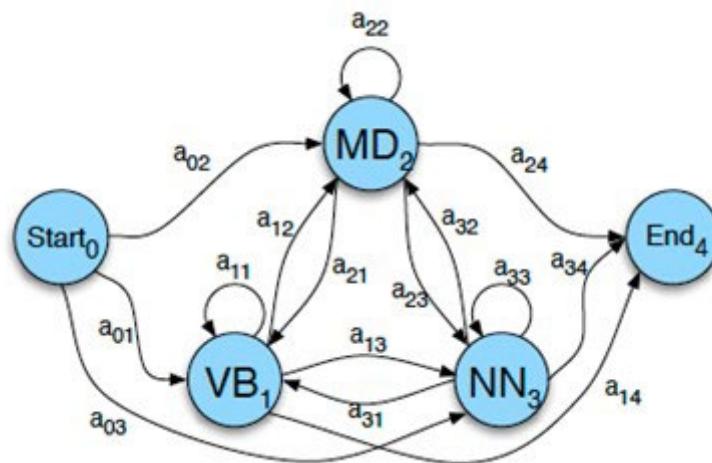


Figura 2. Cadena de Markov correspondiente a los estados ocultos del HMM. Fuente: Jurafsky y Martin, 2009.

La Figura 3 muestra otra vista de estos tres estados, pero centrándose en algunas de las probabilidades de observación B de cada palabra. Cada estado oculto está asociado con un vector de probabilidades para cada palabra en observación.

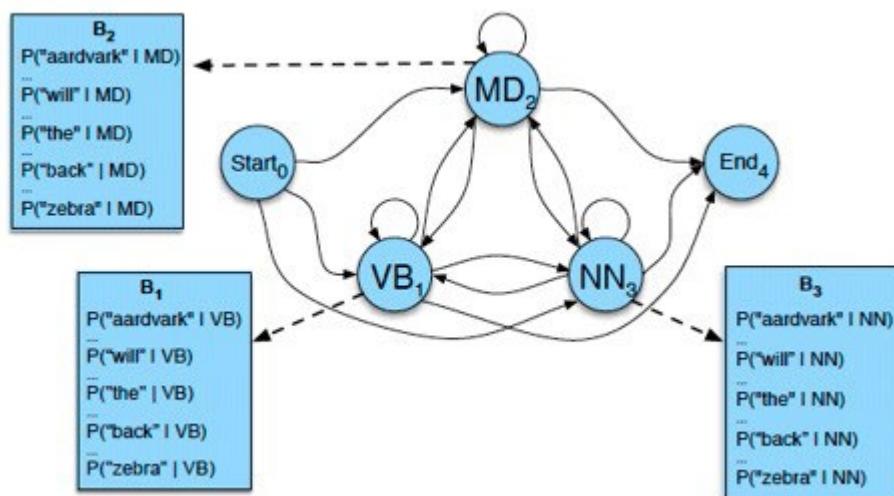


Figura 3. Probabilidades de observación B para el HMM de la Figura 2. Fuente: Jurafsky y Maritn, 2009.

Como indicábamos sobre esta última imagen, cada estado está asociado con un vector de probabilidades con una probabilidad para cada posible palabra en observación, a excepción de los estados no emisores de inicio y fin.

En el vídeo *Construcción de un etiquetador morfosintáctico basado en un HMM bigrama a partir de un corpus etiquetado* se explicará cómo crear un etiquetador morfosintáctico a partir de un corpus etiquetado.



Accede al vídeo

Finalmente, vamos a trabajar a través de un ejemplo de cálculo de la mejor secuencia de etiquetas que corresponde a la siguiente secuencia de palabras:

- ▶ Janet will back the Bill (en español, «Janet respaldará la ley»).

La secuencia de etiquetas correcta es:

Janet/NNP will/MD back/VB the/DT bill/NN

Sea el modelo HMM el definido por el conjunto A de probabilidades de transición (Figura 4), y el conjunto B de probabilidades de observación (Figura 5). Cada elemento a_{ij} del conjunto A describe la probabilidad de transitar de un estado oculto i (etiqueta i) a otro estado oculto j (etiqueta j). Cada elemento $b_i(o_t)$ describe la probabilidad de observar las palabras dadas las etiquetas.

	PNN	MD	VB	JJ	NN	RB	DT
<S>	0.2767	0,0006	0,0031	0,0453	0,0449	0,0510	0.2026
PNN	0.3777	0,0110	0,0009	0,0084	0,0584	0,0090	0,0025
MD	0,0008	0,0002	0.7968	0,0005	0,0008	0,1698	0,0041
VB	0,0322	0,0005	0,0050	0,0837	0,0615	0,0514	0.2231
JJ	0,0366	0,0004	0,0001	0,0733	0,4509	0,0036	0,0036
NN	0,0096	0,0176	0,0014	0,0086	0,1216	0,0177	0,0068
RB	0,0068	0,0102	0,1011	0,1012	0,0120	0,0728	0,0479
DT	0,1147	0,0021	0,0002	0,2157	0,4744	0,0102	0,0017

Figura 4. Conjunto A de probabilidades de transición $P(t_i|t_{i-1})$ calculadas a partir del corpus WSJ. Fuente: Jurafsky y Martin, 2009.

En la imagen anterior vemos que cada fila representa el evento condicionante; por ejemplo:

$$P(VB|MD) = 0.7968.$$

La siguiente imagen (Figura 5) se obtiene a partir del recuento de apariciones de una palabra en el corpus. Así:

- ▶ La palabra «Janet» solo aparece como un nombre propio (NNP).
- ▶ La palabra «will» aparece en el corpus como tres categorías gramaticales diferentes, puede ser:
 - Un verbo modal (MD) para generar el futuro de los verbos en inglés.
 - Un verbo (VB) que significa «desear» en español.
 - O un nombre (NN) que significa «deseo» o «voluntad» en español.

	Janet	will	back	the	bill
PNN	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0.000097	0
NN	0	0.000200	0.000223	0.000006	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Figura 5. Conjunto B de probabilidades de observación calculadas a partir del corpus WSJ. Fuente: Jurafsky y Martin, 2009.

La Figura 6 muestra un esquema con las posibles etiquetas para el ejemplo anterior, así como la correcta secuencia de etiquetado final. Esta secuencia del etiquetado morfosintáctico se ha calculado aplicando el **algoritmo de Viterbi**, cuyo pseudocódigo se presenta en la Figura 7.

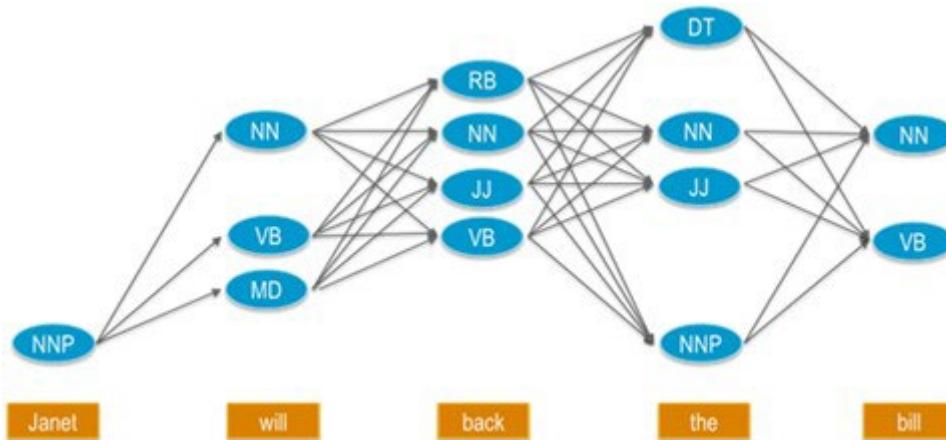


Figura 6. Diagrama de la tarea de etiquetado para la frase ejemplo. Fuente: Jurafsky y Martin, 2009.

```

function VITERBI(observations of len  $T$ ,state-graph of len  $N$ ) returns best-path
    create a path probability matrix viterbi[ $N+2, T$ ]
    for each state  $s$  from 1 to  $N$  do ; initialization step
        viterbi[ $s, 1$ ]  $\leftarrow a_{0,s} * b_s(o_1)$ 
        backpointer[ $s, 1$ ]  $\leftarrow 0$ 
    for each time step  $t$  from 2 to  $T$  do ; recursion step
        for each state  $s$  from 1 to  $N$  do
             $viterbi[s,t] \leftarrow \max_{s'=1}^N viterbi[s',t-1] * a_{s',s} * b_s(o_t)$ 
            backpointer[ $s,t$ ]  $\leftarrow \operatorname{argmax}_{s'=1}^N viterbi[s',t-1] * a_{s',s}$ 
    viterbi[ $q_F, T$ ]  $\leftarrow \max_{s=1}^N viterbi[s,T] * a_{s,q_F}$  ; termination step
    backpointer[ $q_F, T$ ]  $\leftarrow \operatorname{argmax}_{s=1}^N viterbi[s,T] * a_{s,q_F}$  ; termination step
    return the backtrace path by following backpointers to states back in time from backpointer[ $q_F, T$ ]

```

Figura 7. Pseudocódigo del algoritmo de Viterbi para encontrar la secuencia óptima de *tags* en un etiquetador morfosintáctico basado en HMM. Fuente: Jurafsky y Martin, 2009.

El algoritmo de Viterbi crea una matriz de probabilidades con una columna para cada observación t y una fila para cada estado q_i de la máquina de estados finitos (o autómata finito) que representa el HMM. Para el ejemplo, el algoritmo crea $N = 5$

columnas de estado, la primera para la observación de la primera palabra «Janet», la segunda para «will» y así sucesivamente hasta completar las cinco palabras que conforman la frase, tal como se muestra en la Figura 8.

Se empieza en la primera columna para establecer el valor de Viterbi en cada celda $v_t(i)$, que se calcula como el producto de la probabilidad de transición (hasta ese estado desde el estado inicial) por la probabilidad de observación (de la primera palabra). Entonces, se avanza columna a columna y, para cada estado en la columna 1, se calcula la probabilidad de transición a cada estado en la columna 2, y así sucesivamente.

Para cada estado q_i en el tiempo t , se calcula valor de Viterbi (representado como `viterbi[s, t]` en el pseudocódigo de la Figura 7) tomando el máximo sobre las extensiones de todas las rutas que conducen a la celda actual, según la siguiente ecuación:

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t)$$

Donde:

- ▶ $v_{t-1}(i)$ es la probabilidad de la ruta de Viterbi previa, es decir, la ruta para el tiempo anterior o en la observación $t - 1$.
- ▶ a_{ij} es la probabilidad de transición del estado anterior q_i al estado actual q_j .
- ▶ $b_j(o_t)$ es la probabilidad de observación del símbolo o_t dado el estado actual q_j .

Entonces, cada celda de la primera columna donde aparece la palabra «Janet» se calcula multiplicando la probabilidad de Viterbi previa en el estado de inicio q_0 , que es $v_0(0) = 1.0$ por la probabilidad de transición desde el estado de inicio q_0 hasta la etiqueta para esa celda, por ejemplo:

$$P(NNP|start) = 0.2767 \text{ para la celda en la que la etiqueta es NNP.}$$

Y por la probabilidad de observación de la palabra «Janet» dada la etiqueta de esa celda, por ejemplo:

$$P(\text{Janet}|\text{NNP}) = 0.000032 \text{ para la celda en la que la etiqueta es NNP.}$$

Por lo tanto:

- ▶ $v_1(1) = 1.0 \cdot 0.2767 \cdot 0.000032 = 0.000009$ para la celda en la que la etiqueta es NNP para la columna donde la palabra es «Janet».
- ▶ El resto de las celdas en esta columna son cero, ya que la palabra «Janet» no puede tener asociada ninguna de las otras etiquetas gramaticales.

A continuación, cada celda en la columna «will» se actualiza con la ruta de probabilidad máxima desde la columna anterior. En la Figura 8 se muestran los valores para las celdas MD, VB y NN. Cada celda obtiene los siete valores de la columna anterior multiplicados por la probabilidad de transición apropiada; se toma el valor máximo, $v_1(1) \cdot P(\text{MD}|\text{NNP})$, que proviene del estado NNP en la columna anterior y este valor se multiplica por la probabilidad de observación del símbolo «will» dada la etiqueta correspondiente a la celda en cuestión. Por ejemplo, para la celda MD el valor de Viterbi es:

$$v_2(2) = v_1(1) \cdot P(\text{MD}|\text{NNP}) \cdot P(\text{will}|\text{MD}) = 0.00000002772$$

Se continúa aplicando el algoritmo columna a columna hasta llegar al final.

En el vídeo *Creación de la matriz de probabilidades de la ruta de Viterbi* se verá cómo realizar la matriz de Viterbi para llevar a cabo el etiquetado morfosintáctico de una oración.



Accede al vídeo

Para la probabilidad máxima de Viterbi, se traza la inversa para obtener la ruta concreta que ha llevado a ese valor y que se corresponde con la mejor secuencia de etiquetas: NNP MD VB DT NN.

Entonces, esta secuencia de etiquetas se corresponde con el etiquetado morfosintáctico de la frase:

Janet/NNP will/MD back/VB the/DT bill/NN

En el vídeo *Obtención de la ruta de Viterbi con máxima probabilidad* se verá cómo obtener la ruta de Viterbi, lo que significa que cuando apliquemos el logaritmo seamos capaces de decodificar y obtener la ruta más probable que nos dé el mejor etiquetado morfosintáctico de una oración.



Accede al vídeo

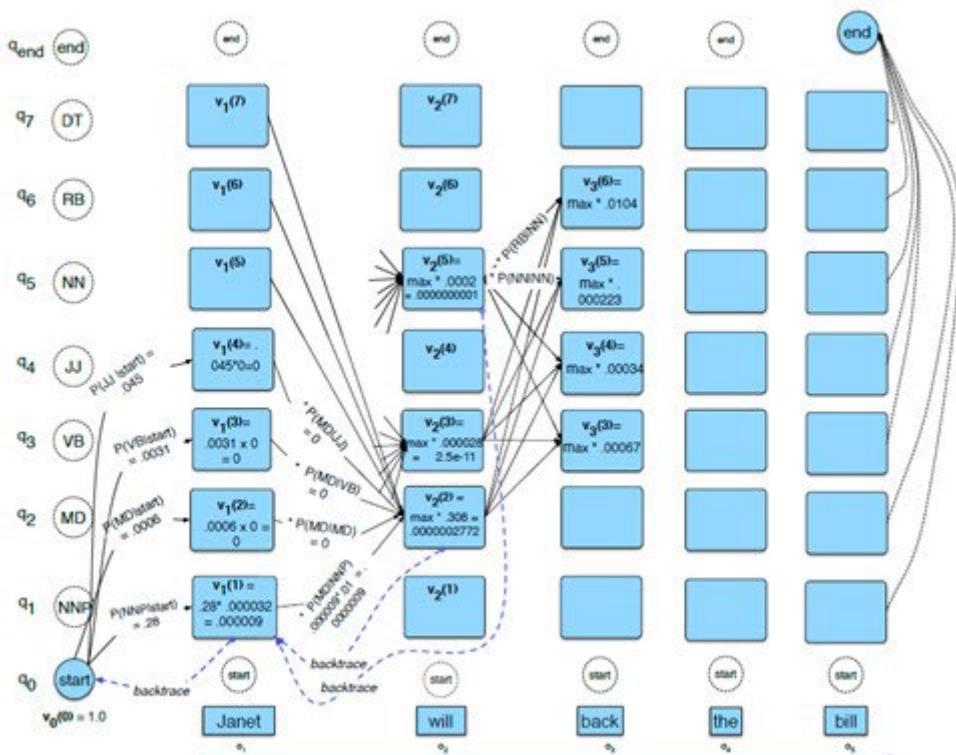


Figura 8. Matriz para la aplicación del algoritmo de Viterbi en el etiquetado morfosintáctico de la frase «Janet will back the bill». Fuente: Jurafsky y Martin, 2009

3.5. Etiquetado morfosintáctico basado en aprendizaje automático

Los modelos más vanguardistas de etiquetado morfosintáctico utilizan diversas técnicas de aprendizaje automático tanto supervisado como no supervisado. Ejemplos de modelos supervisados son, entre otros:

- ▶ El algoritmo perceptrón (Collins, 2002).
- ▶ El modelo logaritmo lineal bidireccional (Toutanova, Klein, Manning y Singer, 2003).
- ▶ Las máquinas de vectores soporte (Giménez y Márquez, 2004).

En estos casos de aprendizaje supervisado, el modelo recibe como entrada una serie de parámetros extraídos del texto, y tiene como salida la predicción del POS tag de una palabra en concreto. A modo de ejemplo, en (Giménez y Márquez, 2004) se propone el uso de las siguientes variables de entrada:

word features	$w_{-3}, w_{-2}, w_{-1}, w_0, w_1, w_2, w_3$
POS features	$p_{-3}, p_{-2}, p_{-1}, p_0, p_1, p_2, p_3$
ambiguity classes	a_0, a_1, a_2, a_3
may_be's	m_0, m_1, m_2, m_3
word bigrams	$(w_{-2}, w_{-1}), (w_{-1}, w_0), (w_{-1}, w_1)$ $(w_0, w_1), (w_1, w_2)$
POS bigrams	$(p_{-2}, p_{-1}), (p_{-1}, a_0), (a_0, a_1)$
word trigrams	$(w_{-2}, w_{-1}, w_0), (w_{-2}, w_{-1}, w_1),$ $(w_{-1}, w_0, w_1), (w_{-1}, w_1, w_2),$ (w_0, w_1, w_2)
POS trigrams	$(p_{-2}, p_{-1}, a_0), (p_{-2}, p_{-1}, a_1),$ $(p_{-1}, a_0, a_1), (p_{-1}, a_1, a_2)$
sentence_info	punctuation ('.', '?', '!')
prefixes	$s_1, s_1s_2, s_1s_2s_3, s_1s_2s_3s_4$
suffixes	$s_n, s_{n-1}s_n, s_{n-2}s_{n-1}s_n, s_{n-3}s_{n-2}s_{n-1}s_n$
binary word features	initial Upper Case, all Upper Case, no initial Capital Letter(s), all Lower Case, contains a (period / number / hyphen ...)
word length	integer

Figura 9. Variables de entrada para predecir los POS tags con un modelo de aprendizaje supervisado. Fuente: Giménez y Márquez, 2004.

Como se puede ver, se generan variables de entrada que hacen referencia tanto a la palabra en sí sobre la que se quiere predecir el POS tag, como a palabras de su contexto, tanto previas o posteriores. Se incluyen también variables que hacen referencia a los posibles POS tags que tiene la palabra en cuestión y las palabras previas o posteriores.

No obstante, tanto estos como los algoritmos presentados en el tema dependen en gran medida del dominio de datos de entrenamiento, así como el etiquetado marcado por los expertos.

Algunos trabajos recientes en etiquetado morfosintáctico se centran en buscar alternativas que permitan relajar estas condiciones, es el caso de los algoritmos no supervisados que etiquetan clústers de palabras en clases morfosintácticas (Christodoulopoulos, Goldwater, y Steedman, 2010) (Sirts, Eisenstein, Elsner, y Goldwater, 2014).

Muchos algoritmos se basan en la combinación de datos etiquetados con datos no etiquetados, por ejemplo, usando **coentrenamiento** (Søgaard, 2010). La asignación de etiquetas a texto de muy distintos tipos como, por ejemplo, aquellos provenientes de *Twitter*, puede requerir la adición de nuevas etiquetas para las direcciones URL (URL), nombre de usuario menciona (USR), *retweets* (RT), y *hashtags* (HT). La normalización de las palabras no estándar y técnicas *bootstrapping* para emplear datos no supervisados (Derczynski et al., 2013).

En estos ejemplos se lleva a cabo un **etiquetado por clasificación**, donde el objetivo es, dada una secuencia, predecir una a una las etiquetas de sus palabras, considerando información de la propia palabra y del contexto. Esto se refleja en la siguiente ecuación, donde $f()$ es la función que relaciona las variables de entrada para una determinada palabra con la salida predicha con la estructura (*token, POS tag*):

$$f(w = \text{they can fish}, m = 2) = (\text{can}, V)$$

Finalmente, además de estos modelos de aprendizaje automático, existen otros algoritmos estadísticos muy populares para tareas de POS tagging, como es el caso del algoritmo Conditional Random Field (CRF) (Lafferty, 2001). También existen métodos basados en la generación de reglas, como es el caso del algoritmo de Brill para POS tagging (Brill, 1992).

3.6. *Named Entity Recognition*

Además de poder obtener el POS Tag de una palabra mediante el uso de algoritmos estadísticos o técnicas de aprendizaje automático, también es posible identificar de entre los distintos sustantivos a qué tipo de entidad nombrada hace referencia (a una persona, a una organización o a una ubicación, por ejemplo).

Esta tarea se denomina **NER (named-entity recognition)**, y con ella se suelen identificar dentro de un texto categorías como las siguientes (a modo de ejemplo, ya que se pueden usar más categorías):

- ▶ **PER**: categoría de personas, como por ejemplo el nombre de alguien (ej.: Frodo Baggins)
- ▶ **GPE**: categoría para identificar países, ciudades o estados (ej.: Madrid).
- ▶ **LOC**: categoría para identificar ubicaciones concretas (ej.: Vesubio).
- ▶ **ORG**: categoría para identificar organizaciones o empresas (ej.: Microsoft).
- ▶ **MONEY**: categoría para identificar referencias a dinero (ej.: \$ 5).
- ▶ **DATE**: categoría para identificar fechas (ej.: 05/02/2021 o «jueves»).

De esta manera, una frase en la que se ha llevado a cabo un NER daría como resultado:

[PER Miguel López], de [ORG Microsoft], ha ido a una conferencia a [GPE Londres].

Existen distintas maneras de representar las NE (*named entity*) dentro de una frase, donde aparecen representaciones para, incluso, diferenciar qué token representa el comienzo de una NE, cuáles están en medio y cuáles están al final para casos en los que varios tokens estén asociados a una misma NE.

Words	IO Label	BIO Label	BIOES Label
Jane	I-PER	B-PER	B-PER
Villanueva	I-PER	I-PER	E-PER
of	O	O	O
United	I-ORG	B-ORG	B-ORG
Airlines	I-ORG	I-ORG	I-ORG
Holding	I-ORG	I-ORG	E-ORG
discussed	O	O	O
the	O	O	O
Chicago	I-LOC	B-LOC	S-LOC
route	O	O	O
.	O	O	O

Figura 10. Distintas representaciones de las NE. Fuente: Jurafsky y Martin, 2009.

De igual manera que ocurre con los algoritmos de POS tagging, la identificación de las NER se puede realizar:

- ▶ Con **diccionarios** donde se tengan ya identificadas palabras (o combinaciones de palabras) junto con su NE.
- ▶ Con **sistemas basados en reglas** que identifiquen las NE en base a ciertos patrones.
- ▶ Con **modelos estadísticos o de aprendizaje automático**, que en base a un entrenamiento previo y a una serie de variables que modelen el contexto de las palabras, puedan predecir qué token o tokens hacen referencia a una NER.

3.7. Referencias bibliográficas

Brill, E. (1992). *A simple rule-based part of speech tagger*. PENNSYLVANIA UNIV PHILADELPHIA DEPT OF COMPUTER AND INFORMATION SCIENCE.

Christodoulopoulos, C., Goldwater, S. y Steedman, M. (2010). Two decades of unsupervised POS induction: How far have we come? En *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 575-584). Association for Computational Linguistics.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. En *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1-8). Association for Computational Linguistics.

Derczynski, L., Ritter, A., Clark, S. y Bontcheva, K. (2013). Twitter part-of-speech tagging for all: Overcoming sparse and noisy data. En *Proceedings of Recent Advances in Natural Language Processing* (pp. 198-206). Association for Computational Linguistics.

Eisenstein, J. (2019). Introduction to Natural Language Processing (pp. 145-148). MIT Press Ltd.

Forney, G. D. (1973). The viterbi algorithm. *Proceedings of the IEEE*, 61(3), 268-278.

Giménez, J. y Márquez, L. (2004). SVMTool: A general POS tagger generator based on Support Vector Machines. En *Proceedings of the 4th International Conference on Language Resources and Evaluation*. LREC.

Jurafsky, D. y Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall.

Marcus, M. P., Santorini, B. y Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2), 313-330.

Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. En *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289

RAE. (s. f.). Morfosintaxis. En *Diccionario de la lengua española* (actualización de la 23^a ed.). <http://dle.rae.es/?id=PpC0akB>

Sirts, K., Eisenstein, J., Elsner, M., y Goldwater, S. (2014). POS induction with distributional and morphological information using a distance-dependent Chinese restaurant process. En *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* (pp. 265-271). Baltimore, Estados Unidos. <http://www.aclweb.org/anthology/P14-2044>

Søgaard, A. (2010). Simple semi-supervised training of part-of-speech taggers. En *Proceedings of the ACL 2010 Conference Short Papers* (pp. 205-208). Uppsala, Suecia: Association for Computational Linguistics.

Toutanova, K., Klein, D., Manning, C. D. y Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. En *Proceedings of HLT-NAACL* (pp. 173-180). Association for Computational Linguistics.

A fondo

Etiquetado morfosintáctico basado en aprendizaje no supervisado

Christodoulopoulos, C., Goldwater, S. y Steedman, M. (2010). Two decades of unsupervised POS induction: How far have we come? En *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 575-584). Association for Computational Linguistics.

<https://homepages.inf.ed.ac.uk/sgwater/papers/emnlp10-20yrsPOS.pdf>

El artículo presenta la evaluación de siete algoritmos que realizan el etiquetado morfosintáctico basándose en técnicas de aprendizaje no supervisado. Sorprendentemente, se muestra que los primeros algoritmos que aparecieron hace ya más de veinticinco años proporcionan mejores resultados que otros más recientes.

Analizador morfológico automático de la Biblioteca Virtual Miguel de Cervantes

Analizador morfológico automático. <http://data.cervantesvirtual.com/analizador-sintactico-automatico>

Esta aplicación de la Biblioteca Virtual Miguel de Cervantes permite introducir un texto en español y realizar el análisis morfosintáctico de forma automática, por lo que se logra identificar la categoría gramatical de cada palabra.

Test

1. Indica las afirmaciones correctas sobre la morfosintaxis:

 - A. Determina las diferentes partes de la oración.
 - B. Estudia la estructura de las palabras.
 - C. Parte de la gramática que aúna la morfología y la sintaxis.
 - D. Estudia el modo en que se combinan las palabras.
2. La clasificación de las diferentes palabras según su tipo o clase se conocen como:

 - A. Categorías morfosintácticas.
 - B. Categorías gramaticales.
 - C. Partes de la oración.
 - D. *Part-of-speech* (POS) en inglés.
3. Indica las afirmaciones correctas sobre el etiquetado morfosintáctico:

 - A. Su salida es la secuencia de etiquetas de las categorías gramaticales.
 - B. Se llama POS *tagging* (*part-of-speech tagging* en inglés).
 - C. Asigna etiquetas sobre la categoría gramatical a cada una de las palabras de la oración.
 - D. Su entrada solo es una secuencia de palabras.
4. Indica las afirmaciones correctas sobre el Penn Treebank:

 - A. Se utiliza en el etiquetado morfosintáctico en inglés.
 - B. Consiste en un conjunto de etiquetas gramaticales.
 - C. Etiqueta los signos de puntuación de la frase.
 - D. Etiqueta todos los verbos con una misma etiqueta.

5. Indica las afirmaciones correctas sobre el algoritmo implementado por un etiquetador morfosintáctico:
- A. No se ve afectado por problemas de ambigüedad gramatical de las palabras.
 - B. Aplica un proceso de desambiguación.
 - C. Etiqueta directamente la oración sin realizar ningún procesado previo.
 - D. Aplica un proceso de obtención de tokens.
6. Indica las afirmaciones correctas sobre un modelo oculto de Markov (HMM):
- A. Es una máquina de estados finitos.
 - B. Es un modelo estadístico.
 - C. Su objetivo es determinar parámetros desconocidos a partir de parámetros observables.
 - D. Es un autómata finito donde las transacciones entre estados son probabilísticas.
7. Indica las afirmaciones correctas sobre un etiquetador morfosintáctico basado en HMM:
- A. Modelo de lenguaje estadístico que permite obtener la secuencia de etiquetas gramaticales que tenga mayor probabilidad para una frase.
 - B. Se entrena con un conjunto de frases en la que cada palabra está anotada con una etiqueta describiendo su categoría gramatical.
 - C. Se fijan las estimaciones de máxima probabilidad para cada una de las condiciones de la máquina de estados finitos a partir de los datos de entrenamiento.
 - D. Se usa el algoritmo de decodificación de Viterbi para estimar las probabilidades.

- 8.** Indica las afirmaciones correctas sobre la probabilidad de transición del HMM utilizado en el etiquetado morfosintáctico:
- A. Representa la probabilidad de la etiqueta anterior dada una etiqueta.
 - B. Representa la probabilidad de una etiqueta dada la etiqueta anterior.
 - C. La estimación de máxima verosimilitud de una probabilidad de transición se calcula como la división entre el recuento de las veces que vemos la primera etiqueta en un corpus etiquetado entre la frecuencia con que la primera etiqueta es seguida por la segunda.
 - D. La estimación de máxima verosimilitud de una probabilidad de transición se calcula como la división entre el recuento de las veces que vemos la primera etiqueta seguida por la segunda en un corpus etiquetado entre la frecuencia con que aparece la primera etiqueta.
- 9.** Indica las afirmaciones correctas sobre la probabilidad de emisión del HMM utilizado en el etiquetado morfosintáctico:
- A. Permite identificar la palabra más probable para una etiqueta dada.
 - B. Representa la probabilidad de que, dada una palabra, esta se asocie a una etiqueta concreta.
 - C. Representa la probabilidad de que, dada una etiqueta, esta esté asociada a una palabra concreta.
 - D. Permite identificar la etiqueta más probable para una palabra dada.

- 10.** Indica las afirmaciones correctas sobre la aplicación del algoritmo de Viterbi para obtener la secuencia de etiquetas más probables en un etiquetador morfosintáctico HMM:
- A. Cada columna en la matriz de probabilidades corresponde a una palabra en la frase a analizar y se llama observación.
 - B. Cada celda en una columna de la matriz de probabilidades corresponde a un estado de la máquina de estados finitos y se corresponde a una etiqueta morfosintáctica.
 - C. La probabilidad de Viterbi de una ruta se calcula como la multiplicación de probabilidad de la ruta de Viterbi previa por la probabilidad de transición del estado anterior al estado actual por la probabilidad de observación del símbolo (etiqueta morfosintáctica), dado el estado actual.
 - D. La secuencia de etiquetas morfosintáctica correctas se obtiene de trazar la ruta inversa que ha llevado a obtener el valor de Viterbi máximo en el estado final.

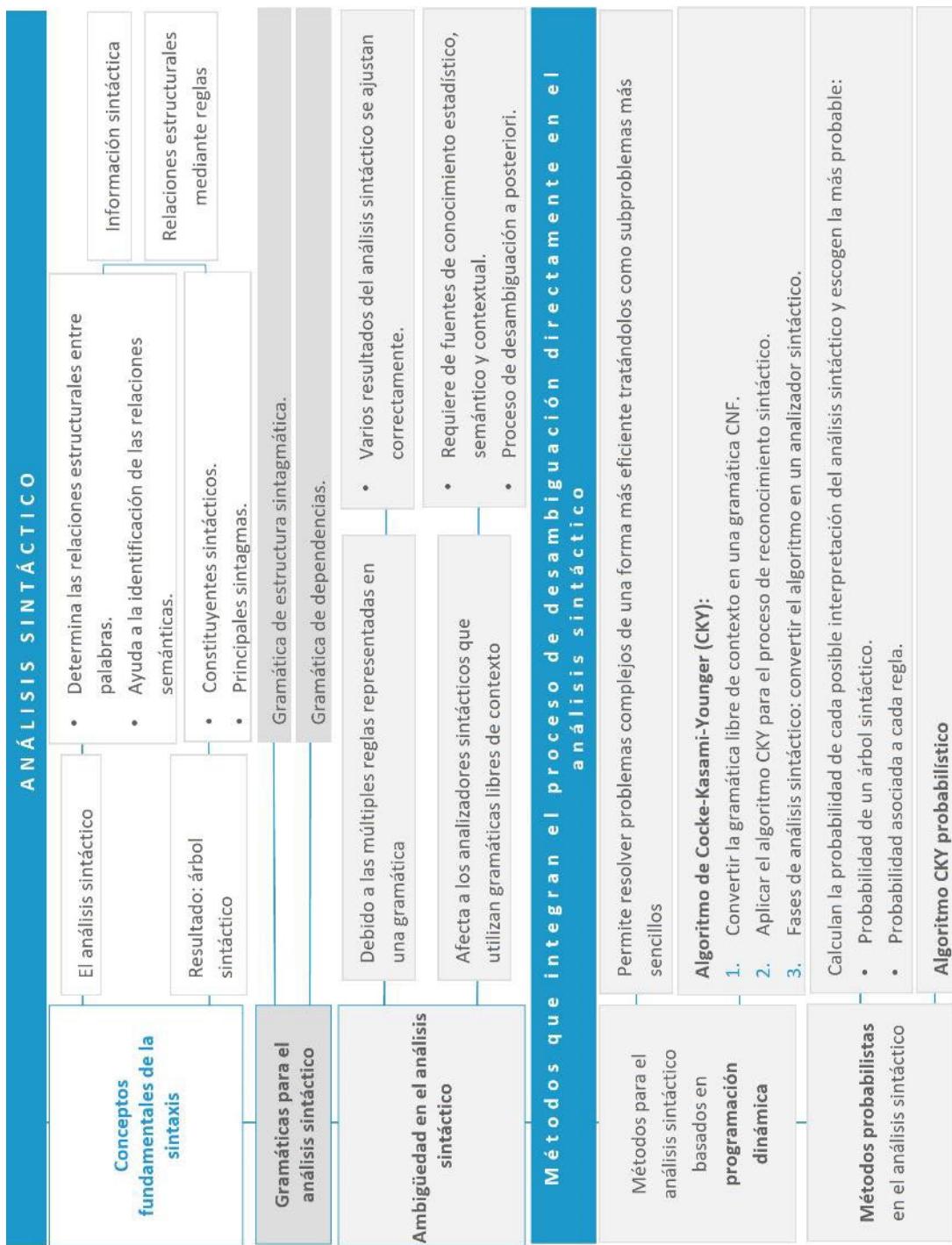
Procesamiento del Lenguaje Natural

Análisis sintáctico

Índice

Esquema	3
Ideas clave	4
4.1. Introducción y objetivos	4
4.2. Sintaxis	5
4.3. Gramáticas de estructura sintagmática	8
4.4. Ambigüedad en el análisis sintáctico	9
4.5. Métodos para el análisis sintáctico, basados en la programación dinámica	11
4.6. Métodos probabilistas en el análisis sintáctico	20
4.7. Gramáticas de dependencias o gramáticas valenciales	25
4.8. Referencias bibliográficas	31
A fondo	33
Test	36

Esquema



4.1. Introducción y objetivos

Se profundizará en el análisis sintáctico y las relaciones estructurales entre las palabras. Esta información sintáctica se modela mediante las **gramáticas sintagmáticas**, de las cuales veremos las **gramáticas libres de contexto**. Tras esto, analizaremos el **concepto de ambigüedad estructural**, uno de los mayores problemas que se dan en el análisis sintáctico, y cuáles son las técnicas para solucionarla como son los métodos basados en programación dinámica y los métodos probabilistas.

Objetivos

- ▶ Entender cómo el problema de la ambigüedad en la estructura de las frases afecta al funcionamiento de los analizadores sintácticos basados en búsqueda, ya sea ascendente o descendente.
- ▶ Describir el funcionamiento de algunas técnicas de programación dinámica que se utilizan en el análisis sintáctico porque son capaces de representar ambigüedades.
- ▶ Describir el concepto de gramática libre de contexto probabilística y modelar un problema de ambigüedad utilizando este tipo de gramática.
- ▶ Aplicar métodos basados en gramáticas libres de contexto probabilísticas para solventar problemas de ambigüedad.
- ▶ Definir el concepto de gramática de dependencias o gramática valencial e identificar sus diferencias respecto a las gramáticas de estructura sintagmática.

4.2. Sintaxis

La Real Academia Española (RAE, s. f.) en su diccionario de la lengua española define la palabra **sintaxis**:

«Del lat. tardío *syntaxis*, y este del gr. σύνταξις *sýntaxis*, de συντάσσειν “disponer conjuntamente”, “ordenar”. 1. f. Gram. Parte de la gramática que **estudia el modo en que se combinan las palabras y los grupos que estas forman para expresar significados**, así como las relaciones que se establecen entre todas esas unidades».

El análisis sintáctico

En el análisis sintáctico se **determinan las relaciones estructurales entre palabras** y es muy relevante en el procesamiento del lenguaje natural no solo por la información sintáctica que aporta, también porque es un paso esencial para la posterior identificación de las relaciones semánticas de las oraciones.

Árbol sintáctico

Es el resultado del análisis sintáctico. Sus nodos son los constituyentes sintácticos y las hojas, las palabras que componen la oración analizada. La estructura jerárquica del árbol permite observar que un constituyente está formado por una o varias palabras y por otros constituyentes.

Un constituyente sintáctico es una palabra o una secuencia de palabras que realizan una función conjunta dentro de la estructura jerárquica de la oración.

En la gramática tradicional se llama **sintagma al constituyente sintáctico compuesto de dos a más elementos** y según la *Nueva gramática de la lengua española* (Real Academia Española, 2009) se denomina **grupo sintáctico**.

Los principales tipos de sintagma de la lengua española se presentan en la siguiente tabla:

Sintagma	Núcleo	Función
Sintagma Nominal (SN)	Sustantivo Pronombre	Sujeto. Atributo. Complemento directo. Complemento indirecto (solo si el núcleo es un pronombre). Complemento predicativo. Complemento circunstancial.
Sintagma Verbal (SV)	Verbo	Predicado (predicado verbal o predicado nominal).
Sintagma Preposicional (SP o SPrep)	Preposición	Complemento de régimen. Complemento directo. Complemento preposicional. Complemento circunstancial.
Sintagma Adjetival (SAdj)	Adjetivo	Complemento adyacente. Complemento predicativo.
Sintagma Adverbial (SAdv)	Adverbio	Adjunto. Complemento circunstancial.

Tabla 1. Principales sintagmas de la lengua española. Fuente: elaboración propia.

En el vídeo *Sintaxis y constituyentes sintácticos* se presentará la definición de sintaxis (parte de la gramática que estudia cómo se combinan las palabras) y los diferentes constituyentes sintácticos que existen.



Accede al vídeo

Información sintáctica

Es información relativa a las relaciones estructurales entre palabras. Dicha información de una palabra permite determinar las combinaciones posibles de este

elemento con el resto de los elementos de la oración. Por ejemplo, la información sintáctica de un verbo puede indicar si este puede tener asociado un objeto directo, un objeto indirecto o un complemento preposicional. En concreto, los verbos transitivos exigen la presencia de un objeto directo y este conocimiento lingüístico sobre la estructura sintáctica de los verbos transitivos debe modelarse correctamente para poder realizar el análisis sintáctico.

Las relaciones estructurales entre palabras se pueden formular mediante reglas

Para indicar que en español un elemento que sea un **sintagma nominal**, por ejemplo, está **compuesto por dos elementos**, un determinante y un nombre (colocados en este orden específico) se puede utilizar la siguiente regla:

$$SN \rightarrow Det\ N$$

Donde:

- ▶ *SN* indica el sintagma nominal.
- ▶ *Det*: el determinante.
- ▶ *N*: el nombre.

A partir de esta información sintáctica se podrá determinar que el elemento «la mesa» es un sintagma nominal correcto porque se compone del determinante «la» seguido del nombre «mesa».

Una definición formal de la estructura sintáctica de una lengua es imprescindible para poder realizar correctamente el análisis sintáctico. Las **gramáticas de estructura sintagmática** que se presentan a continuación modelan la información sintáctica y, por tanto, recogen la información relativa a las relaciones estructurales de una lengua necesaria en el análisis sintáctico realizado en las tareas de procesamiento del lenguaje natural.

Cabe destacar que, en la lingüística moderna, se están extendiendo las llamadas **gramáticas valenciales o gramáticas de dependencias**.

Estas gramáticas, a diferencia de las de estructura sintagmática, se basan en dependencias y no en los constituyentes sintácticos.

Por lo que en las gramáticas de dependencias se modelan los elementos léxicos y las relaciones existentes entre estos elementos, tal como se muestra en el último apartado.

4.3. Gramáticas de estructura sintagmática

Una gramática de estructura sintagmática **permite definir la estructura formal del lenguaje**. En esta sección se define el concepto y los elementos de una gramática de estructura sintagmática y, además, se identifican los diferentes tipos: gramáticas de estados finitos, gramáticas libres de contexto, gramáticas sensibles al contexto, gramáticas enumerables recursivamente y gramáticas generativas transformacionales.

Las páginas señaladas corresponden al apartado «1.2. Las gramáticas formales» del capítulo indicado para el estudio de esta sección».

4.4. Ambigüedad en el análisis sintáctico

Uno de los mayores problemas que se dan en el análisis sintáctico es la ambigüedad.

La ambigüedad estructural se debe a las múltiples reglas representadas en una gramática, que provienen del uso común de la lengua y que permiten que se puedan encontrar varios resultados del análisis sintáctico que se ajusten correctamente a la analizada.

En una frase en la que aparezca la coordinación «hombres y mujeres mayores», será difícil determinar en el análisis sintáctico si el emisor de la frase se refiere a que tanto los hombres como las mujeres son mayores o, por el contrario, se refiere a todos los hombres, sean o no mayores, y a las mujeres mayores.

- ▶ En el primer caso, el adjetivo «mayores» está asociado al sintagma nominal compuesto de la coordinación de los dos nombres: «hombres» y «mujeres». Esta relación se podría escribir utilizando corchetes como [[hombres y mujeres] mayores].
- ▶ En el segundo caso, el adjetivo «mayores» está asociado solo al nombre «mujeres» y el sintagma nominal que forman estas dos palabras está relacionado a través de la conjunción copulativa «y» al nombre «hombres». Por tanto, esta relación se podría expresar utilizando la notación de corchetes como [hombres y [mujeres mayores]].

El siguiente ejemplo ilustrativo presenta un caso de ambigüedad estructural en la que existen dos posibles resultados del análisis sintáctico de una oración y que se representan gráficamente en forma de árboles sintácticos.

Al final, el analizador sintáctico que obtiene esos dos resultados debe optar por resolver la ambigüedad eligiendo uno de los árboles sintácticos como salida del proceso de análisis.

Ejemplo ilustrativo 1

Ambigüedad en análisis sintáctico de la frase de Groucho Marx en la película Animal Crackers (1930): «*I shot an elephant in my pajamas*».

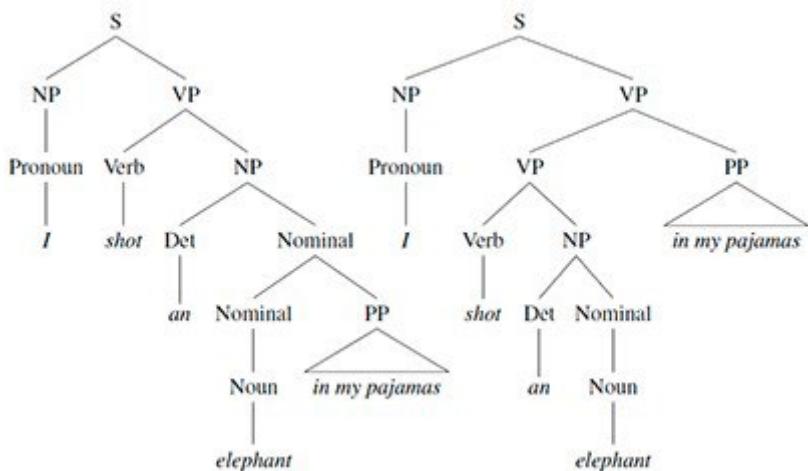


Figura 1. Árboles sintácticos que representan el resultado del análisis sintáctico.

Fuente: Jurafsky y Martin, 2009.

El análisis sintáctico de la frase «*I shot an elephant in my pajamas*» puede derivar en dos posibles soluciones y, por tanto, se observa en este caso el problema de la ambigüedad estructural. Tal como se muestra en la Figura 1, el elemento «*in my pajamas*» puede ser parte del sintagma nominal (NP) cuyo núcleo es el nombre (*noun*) «*elephant*», en el árbol sintáctico de la izquierda, o parte del predicado verbal (VP) cuyo núcleo es el verbo (*verb*) «*shot*», en el árbol sintáctico de la derecha.

El problema de la ambigüedad en la estructura de las frases afecta el funcionamiento de la mayoría de los sistemas de procesamiento del lenguaje natural. Entonces, también afecta los analizadores sintácticos que utilizan gramáticas libres de contexto y que se basan en búsqueda, ya sea ascendente o descendente, presentados en el tema anterior.

Por lo tanto, esos analizadores deben ser capaces de elegir un único resultado correcto del análisis de entre la multitud de resultados posibles.

La selección del mejor resultado del análisis, es decir, del mejor árbol sintáctico, se realiza normalmente a través de un **proceso de desambiguación sintáctica** que requiere de fuentes de conocimiento estadístico, semántico y contextual.

Métodos más avanzados evitan el uso de conocimiento lingüístico para realizar la desambiguación *a posteriori* e integran el **proceso de desambiguación** directamente en el análisis sintáctico. Algunos de estos métodos se basan en programación dinámica y otros, en métodos probabilísticos que se presentan en este tema.

4.5. Métodos para el análisis sintáctico, basados en la programación dinámica

Técnicas de programación dinámica se utilizan en el análisis sintáctico para tratar el problema de la ambigüedad estructural. La programación dinámica es un método que sirve para optimizar la resolución de problemas que pueden ser discretizados y secuencializados. Así, la programación dinámica permite resolver problemas complejos de una forma más eficiente tratándolos como subproblemas más sencillos.

El principal fundamento de la programación dinámica es que las soluciones óptimas de subproblemas permiten encontrar la solución óptima del problema en su conjunto.

Basándose en esta idea, en la programación dinámica se completan de forma sistemática tablas de soluciones para los diferentes subproblemas y, cuando estas tablas están completas, contienen la solución a todos los subproblemas necesarios para resolver el problema global.

La programación dinámica se utiliza en el análisis sintáctico para tratar el problema de la ambigüedad. Para ello, los subproblemas representan los posibles árboles sintácticos para todos los constituyentes sintácticos detectados como entrada del análisis.

El algoritmo más utilizado para implementar el análisis sintáctico basado en programación dinámica es el **algoritmo de Cocke-Kasami-Younger (CKY)** (Kasami, 1965 y Younger, 1967). El algoritmo CKY se aplica cuando la gramática utilizada en el análisis sintáctico es del tipo *Chomsky Normal Form* (CNF) (Chomsky, 1963). En una gramática CNF las reglas solo pueden tener a la derecha:

- ▶ Dos símbolos no terminales (regla $A \rightarrow B C$).
- ▶ O un símbolo terminal (regla $A \rightarrow w$).

Sin embargo, esto no representa un problema ya que una gramática libre de contexto se puede transformar en una gramática CNF sin perder expresividad.

El primer paso del algoritmo CKY es convertir la gramática libre de contexto en una gramática CNF. Para ello, todas aquellas reglas que cumplen por defecto con las condiciones de una CNF (esto es, que a la derecha tengan dos símbolos no terminales o un símbolo terminal) son directamente copiadas a la nueva gramática. El resto de las reglas que no cumplen con las condiciones de una gramática CNF son transformadas como se describe a continuación.

Aquellas reglas que mezclan símbolos terminales y no terminales son alteradas con la introducción de un no símbolo terminal comodín que involucra únicamente el símbolo terminal original. Por ejemplo, una regla para un verbo en infinitivo en inglés como $INF-VP \rightarrow to VP$ sería sustituida por las dos reglas $INF-VP \rightarrow TO VP$ y $TO \rightarrow to$. Ahora sí, estas nuevas reglas cumplen las condiciones de una gramática CNF.

Las reglas con un solo símbolo no terminal a la derecha, llamémosle **unitarias**, se pueden eliminar reescribiendo la parte derecha de las reglas originales con el lado

derecho de todas aquellas reglas no unitarias. Más formalmente, si $A \Rightarrow B$ es una cadena de una o más reglas unitarias y $B \rightarrow \gamma$ es una regla no unitaria, entonces podemos añadir $A \rightarrow \gamma$ para cada regla en la gramática y eliminar todas las reglas unitarias.

Se puede demostrar que esta operación puede conducir a un aplanamiento sustancial de la gramática y la consiguiente promoción de terminales a niveles bastante altos en los árboles resultantes.

Las reglas con más de dos términos en el lado derecho se normalizan a través de la introducción de nuevos símbolos no terminales que extienden las secuencias más largas en varias reglas nuevas. Formalmente:

$$A \rightarrow B C \gamma$$

Entonces, si tenemos una regla como la anterior, podemos reemplazar el par más a la izquierda de los símbolos no terminales con un nuevo símbolo no terminal e introducir las siguientes nuevas reglas:

$$A \rightarrow X_1 \gamma$$

$$X_1 \rightarrow B C$$

Este proceso se puede repetir tantas veces como sea necesario hasta alcanzar reglas de una longitud 2. La elección del par de símbolos no terminales a reemplazar es puramente arbitraria; cualquier procedimiento sistemático que resulte en reglas binarias es adecuado. Una vez se hayan convertido todas las reglas en binarias se añaden a la nueva gramática, finalizando aquí el proceso de conversión a CNF.

En el vídeo *Paso 1 del algoritmo CKY: conversión de la gramática libre de contexto en una gramática CNF* se verá el primer paso para aplicar el algoritmo CKY para el análisis sintáctico. Este consiste en convertir la gramática en un formato libre de contexto al formato CNF.



Accede al vídeo

Una vez tengamos la gramática en formato CNF, podemos aplicar el algoritmo CKY, el cual nos permite llevar a cabo el proceso de reconocimiento sintáctico. Con nuestra gramática en CNF, cada nodo no terminal por encima del nivel de categoría gramatical en un árbol sintáctico tendrá exactamente dos hijos. Se puede usar una matriz de dos dimensiones para codificar la estructura de todo un árbol.

Para una frase de longitud n , trabajaremos con la parte superior triangular de una matriz:

$$(n + 1) \times (n + 1)$$

Cada celda $[i, j]$ de esta matriz contiene el conjunto de símbolos no terminales que representan todos los constituyentes sintácticos que abarcan posiciones de entrada desde i hasta j . Ya que nuestro sistema de indexación comienza en 0, se deduce que la celda que representa la entrada completa reside en la posición $[0, n]$ de la matriz.

Dado que cada entrada no terminal en nuestra tabla tiene dos hijos en el análisis sintáctico, podemos decir que:

- ▶ Para cada constituyente sintáctico representado por una entrada $[i, j]$, tiene que haber una posición en la entrada k , que puede ser dividida en dos partes de tal manera que:

$$i < k < j$$

- ▶ Dada una posición k :

- El primer constituyente sintáctico $[i, k]$ debe estar a la izquierda de la entrada $[i, j]$ en algún lugar a lo largo de la fila i .
- Y la segunda entrada $[k, j]$ debe encontrarse por debajo de aquel, a lo largo de la columna j .

La superdiagonal en la matriz contiene las categorías gramaticales para cada una de las palabras de la oración. Las subsiguientes diagonales por encima de la superdiagonal contienen los constituyentes sintácticos para los diferentes tramos de longitud creciente en la oración.

Vista esta configuración, el reconocimiento CKY consiste en rellenar la tabla de análisis sintáctico de la manera correcta.

Para ello, se procederá de abajo hacia arriba de manera que, a la hora de llenar la celda $[i, j]$, las celdas que pueden contener partes que podrían contribuir a esta entrada de la tabla (las celdas a la izquierda y debajo) deben estar ya llenas.

El pseudocódigo del algoritmo se muestra en la Figura 2. Dicho algoritmo rellena la matriz triangular superior operando por columnas de abajo a arriba y de izquierda a derecha como se muestra en la Figura 3. Este esquema garantiza que en cada momento tengamos toda la información necesaria.

```
function CKY-PARSE(words, grammar) returns table
  for j  $\leftarrow$  from 1 to LENGTH(words) do
    for all {A | A  $\rightarrow$  words[j]  $\in$  grammar} do
      table[j - 1, j]  $\leftarrow$  table[j - 1, j]  $\cup$  A
    for i  $\leftarrow$  from j - 2 downto 0 do
      for k  $\leftarrow$  i + 1 to j - 1 do
        for all {A | A  $\rightarrow$  BC  $\in$  grammar and B  $\in$  table[i, k] and C  $\in$  table[k, j]} do
          table[i, j]  $\leftarrow$  table[i, j]  $\cup$  A
```

Figura 2. Pseudocódigo del algoritmo CKY. Fuente: Jurafsky y Martin, 2009.

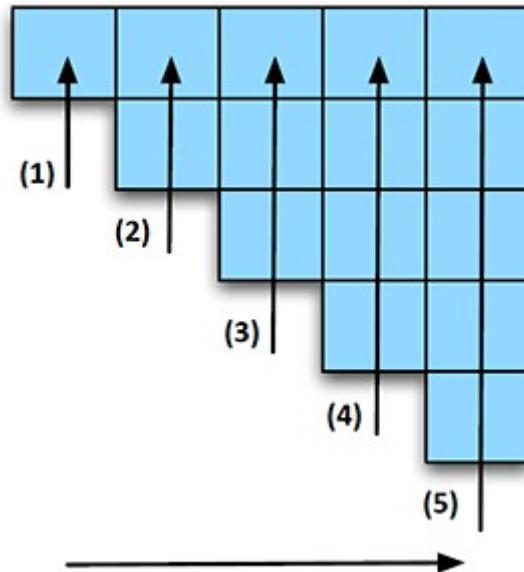


Figura 3. Secuencia seguida para llenar la tabla en el proceso de reconocimiento CKY. Fuente: Jurafsky y Martin, 2009.

La Figura 4 muestra el caso general de relleno de la celda $[i, j]$. En cada partición, el algoritmo considera si los contenidos de las dos celdas pueden ser combinados de modo que se cumpla alguna de las reglas de la gramática.

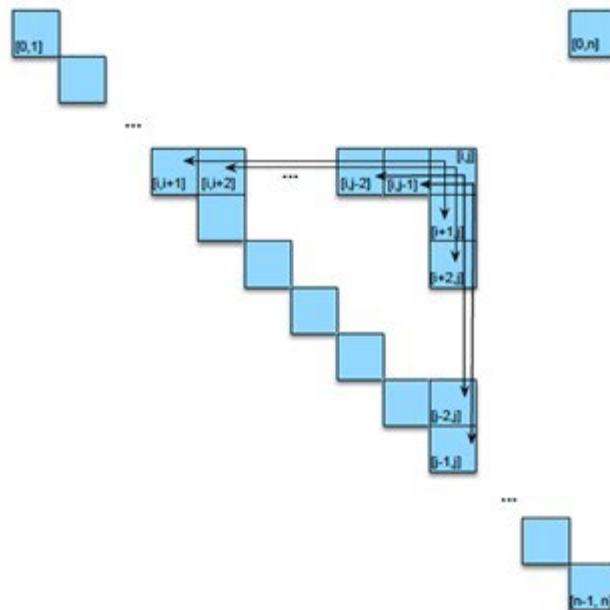


Figura 4. Todas las posibles formas de llenar la celda $[i, j]$ en la tabla CKY. Fuente: Jurafsky y Martin, 2009.

Ejemplo ilustrativo 2

Algoritmo CKY para el reconocimiento sintáctico de la frase en inglés «*Book the flight through Houston*» (en español, «Reserva el vuelo a través de Houston»).

<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb Nominal, Noun [0, 1]	[0, 2]	S, VP, X2 [0, 3]	[0, 4]	S, VP, X2 [0, 5]
Det [1, 2]	NP [1, 3]			NP [1, 5]
		[1, 4]		
		Nominal, Noun [2, 3]		Nominal [2, 5]
			Prep [3, 4]	PP [3, 5]
				NP, Proper- Noun [4, 5]

Figura 5. Matriz del análisis sintáctico al aplicar el algoritmo CKY. Fuente: Jurafsky y Martin, 2009.

La Figura 5 muestra la matriz del análisis sintáctico completo para la frase «*Book the flight through Houston*» cuando se utiliza la gramática en formato CNF presentada en la Figura 6 y 7 para realizar el reconocimiento sintáctico.

\mathcal{L}_1 Grammar	\mathcal{L}_1 in CNF
$S \rightarrow NP VP$	$S \rightarrow NP VP$
$S \rightarrow Aux NP VP$	$S \rightarrow XI VP$
$S \rightarrow VP$	$XI \rightarrow Aux NP$
	$S \rightarrow book include prefer$
	$S \rightarrow Verb NP$
	$S \rightarrow X2 PP$
	$S \rightarrow Verb PP$
	$S \rightarrow VP PP$
$NP \rightarrow Pronoun$	$NP \rightarrow I she me$
$NP \rightarrow Proper-Noun$	$NP \rightarrow TWA Houston$
$NP \rightarrow Det Nominal$	$NP \rightarrow Det Nominal$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book flight meal money$
$Nominal \rightarrow Nominal Noun$	$Nominal \rightarrow Nominal Noun$
$Nominal \rightarrow Nominal PP$	$Nominal \rightarrow Nominal PP$
$VP \rightarrow Verb$	$VP \rightarrow book include prefer$
$VP \rightarrow Verb NP$	$VP \rightarrow Verb NP$
$VP \rightarrow Verb NP PP$	$VP \rightarrow X2 PP$
$VP \rightarrow Verb PP$	$X2 \rightarrow Verb NP$
$VP \rightarrow VP PP$	$VP \rightarrow Verb PP$
$PP \rightarrow Preposition NP$	$VP \rightarrow VP PP$
	$PP \rightarrow Preposition NP$

Figura 6. Gramática libre de contexto (columna izquierda) y en formato CNF (columna derecha) y léxico (abajo) para realizar el análisis sintáctico en lengua inglesa. Fuente: Jurafsky y Martin, 2009.

Lexicon
<i>Det → that this the a</i>
<i>Noun → book flight meal money</i>
<i>Verb → book include prefer</i>
<i>Pronoun → I she me</i>
<i>Proper-Noun → Houston NWA</i>
<i>Aux → does</i>
<i>Preposition → from to on near through</i>

Figura 7. Gramática libre de contexto (columna izquierda) y en formato CNF (columna derecha) y lexicon (abajo) para realizar el análisis sintáctico en lengua inglesa. Fuente: Jurafsky y Martin, 2009.

El ejemplo concreto de cómo se rellenarían las celdas de la columna 5 de la matriz después de leer la palabra «Houston» se muestra en la Figura 8. Las flechas señalan los elementos de las dos celdas que se utilizan para agregar una entrada a la tabla.

Cabe destacar que en la celda [0; 5] se indica la presencia de tres posibles análisis alternativos para esta entrada:

- ▶ Uno, donde la preposición (PP) modifica la palabra «flight».
- ▶ Otro, donde esta misma preposición modifica la palabra «book».
- ▶ Y el tercero, que captura la regla $VP \rightarrow X_2 PP$.

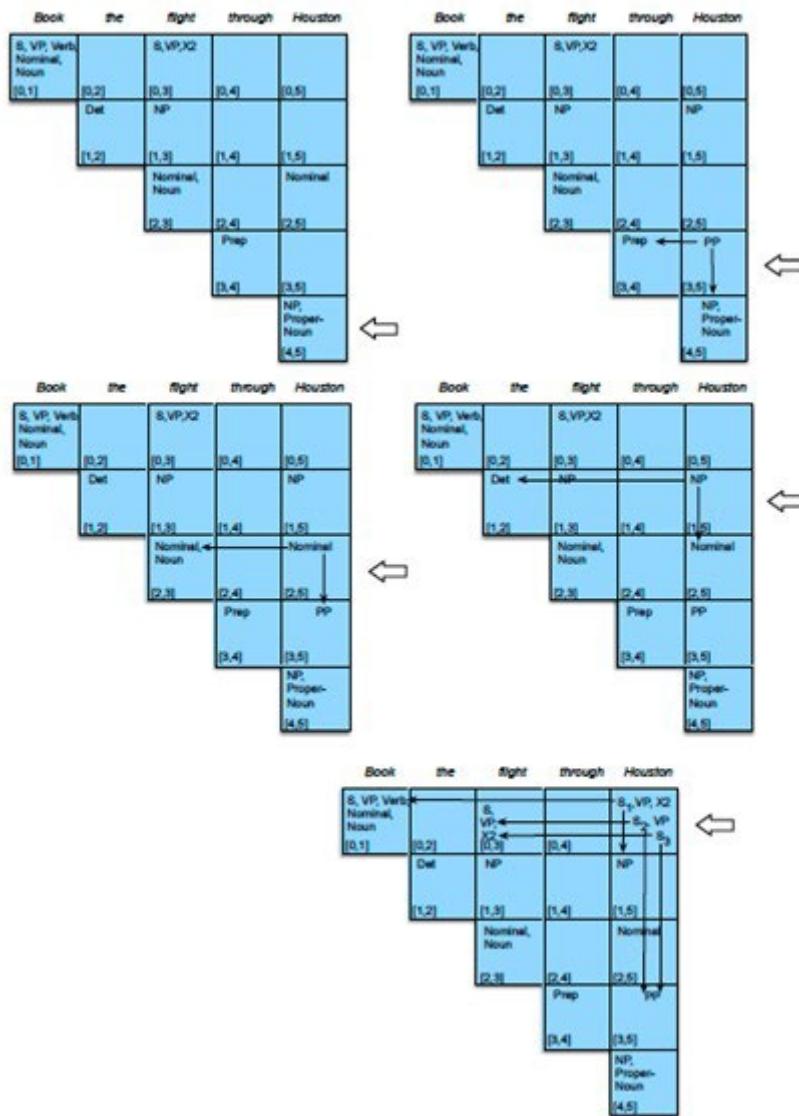


Figura 8. Rellenado de las celdas de la columna 5 después de leer la palabra «Houston». Fuente: Jurafsky y Martin, 2009

En el vídeo *Paso 2 del algoritmo CKY: reconocimiento sintáctico* se verá el segundo paso para aplicar el algoritmo CKY para el análisis sintáctico. Este consiste en crear la matriz que codifique los diferentes árboles sintácticos.



Accede al vídeo

Finalmente, llegamos a la **fase de análisis sintáctico**. Hay que tener en cuenta que el algoritmo mostrado en la Figura 3 es un reconocedor sintáctico, no un analizador sintáctico. Por tanto, para convertirlo en un analizador capaz de devolver todos los

análisis sintácticos posibles para una entrada dada, podemos hacer dos simples cambios en el algoritmo.

- ▶ El primer cambio es aumentar las entradas de la tabla de manera que cada símbolo no terminal esté emparejado con punteros a las entradas de la tabla de las que se derivaron (parecido a como se muestra en la Figura 8 del ejemplo ilustrativo 2).
- ▶ El segundo cambio consiste en permitir múltiples versiones del mismo símbolo no terminal para ser introducidos en la tabla (véase la Figura 8).

Con estos cambios, el cuadro completo contiene todos los posibles análisis sintácticos para una entrada dada. Devolver un único análisis sintáctico arbitrario consiste en la elección de una oración S de la celda $[0, n]$ y luego recuperar de forma recursiva sus constituyentes sintácticos de la tabla.

En el vídeo *Paso 3 del algoritmo CKY: análisis sintáctico* se explicará el tercer paso del algoritmo CKY, donde se realiza la decodificación o la obtención de los distintos árboles sintácticos a partir de la matriz que se ha creado anteriormente.



Accede al vídeo

4.6. Métodos probabilistas en el análisis sintáctico

La ambigüedad estructural se puede modelar de forma eficiente utilizando el algoritmo CKY presentado en el apartado anterior. Sin embargo, el analizador sintáctico va a devolver múltiples resultados del análisis y para poder resolver la ambigüedad y llegar a una única solución es necesario **implementar métodos probabilistas** en él. De hecho, la mayoría de los analizadores sintácticos utilizados hoy en día implementan métodos probabilistas como método de **desambiguación**.

Los analizadores sintácticos probabilistas calculan la probabilidad de cada posible interpretación del análisis sintáctico y escogen la más probable.

Formalmente, un analizador sintáctico probabilista tiene como objetivo producir el análisis sintáctico más probable \hat{T} para una oración dada S :

$$\hat{T}(S) = \operatorname{argmax}_{T \text{ s.t. } S = \text{yield}(T)} P(T)$$

Por lo tanto, el analizador sintáctico probabilista va a computar el árbol sintáctico T que maximice $P(T)$, esto es, la probabilidad de dicho árbol. Dicho de otra forma, el analizado sintáctico va a buscar el árbol sintáctico más probable. Notar que la cadena de palabras S se denomina el «*yield*» de cualquier árbol sintáctico que se puede aplicar a la oración a analizar.

Para calcular la **probabilidad de un árbol sintáctico** $P(T)$ de una oración S a partir de las probabilidades de las reglas de una gramática, se considera que las reglas son independientes y se calcula la probabilidad del árbol multiplicando las probabilidades de cada una de las reglas involucradas en el análisis de la oración.

Para calcular la **probabilidad asociada a una regla** representada en una gramática libre de contexto existen dos maneras. La forma más sencilla es utilizar un *treebank* (Marcus, Santorini y Marcinkiewicz, 1993), esto es un corpus de frases ya analizados. Dado un *treebank*, podemos calcular la probabilidad de cada expansión de un símbolo no terminal mediante el recuento del número de veces que se produce la expansión y luego normalizarlo, tal como se muestra en la siguiente fórmula:

$$P(\alpha \rightarrow \beta | \alpha) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\sum_{\gamma} \text{Count}(\alpha \rightarrow \gamma)} = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

Si no tenemos un *treebank*, pero sí tenemos un analizador sintáctico (no probabilístico), podemos generar los recuentos que necesitamos para el cálculo de

las probabilidades asociadas a cada regla. Para ello, analizamos sintácticamente el corpus de frases con el analizador. Si las frases no son ambiguas, sería tan simple como analizar todo el corpus, incrementar un contador para cada regla dada en el análisis sintáctico y, luego, normalizar para obtener las probabilidades.

Como la mayor parte de las oraciones son ambiguas, es decir, tienen múltiples análisis sintácticos, tenemos que mantener una cuenta separada para cada análisis sintáctico de una oración y ponderar cada uno de estos recuentos parciales por la probabilidad del análisis sintáctico en el que aparece. Pero para obtener las probabilidades para ponderar las reglas, debemos tener ya un analizador sintáctico probabilístico, lo cual nos lleva a un problema recurrente.

La forma de resolver este problema es mejorar de forma creciente nuestras estimaciones:

- ▶ Comenzar con un analizador sintáctico con probabilidades iguales para cada regla.
- ▶ Seguidamente, analizar la frase.
- ▶ Calcular la probabilidad para cada análisis sintáctico.
- ▶ Utilizar estas probabilidades para ponderar los contadores.
- ▶ Reestimar las probabilidades de cada regla, y así sucesivamente, hasta que nuestras probabilidades converjan.

El algoritmo estándar para el cálculo de esta solución se llama *inside-outside* (Baker, 1979) y es un caso especial del algoritmo *Expectation-Maximization* (Manning y Schütze, 1999).

Algoritmo CKY probabilístico

La mayoría de los analizadores sintácticos probabilistas modernos se basan en el algoritmo CKY probabilístico (Ney, 1991), una versión probabilística del algoritmo CKY presentado en la sección anterior.

El algoritmo CKY probabilístico extiende el algoritmo CKY para incluir información probabilística.

La versión probabilística del algoritmo CKY, igual que el algoritmo CKY básico, utiliza una gramática CNF (*Chomsky Normal Form*). Sin embargo, en la versión probabilística, cada regla está anotada con la probabilidad de que se cumpla dicha regla.

Por lo tanto, el **primer paso del algoritmo CKY probabilístico** consiste en convertir las reglas de una gramática libre de contexto anotada con probabilidades al **formato CNF**. Para realizar este proceso se aplica un método análogo al utilizado para el algoritmo CKY básico. En la conversión de las reglas al formato CNF, además de aplicar los criterios explicados en la sección anterior, **se deben recalcular las probabilidades de modo que la probabilidad asociada a cada posible árbol resultante del análisis sintáctico de la oración permanezca constante para la nueva gramática CNF**.

En el **segundo paso del algoritmo CKY probabilístico**, y una vez se tienen las reglas en formato CNF, **se crearía una matriz similar a la del CKY básico, pero con una dimensión adicional**. Recordemos que en el CKY básico cada celda de la matriz contenía una lista con los constituyentes sintácticos para cada palabra. En el caso CKY probabilístico, cada celda tiene una dimensión adicional que se utiliza para indexar cada constituyente sintáctico. Específicamente, para una frase de longitud n (palabras) y una gramática que contiene V símbolos no terminales (constituyentes sintácticos), tendríamos una matriz $(n + 1) \times (n + 1) \times V$. El valor de cada una de las celdas corresponde en este caso a la probabilidad de cada constituyente o símbolo no terminal para cada palabra.

El pseudocódigo del algoritmo CKY probabilístico se presenta en la Figura 9.

```

function PROBABILISTIC-CKY(words,grammar) returns most probable parse
    and its probability
    for j ← from 1 to LENGTH(words) do
        for all { A | A  $\rightarrow$  words[j]  $\in$  grammar }
            table[j − 1, j, A]  $\leftarrow P(A \rightarrow words[j])$ 
        for i ← from j − 2 downto 0 do
            for k ← i + 1 to j − 1 do
                for all { A | A  $\rightarrow$  BC  $\in$  grammar,
                    and table[i,k,B] > 0 and table[k,j,C] > 0 }
                    if (table[i,j,A] <  $P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C]$ ) then
                        table[i,j,A]  $\leftarrow P(A \rightarrow BC) \times table[i,k,B] \times table[k,j,C]$ 
                        back[i,j,A]  $\leftarrow \{k, B, C\}$ 
    return BUILD_TREE(back[1, LENGTH(words), S]), table[1, LENGTH(words), S]

```

Figura 9. Pseudocódigo del algoritmo CKY probabilístico. Fuente: Jurafsky y Martin, 2009.

Ejemplo ilustrativo 3

Algoritmo CKY probabilístico para el reconocimiento sintáctico de la frase en inglés «*The flight includes a meal*», en español significa «El vuelo incluye comida».

<i>The</i>	<i>flight</i>	<i>includes</i>	<i>a</i>	<i>meal</i>
Det: .40	NP: .30 * .40 * .02 = .0024			
[0,1]	[0,2]	[0,3]	[0,4]	[0,5]
	N: .02			
	[1,2]	[1,3]	[1,4]	[1,5]
		V: .05		
		[2,3]	[2,4]	[2,5]
			Det: .40	
			[3,4]	[3,5]
				N: .01
				[4,5]

Figura 10. Matriz para el análisis sintáctico aplicando el algoritmo CKY probabilístico. Fuente: Jurafsky y Martin, 2009.

Los primeros pasos de aplicar el algoritmo probabilístico CKY para analizar la frase «*The flight includes a meal*», se muestran en la matriz de la Figura 10. La gramática en formato CNF que incluye las probabilidades de cada una de las

reglas y que se utiliza para realizar el reconocimiento sintáctico se presenta en la Figura 11.

$S \rightarrow NP VP .80$	$Det \rightarrow the .40$
$NP \rightarrow Det N .30$	$Det \rightarrow a .40$
$VP \rightarrow V NP .20$	$N \rightarrow meal .01$
$V \rightarrow includes .05$	$N \rightarrow flight .02$

Figura 11. Gramática de la lengua inglesa en formato CNF donde cada regla está anotada con la probabilidad y que se utiliza para realizar el análisis sintáctico. Fuente: Jurafsky y Martin, 2009.

En el vídeo *Algoritmo CKY probabilístico* se explicará este, que amplía el CKY clásico y permite añadir probabilidades para obtener la probabilidad de cada uno de los árboles sintácticos codificados en la matriz.



Accede al vídeo

4.7. Gramáticas de dependencias o gramáticas valenciales

valenciales

Una gramática valencial modela las dependencias entre los elementos léxicos de una oración y, de este hecho, proviene su nombre de gramática de dependencias.

A diferencia de las gramáticas de estructura sintagmática, que modelan los constituyentes sintácticos de una oración, las gramáticas de dependencias describen la estructura sintáctica de una oración únicamente en términos de las palabras (o lemas) que la componen y del conjunto de las relaciones gramaticales establecidas entre las palabras.

Ejemplo ilustrativo 1

Análisis sintáctico utilizando una gramática de dependencias.

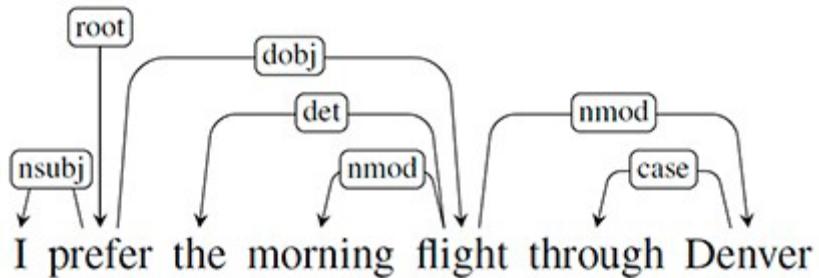


Figura 12. Resultado del análisis sintáctico utilizando una gramática de dependencias. Fuente: Jurafsky y Martin, 2009.

El análisis sintáctico de la oración en inglés «*I prefer the morning flight through Denver*» (en español, «Prefiero un vuelo a través de Denver por la mañana»).

Utilizando una gramática de dependencias, produciría la salida presentada en la Figura 1. De la que se desprende que la palabra «*prefer*» (prefiero) es la raíz de la oración y que esta se relaciona a través de una dependencia del tipo *dobj* (objeto directo) con la palabra «*flight*» (vuelo). A su vez, la palabra «*flight*» se relaciona a través de una dependencia del tipo *det* (determinante) con la palabra «*flight*». Cada relación se da entre un origen (*head*) y su elemento dependiente (*dependent*).

El resultado del análisis sintáctico utilizando una gramática de dependencias (Figura 12) se puede comparar con el árbol sintáctico (Figura 13) resultante de realizar el análisis sintáctico de la misma oración utilizando una gramática de estructura sintagmática como podría ser una gramática libre de contexto.

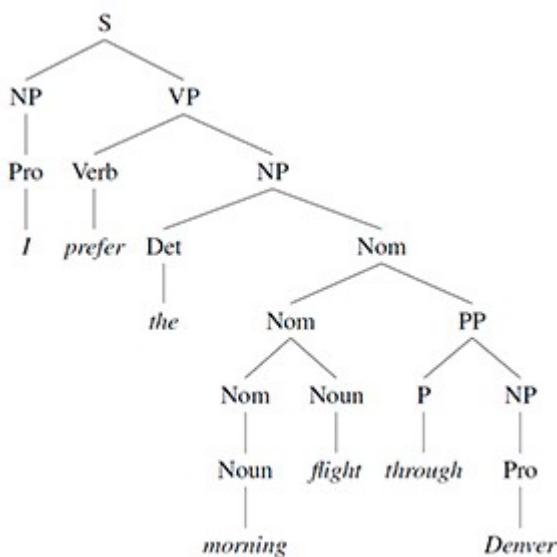


Figura 13. Árbol sintáctico que representan el resultado del análisis sintáctico utilizando una gramática de estructura sintagmática. Fuente: Jurafsky y Martin, 2009.

El análisis sintáctico utilizando gramáticas de dependencias se popularizó en la década de los 90. Por lo tanto, en las últimas décadas han aparecido diferentes analizadores sintácticos de dependencias basados en principios diversos: en programación dinámica (Eisner, 1996), en enfoques transicionales deterministas (Covington, 2001) (Nivre y Scholz, 2004), en aprendizaje automático supervisado (Yamada y Matsumoto, 2003) o en representaciones gráficas (McDonald et al., 2005).

Entre las ventajas que tienen estas gramáticas aparece su habilidad para trabajar con idiomas que son morfológicamente ricos y dónde se tiene libertad para ordenar las palabras de distinta manera dentro de una misma oración. En estos casos, se pueden tener, por ejemplo, objetos gramaticales que pueden aparecer bien antes o bien después de un adverbio de lugar.

Una gramática de estructura sintagmática necesitaría reglas que modelasen todas las posibles combinaciones. En cambio, las gramáticas de dependencias son agnósticas a esto.

Otra ventaja de las gramáticas de dependencia es que las relaciones *head-dependent* tienen una similitud respecto a las relaciones semánticas entre predicados y sus argumentos. En estas gramáticas esta información se ve de manera directa, a diferencia de las gramáticas de estructura sintagmática, donde esta información se tiene que extraer mediante técnicas adicionales sobre los árboles generados.

Así, las gramáticas de dependencias permiten detectar las relaciones entre -*head-dependent*- identificándolas con una relación gramatical concreta según la función gramatical del *dependent* con respecto a su *head*.

Algunas de estas categorías son:

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	Description
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	Description
CONJ	Conjunct
CC	Coordinating conjunction

Figura 14. Lista de algunas de las posibles relaciones gramaticales que pueden aparecer con las gramáticas de dependencias. Fuente: Jurafsky y Martin, 2009.

Las gramáticas de dependencias se caracterizan por estar formadas según una estructura de grafos $G = (V, A)$, donde se tiene una relación entre los vértices V (las palabras) mediante una serie de arcos (A). Estas relaciones cumplen que:

- ▶ Siempre hay un único nodo raíz que no tiene arcos que llevan a él.
- ▶ Excepto para el nodo raíz, los demás nodos tienen un único arco que lleva a ellos.
- ▶ Hay un único camino entre el nodo raíz y los demás vértices V .

Con esto se garantiza que cada palabra tenga un único *head*, que la estructura de dependencia esté conectada, y que haya un único nodo raíz desde el que se pueda partir para seguir un camino único hacia cualquiera de las palabras de la frase. Junto con esto, aparece otra propiedad en los arcos *head-dependent* denominada **proyectividad**. Esta se da si existe un camino entre el *head* y todas las palabras que aparecen entre él y su *dependent*. Por ejemplo, en la frase de la Figura 12 todos los arcos son proyectivos. Sin embargo, en la frase siguiente el arco entre *flight* y *was* no es proyectivo ya que no existe un camino entre *flight* y *this* o *morning* considerando sólo las palabras que están entre *head-dependent* (*flight-was*).

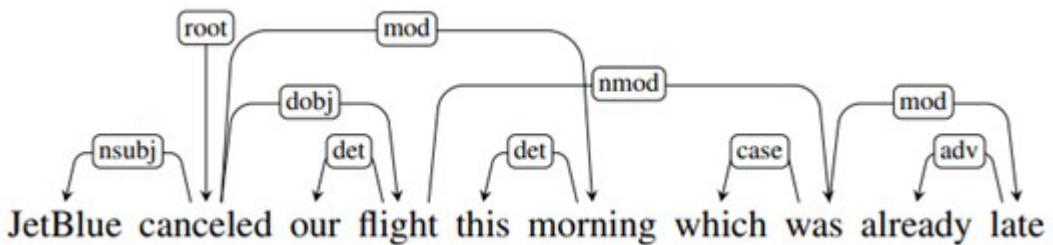


Figura 15. Ejemplo de salida de una gramática de dependencia donde no todos los arcos son proyectivos.

Fuente: Jurafsky y Martin, 2009.

Con ello, cuando todos los arcos de la gramática de dependencia son proyectivos se dice por extensión que el árbol de dependencias es proyectivo.

Algoritmos para gramáticas de dependencias

De cara a construir un algoritmo que lleve a cabo un análisis con gramáticas de dependencias, es importante partir de *treebanks* donde ya exista información anotada respecto a frases y sus análisis correspondientes. Estos *treebanks* están disponibles para distintas lenguas, como por ejemplo el de (Weischedel et al., 2011).

Partiendo de esto, una de las aproximaciones más conocidas para las gramáticas de dependencias es el *transitioned-based dependency parsing*. Con este algoritmo, ilustrado en la siguiente figura, se tienen distintos elementos: una pila donde están los elementos sobre los que se hace el análisis, un *buffer* con los tokens que todos los tokens sobre los que no se ha hecho aún el análisis, y un oráculo que dará como salida la acción concreta que hacer dentro de la iteración del algoritmo. Las acciones que puede llevar a cabo el oráculo se dividen en tres:

- ▶ **LeftArc:** Definir una relación *head-dependent* entre la palabra en la posición primera de la pila con respecto a la segunda palabra. Eliminar la segunda palabra de la pila.

- ▶ **RightArc:** Definir una relación *head-dependent* entre la segunda palabra de la pila y la primera. Eliminar la primera palabra de la pila.
- ▶ **Shift:** Pasar la primera palabra del *buffer* a la pila.

Con ello, la Figura 16 muestra el esquema del algoritmo, y la Figura 17 muestra un ejemplo de las salidas que se van obteniendo para cada iteración.

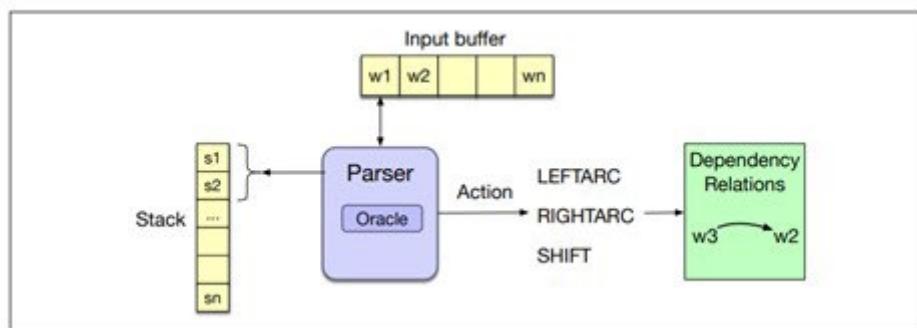


Figura 16. Esquema del algoritmo de *transitioned-based dependency parsing*. Fuente: Jurafsky y Martin, 2009.

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

Figura 17. Ejemplo de salida para cada iteración del algoritmo *transitioned-based dependency parsing*. Fuente: Jurafsky y Martin, 2009.

El punto fundamental es la creación del oráculo; es decir, cómo tener un sistema que determine en cada paso cuál de las tres acciones se deben ejecutar. Este se puede generar mediante el uso de algoritmos de aprendizaje supervisado entrenados sobre la información que proviene de los *treebanks*.

4.8. Referencias bibliográficas

- Baker, J. K. (1979). Trainable grammars for speech recognition. En D. H. Klatt y J. J. Wolf (Eds.), *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America* (pp. 547-550). Acoustical Society of America.
- Chomsky, N. (1963). Formal properties of grammars. En R. D. Luce, R. Bush y E. Galanter (Eds.), *Handbook of Mathematical Psychology* (Vol. 2, pp. 323-418). Cambridge, Estados Unidos: Wiley.
- Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall.
- Kasami, T. (1965). *An efficient recognition and syntax analysis algorithm for context-free languages*. (Informe No. R-257). Universidad de Illinois.
- Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. The MIT Press.
- Marcus, M. P., Santorini, B. y Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2), 313-330.
- Ney, H. (1991). Dynamic programming parsing for contextfree grammars in continuous speech recognition. *IEEE Transactions on Signal Processing*, 39(2), 336-340.
- RAE. (s. f.). Sintaxis. En *Diccionario de la lengua española* (actualización de la 23^a ed.). <http://dle.rae.es/?id=XzfiT9q>
- Real Academia Española. (2009). *Nueva gramática de la lengua española*. Espasa.

Younger, D. H. (1967). Recognition and parsing of context-free languages in time n^3 .
Information and Control, 10, 189-208.

A fondo

Modelos probabilísticos para el análisis sintáctico de dependencias

Eisner, J. (1996). Three new probabilistic models for dependency parsing: An exploration. Proceedings of the 16th International Conference on Computational Linguistics (COLING-96), 340-345. <https://cs.jhu.edu/~jason/papers/eisner.coling96.pdf>

El artículo presenta tres métodos probabilísticos para el análisis sintáctico basado en una gramática de dependencias. Los métodos propuestos se evalúan utilizando el conocido corpus Wall Street Journal (WSJ).

Análisis sintáctico de dependencias

Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall. <https://web.stanford.edu/~jurafsky/slp3/14.pdf>

El capítulo 14 de esta edición describe en detalle los aspectos relacionados con el análisis sintáctico de dependencias, comentando distintas maneras de construir el oráculo, incluso mediante el uso de modelos de aprendizaje profundo.

Técnicas de procesamiento de lenguaje

Badia, T. (2003). Técnicas de procesamiento de lenguaje. En M. A. Martí (Coord.). *Tecnologías del lenguaje* (pp. 199-206). Editorial UOC. Disponible en la Biblioteca Virtual de UNIR.

Para completar el estudio de esta sección puedes leer las páginas 199-206 del siguiente libro: Técnicas de procesamiento de lenguaje.

Análisis sintáctico: estrategia ascendente y búsqueda en profundidad



Accede al vídeo

En este vídeo se verá desde donde parte el análisis sintáctico utilizando una estrategia ascendente y la búsqueda en profundidad. Se parte de las palabras que forman la oración y se va creando el árbol sintáctico hacia arriba.

Análisis sintáctico: estrategia descendente y búsqueda en profundidad



Accede al vídeo

En este vídeo se estudiará el análisis sintáctico usando una estrategia descendente y una búsqueda en profundidad (hacia abajo).

Análisis sintáctico: estrategia ascendente y búsqueda en anchura



Accede al vídeo

En este vídeo se verá cómo se crea el árbol de abajo hacia arriba a partir de las palabras hasta que llegamos a la composición completa. Al realizar la búsqueda en anchura se irá desarrollando el árbol por niveles

Test

1. Indica las afirmaciones correctas sobre el problema de la ambigüedad en el análisis sintáctico:

 - A. Solo afecta a un grupo reducido de analizadores sintácticos.
 - B. Se debe a un mal diseño de la gramática en la que aparecen múltiples reglas para una misma frase analizada.
 - C. Se da cuando el analizador sintáctico encuentra varios árboles sintácticos válidos y no es capaz de decidir cuál es el mejor.
 - D. Se puede resolver *a posteriori* una vez efectuado el análisis sintáctico.
2. Indica las afirmaciones correctas sobre los métodos para el análisis sintáctico-basados en programación dinámica:

 - A. Tratan el problema de la ambigüedad estructural.
 - B. Buscan soluciones óptimas a subproblemas que permiten encontrar la solución al problema en su conjunto.
 - C. Devuelven un único resultado para el análisis sintáctico.
 - D. Los posibles árboles sintácticos debidos a la ambigüedad estructural se representan como subproblemas.
3. Indica las afirmaciones correctas sobre el algoritmo de Cocke-Kasami-Younger (CKY):

 - A. Está basado en programación dinámica.
 - B. Puede incluir probabilidades para cada árbol sintáctico
 - C. Utiliza una gramática libre de contexto.
 - D. Utiliza una gramática del tipo Chomsky Normal Form (CNF).

4. Indica las afirmaciones correctas sobre una gramática CNF:

- A. Las reglas solo pueden tener a la derecha dos símbolos no terminales o un símbolo terminal.
- B. Cualquier gramática libre de contexto se podrá transformar en una gramática CNF.
- C. Una regla en una gramática libre de contexto que mezcle símbolos terminales y no terminales se podrá transformar al formato CNF como una única regla con un símbolo comodín.
- D. Las reglas unitarias en una gramática libre de contexto se podrán transformar al formato CNF reescribiendo la parte derecha de estas reglas originales con el lado derecho de todas aquellas reglas no unitarias.

5. Indica las afirmaciones correctas sobre la fase de reconocimiento sintáctico del algoritmo CKY:

- A. El algoritmo aplicado en esta fase permite obtener un único árbol sintáctico.
- B. Cada nodo del árbol sintáctico tiene exactamente dos hijos.
- C. La estructura de un árbol sintáctico se puede codificar como una matriz de dos dimensiones.
- D. El algoritmo aplicado en esta fase se puede modificar para convertirlo en un analizador sintáctico.

6. Indica las afirmaciones correctas sobre la matriz de análisis sintáctico en el algoritmo CKY:

- A. Para una frase de longitud n , se trabaja con una matriz de dimensión $n \times n$.
- B. Cada celda $[i, j]$ de esta matriz contiene el conjunto de símbolos no terminales que representan todos los constituyentes sintácticos que abarcan posiciones de entrada desde i hasta j .
- C. La superdiagonal en la matriz contiene las categorías gramaticales para cada una de las palabras de la oración.
- D. Las subsiguientes diagonales por encima de la superdiagonal contienen los constituyentes sintácticos para los diferentes tramos de longitud decreciente en la oración.

7. Indica las afirmaciones correctas sobre el funcionamiento del algoritmo CKY:

- A. Para llenar la celda $[i, j]$ de la matriz, el algoritmo considera si los contenidos de una celda en la fila i y una celda en la columna j pueden ser combinados de modo que se cumpla alguna de las reglas de la gramática.
- B. El algoritmo llena la matriz triangular superior operando por columnas de abajo a arriba y de izquierda a derecha.
- C. Una celda de la matriz puede contener varios posibles análisis sintácticos alternativos para la oración.
- D. Para devolver un único análisis sintáctico se debe elegir de entre los símbolos presentes en la celda $[0, n]$ de la matriz uno que represente la oración, por ejemplo S o O dependiendo del vocabulario utilizado en la gramática, y recuperar de forma recursiva sus constituyentes sintácticos.

- 8.** Indica las afirmaciones correctas sobre los analizadores sintácticos probabilistas:
- A. Buscan el árbol sintáctico más probable para una oración maximizando la probabilidad de dicho árbol.
 - B. Tienen como objetivo producir el análisis sintáctico más probable para una oración.
 - C. Calculan la probabilidad de cada interpretación del análisis sintáctico para una oración y escogen la más probable.
 - D. Calculan la probabilidad de un árbol sintáctico de una oración a partir de las probabilidades de las reglas involucradas en el análisis de la oración.
- 9.** Indica las afirmaciones correctas sobre el cálculo de la probabilidad asociada a una regla representada en una gramática libre de contexto:
- A. Si no se dispone de un corpus de frases analizadas sintácticamente, no se puede calcular dicha probabilidad.
 - B. Si se dispone de un *treebank*, se recuenta el número de veces que se expande un símbolo no terminal y se normaliza.
 - C. Si no se dispone de un *treebank*, pero sí se dispone de un analizador sintáctico no probabilístico, se analiza sintácticamente el corpus de frases con el analizador y si estas no son ambiguas se recuenta el número de veces que se aplica cada regla y se normaliza.
 - D. Si no se dispone de un *treebank*, se estiman las probabilidades con un algoritmo basado en Expectation Maximization, donde se parte de unas probabilidades que se van ajustando iterativamente hasta que se dé el criterio de convergencia.

10. Indica las afirmaciones correctas sobre el algoritmo CKY probabilístico:

- A. Extiende el algoritmo CKY añadiendo información sobre la probabilidad de que se cumpla cada regla.
- B. En la conversión de las reglas de una gramática libre de contexto al formato CNF, las probabilidades de las reglas permanecen constantes.
- C. Crea una matriz cuadrada con tantas celdas como la longitud de la frase y una dimensión adicional del tamaño del número de símbolos no terminales.
- D. El valor de cada una de las celdas de la matriz se corresponde con la probabilidad de cada constituyente o símbolo no terminal para cada palabra.

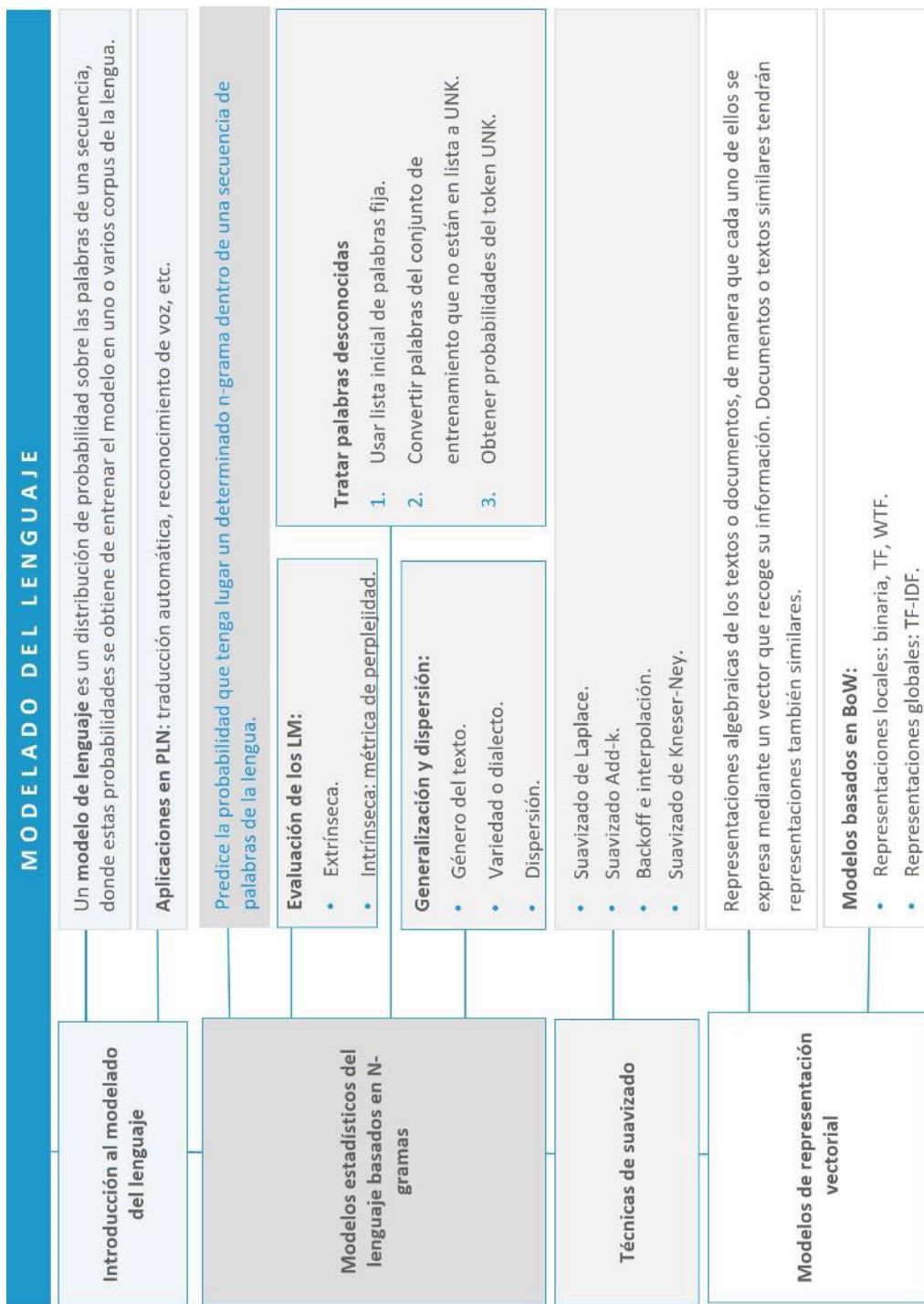
Procesamiento del Lenguaje Natural

Modelado del lenguaje

Índice

Esquema	3
Ideas clave	4
5.1. Introducción y objetivos	4
5.2. Introducción al modelo del lenguaje	4
5.3. Modelos estadísticos del lenguaje basados en N- gramas	6
5.4. Técnicas de suavizado	14
5.5. Modelos de representación vectorial	20
4.6. Referencias bibliográficas	25
A fondo	26
Test	27

Esquema



5.1. Introducción y objetivos

A continuación, profundizaremos en qué es el modelado del lenguaje, y cómo se puede llevar a cabo con técnicas estadísticas como en el caso del uso de n-gramas. Además, veremos algunas limitaciones de esta aproximación y cómo se pueden solucionar con distintas técnicas de suavizado. Finalmente, presentaremos el concepto de modelado vectorial para representar el contenido de los textos de cara a poder trabajar con ellos para algunas tareas específicas de PLN usando como caso concreto el modelado mediante bolsas de palabras.

Objetivos

- ▶ Definir qué es un modelo del lenguaje y para qué tareas de PLN puede ser útil.
- ▶ Explicar cómo poder construir un modelo de lenguaje basado en n-gramas, y cómo poder evaluar que funciona correctamente.
- ▶ Identificar las limitaciones del modelado del lenguaje en base a n-gramas, y cómo se pueden solucionar algunas de ellas con técnicas de suavizado.
- ▶ Entender qué son los modelos de representación vectorial de los textos, para qué sirven, y cómo se pueden construir mediante el modelado con bolsas de palabras.

5.2. Introducción al modelo del lenguaje

Si se tiene una frase como:

- ▶ El alumno entregó al profesor...

Donde el final de ella no es conocido, se puede intentar predecir que podría seguir en base al conocimiento que se tiene *a priori* del lenguaje. Así, una persona que leyese esto podría pensar que esa frase podría estar seguida de cosas como «el examen» o «el ejercicio» más que de otras como «las llaves del coche» o «la ropa de deporte». Esto no quiere decir que estas últimas opciones no puedan aparecer en un determinado texto, sino que cuando no se tiene más información, en muchas ocasiones se acude a lo que es más probable.

Este mismo concepto es el que tratan de resolver los **modelos de lenguaje** (LM, Language Model): teniendo un **conocimiento estadístico de la lengua**, ven qué probabilidades hay en una secuencia de que continúe con un determinado elemento u otro. En general, con secuencia se hace referencia a secuencias de palabras, y con elementos a las palabras que la forman, de manera que se obtiene la probabilidad de que una determinada palabra siga a la secuencia previa de palabras

Un modelo de lenguaje es una distribución de probabilidad sobre las palabras de una secuencia, donde estas probabilidades se obtiene de entrenar el modelo en uno o varios corpus de la lengua.

Los LM son útiles para distintas tareas de PLN. Un ejemplo de ello es en el caso del **reconocimiento de voz**. Por ejemplo, supongamos el caso de un sistema de reconocimiento de voz que recibe un mensaje que puede corresponder a las siguientes frases:

- ▶ Pedro tiene mucho pelo y se quiere depilar-
- ▶ Pedro tiene mucho pelo y se quiere de Pilar-

En estos casos aparece una **ambigüedad fonética**, de manera que el sistema de reconocimiento de voz puede convertir ese audio en dos construcciones de texto distintas. Ahora bien, gracias a los LM, es más probable que «depilar» siga a la secuencia de texto previa que de «Pilar».

Los LM también son útiles para tareas como la traducción automática. Supongamos que queremos traducir la siguiente frase de inglés a castellano:

- ▶ *He is my friend Stephen, let me introduce him to you.*

El verbo «introduce» puede hacer referencia a presentar o a «introducir», de manera que dos posibles traducciones serían:

- ▶ Él es mi amigo Stephen, déjame que te lo presente.
- ▶ Él es mi amigo Stephen, déjame que te lo introduzca.

Con el conocimiento que se tiene en el LM, se puede detectar que dada la secuencia previa a la palabra introduce, es más probable que se tenga la primera secuencia que la segunda. No sólo eso, sino que como se puede ver, la secuencia de palabras traducida al castellano no sigue el orden textual de la secuencia de palabras en inglés. De esta manera, es más probable tener una construcción en castellano como «Él es mi amigo Stephen, déjame que te lo presente» que una más textual como «Él es mi amigo Stephen, déjame que lo presente a ti».

5.3. Modelos estadísticos del lenguaje basados en N-gramas

Supongamos que se tiene una frase como:

- ▶ La playa está tan tranquila.

Y se quiere obtener la probabilidad de que la palabra «que» siga a esa secuencia. Esto, a nivel formal, se expresaría como la probabilidad de tener «que» condicionada a tener la secuencia «la playa está tan tranquila»:

$$P(\text{que} \mid \text{la playa está tan tranquila})$$

El cálculo de esta probabilidad se puede obtener de manera similar a cómo se obtenía la **probabilidad de transición** para el caso del POS tagging:

$$P(\text{que} \mid \text{la playa está tan tranquila}) = \frac{C(\text{la playa está tan tranquila que})}{C(\text{la playa está tan tranquila})}$$

Es decir, que la probabilidad se obtiene como el número de veces que aparece que después de esa secuencia dividido entre el número de veces que se tiene esa secuencia. Estos valores se obtienen en base a un análisis previo de uno o varios corpus de la lengua.

No obstante, si se consideran secuencias muy largas no será habitual encontrarlas dentro del corpus. Las lenguas son a menudo muy flexibles, y pueden expresar un mismo tema con palabras distintas o con órdenes de la secuencia distintos. La misma frase de antes, si fuese «la playa estuvo tan tranquila que» sería una secuencia distinta a todos los efectos bajo esta aproximación.

Podríamos también calcular la probabilidad de tener esa secuencia mediante el uso de la **regla de la cadena**, donde la probabilidad de cada palabra de la secuencia se va obteniendo de manera recursiva en función de las probabilidades condicionadas previas:

$$P(w_i:n) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2}) \dots P(w_n|w_{1:n-1}) = \prod_{k=1}^n P(w_k|w_{1:k-1})$$

La cuestión es que a medida que vaya creciendo la secuencia, esta expresión irá también creciendo. Sin embargo, en muchas ocasiones es suficiente con definir un tamaño de histórico previo concreto y calcular las probabilidades sólo dentro de esa subsecuencia. Esto es lo que se hace con el **modelo de n-gramas**, donde la

probabilidad condicionada de una determinada palabra se calcula sólo con respecto a las **n palabras previas**. Un caso concreto es el modelo de bigramas, donde la probabilidad de una palabra dada la secuencia previa se aproxima a la probabilidad de tener esa palabra dada únicamente la palabra anterior:

$$P(w_k|w_{1:k-1}) \sim P(w_k|w_{k-1})$$

El caso de los bigramas está relacionado con la **hipótesis de Markov**, ya vista en el tema de POS tagging, donde se trabaja con modelos probabilísticos de Markov que asumen que la probabilidad de obtener un elemento concreto de una secuencia sólo depende del elemento inmediatamente anterior. Por tanto, la probabilidad de tener toda una secuencia concreta usando bigramas sería:

$$P(w_i:n) \sim \prod_{k=1}^n P(w_k|w_{k-1})$$

Este concepto de bigramas se puede generalizar a otros n-gramas, como los trigramas, con los que ya no se considera sólo la palabra previa, sino que se amplía para considerar las dos palabras anteriores:

$$P(w_k|w_{1:k-1}) \sim P(w_k|w_{k-2}:w_{k-1})$$

Así, esta expresión se generalizaría para un n-grama de tamaño N de la siguiente manera:

$$P(w_k|w_{1:k-1}) \sim P(w_k|w_{k-N+1}:w_{k-1})$$

Las probabilidades condicionadas se calculan utilizando el **método de máxima verosimilitud**, igual que se hizo en el tema de *POS tagging*, dividiendo el número de veces que aparece la secuencia previa considerada más la palabra en concreto entre

el número de veces totales que aparece esa secuencia. A modo de ejemplo, para el caso de bigramas sería:

$$P(w_k | w_{k-1}) = \frac{C(w_{k-1}, w_k)}{C(w_{k-1})}$$

Y generalizándola para un caso de un n-grama cualquiera:

$$P(w_k | w_{k-N+1:k-1}) = \frac{C(w_{k-N+1:k-1}, w_k)}{C(w_{k-N+1:k-1})}$$

Un modelo de lenguaje basado en n-gramas predice la probabilidad de que tenga lugar un determinado n-grama dentro de una secuencia de palabras de la lengua.

Evaluación de los modelos: perplejidad

De cara a la evaluación de los LM se plantean dos opciones. En primer lugar, se puede evaluar cómo funcionan a nivel de la aplicación de PLN para la que se estén utilizando. Por ejemplo, si se usa el LM dentro de un sistema de reconocimiento de voz, se podría evaluar si el rendimiento mejora al utilizar dicho LM. Este tipo de evaluaciones se denomina **evaluación extrínseca**. La parte negativa es que el coste de ejecutar todo un sistema de PLN para hacer evaluaciones puede ser muy costoso.

Una evaluación extrínseca de un modelo de lenguaje consiste en evaluarlo desde el análisis de la salida que tiene la aplicación de PLN que lo usa.

Por este motivo, existen también otro tipo de evaluaciones denominadas **intrínsecas**, en las que se evalúan los resultados del LM con respecto a un conjunto de datos de referencia. Este conjunto de datos de referencia se puede obtener directamente del corpus que se fuese a usar para entrenar el LM, haciendo una separación en datos de **entrenamiento/test** como se hace con los modelos de aprendizaje automático.

El conjunto de datos de entrenamiento se usaría para ajustar las probabilidades de las secuencias en el LM, y el conjunto de datos de test para su evaluación. Comparando dos LM, darían mejores resultados el que diese una **mayor probabilidad** a las secuencias correctas. El conjunto de datos original también se podría separar en **entrenamiento/validación/test** si se quiere tener un subconjunto para hacer validaciones iniciales y ajustar parámetros (por ejemplo, decidir si usar bigramas o trigramas para construir el LM).

Una evaluación intrínseca de un modelo de lenguaje consiste en evaluar sus resultados con respecto a un conjunto de datos de referencia (test).

La evaluación intrínseca en base a la máxima probabilidad se hace mediante el cálculo de una métrica denominada **perplejidad** (*perplexity*, PP). Esta métrica calcula la probabilidad inversa sobre los datos de test, normalizada en función del número de palabras. Por ejemplo, para un conjunto de palabras $W = w_1, w_2, \dots, w_N$ sería:

$$PP(W) = P(w_1, w_2, \dots, w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, \dots, w_N)}}$$

Aplicando la regla de la cadena para expandir el denominador de las probabilidades se tiene:

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_1, \dots, w_{i-1})}}$$

A modo de ejemplo cuando se trabaja con bigramas, la fórmula sería:

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$

Ya que la métrica PP se calcula en base al valor inverso de las probabilidades, cuanto mayor probabilidad tenga la secuencia, menor será el valor de PP. **Por este motivo, el LM será mejor cuanto más bajo sea el valor de PP.**

En el vídeo *Modelos estadísticos del lenguaje* se verá qué es un modelo de lenguaje y cómo se pueden construir utilizando técnicas estadísticas.



Accede al vídeo

Generalización y dispersión en los modelos

Como cualquier modelo estadístico, los LM basados en n-gramas dependen en gran medida del conjunto de datos usado para el entrenamiento, ya que las probabilidades de las subsecuencias estarán calculadas en base a lo que se ha visto en ese subconjunto de datos. Por este motivo, es importante tener en cuenta el tipo de texto (o textos) que se han usado como corpus de entrenamiento del modelo.

Si, por ejemplo, el LM está generado sobre textos de Shakespeare, las secuencias que se podrán generar con ese LM serán distintas de las que se podrían generar con un LM entrenado sobre textos de noticias, a pesar de que haya n-gramas que puedan ocurrir en ambos conjuntos de textos.

Así, el género de los textos es importante y se debe tener en cuenta cuando se crean LM o se usa un LM ya existente.

No sólo eso, sino que también ocurrirá que los n-gramas de noticias obtenidos a través de *tweets* serán también distintos respecto a los que se podrían obtener de las noticias de un periódico.

Por lo que, además del género, hay que tener en cuenta la variedad o el dialecto de los textos utilizados.

No obstante, aun considerando un mismo género y variedad/dialecto, el LM estará siempre limitado al conjunto de datos de entrenamiento, de manera que puede que haya secuencias válidas en el lenguaje que nunca se generarían por no aparecer en esos datos de entrenamiento. Este es el problema de la dispersión, y se acrecienta cuanto más grandes son los n-gramas, ya que se limitarán más las posibles secuencias que se pueden generar con el LM.

Cuanto mayor sea el n-grama, las secuencias que se pueden generar serán más fieles al texto original, pero precisamente por esto, será también más fácil que haya secuencias válidas que no estén contempladas en el LM.

1 gram	-To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have -Hill he late speaks; or! a more to leg less first you enter
2 gram	-Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. -What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	-Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. -This shall forbid it should be branded, if renown made it empty.
4 gram	-King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; -It cannot be but so.

Figura 1. Ejemplo de secuencias generadas con un LM entrenado en textos de Shakespeare. Fuente: Jurafsky y Martin, 2020.

Cuanto mayor es el n-grama, los textos son más coherentes y fieles a la fuente original. Sin embargo, también será más fácil que haya secuencias válidas en el lenguaje que no estén recogidas en el LM.

El problema de las secuencias válidas no contempladas en los datos de entrenamiento tiene también impacto en el cálculo de la PP, ya que, si hay secuencias en los datos de test que no están en los de entrenamiento, no se podría calcular como tal la PP con la fórmula vista previamente ya que se estaría dividiendo por 0.

Palabras desconocidas

Además de las problemáticas anteriores, un LM va a estar limitado a las palabras que haya en el conjunto de datos de entrenamiento, de manera que se tiene que ver cómo lidiar con palabras que no estén en él.

Estas palabras se denominan palabras desconocidas (*unknown words*) o palabras fuera del vocabulario (*out of vocabulary, OOV*). En relación a ellas, una métrica de evaluación adicional consiste en calcular el *ratio de OOV*, que se obtiene calculando el *porcentaje de palabras del conjunto de test que son OOV*.

La forma de trabajar con ellas en un LM es mediante el uso de un token genérico denominado **UNK** (*unknown*), que sirve para considerar a través de él todas las palabras OOV. Los pasos para utilizarlo son:

1. Usar una lista inicial de palabras fija.
2. Antes de entrenar el LM, convertir todas las palabras del conjunto de datos de entrenamiento que no están en esa lista fija en el token UNK.
3. Obtener las probabilidades del token UNK como si fuese cualquier otro token, de manera que con ello se modelaría la aparición de palabras desconocidas.

Si no se tuviese una lista inicial de palabras, esta se podría obtener directamente desde el conjunto de datos de entrenamiento, reemplazando algunas de las palabras por ese token UNK (por ejemplo, reemplazando las que tengan una baja frecuencia en el corpus por aparecer poco; por ejemplo, las que aparezcan menos de n veces).

5.4. Técnicas de suavizado

Además de palabras que sean OOV, ocurre también que pueden aparecer palabras en el conjunto de datos de test que, aunque existen en el de entrenamiento, aparecen en **contextos distintos**. Es decir, que aparecen, por ejemplo, tenemos «gran perro» en el conjunto de datos de test cuando en el de entrenamiento sólo se tenía «pequeño perro». Para evitar que se dé directamente un valor de probabilidad 0 a estos eventos (que es lo que ocurriría con las técnicas vistas hasta ahora), **se aplican técnicas denominadas de suavizado (smoothing)**, con las que se busca **asignar un pequeño valor de probabilidad para esos eventos no vistos**, de manera que sus probabilidades no sean 0 y, así, sea posible generar esas secuencias con el LM.

Suavizado de Laplace

Una de las técnicas más sencillas para hacer suavizado es el **suavizado de Laplace**, donde **se analizan todas las combinaciones posibles de n-gramas que se pueden dar con los datos de entrenamiento (no sólo las que ocurren como tal)**. Con ello, **se aumentan en 1 todas las ocurrencias de n-gramas, de manera que los que no ocurren en los datos de entrenamiento (y que tendrían, por tanto, una probabilidad de 0 al dar 0 la cuenta de cuantas veces ocurre) tendrían una probabilidad por defecto distinta de 0, aunque muy baja**.

Esto se ve a modo de ejemplo en la Figura 2 y 3, donde se tiene una matriz con los bigramas posibles que se obtendrían sobre un corpus de entrenamiento de ejemplo,

con un valor 0 en los bigramas que no ocurren. Con el suavizado de Laplace se incrementan todos los valores en 1, con lo que los valores que eran 0 pasan a valer 1.

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figura 2. Ejemplo de combinaciones de n-gramas (bigramas) antes (arriba) y después (abajo) de aplicar el suavizado de Laplace. Fuente: Jurafsky y Martin, 2020.

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Figura 3. Ejemplo de combinaciones de n-gramas (bigramas) antes (arriba) y después (abajo) de aplicar el suavizado de Laplace (continuación). Fuente: Jurafsky y Martin, 2020.

Como se puede ver, todas las probabilidades que eran 0 porque no se daban esos bigramas, ahora tienen un valor de 1. Las demás probabilidades también aumentan en 1.

Una vez se tiene esto se recalcularían las probabilidades de los n-gramas. A modo de ejemplo, para el caso de unigramas sería:

$$P(w_i) = \frac{c_i}{N} \rightarrow P_{Laplace}(w_i) = \frac{c_i + 1}{N + V}$$

De esta manera, para cada palabra se incrementaría en 1 el valor del número de veces que aparece, y se dividiría entre la suma del número de veces que aparecen todas las palabras originalmente (N) más el número de palabras que hay en el vocabulario (V), ya que se está incrementando en 1 el número de veces que aparece cada una de ellas.

De manera análoga, para el caso de bigramas sería:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})} \rightarrow$$

$$P_{Laplace}(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n) + 1}{\sum_w (C(w_{n-1}, w) + 1)} = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + V}$$

A pesar de que con este algoritmo simplemente, se incrementan en 1 el número de veces que aparecen todas las combinaciones de n-gramas en el corpus, la consecuencia es que debido a que esto afecta a todas las probabilidades (no sólo a las de los n-gramas que no aparecen) la contribución de los n-gramas que sí existen en el corpus para el cálculo de las probabilidades finales sea menor. Por ejemplo, para el caso de un unígrafo:

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V} = \frac{c_i^*}{N} \rightarrow c_i^* = (c_i + 1) \frac{N}{N + V}$$

De esta manera se puede ver cómo va a cambiar el peso que tiene un determinado n-grama en el cálculo de las probabilidades finales. Así, al aplicar la técnica de suavizado, es como si cambiase el número de veces que aparecía el n-grama originalmente (c_i) para pasar tener otro valor (c_i^*).

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Figura 4. Ejemplo del valor reconstruido de c_i (c_i^*) tras haber usado el suavizado de Laplace. Fuente: Jurafsky y Martin, 2020.

Propiamente, con las técnicas de suavizado va a tener lugar un descuento en el valor que se va a usar en el cálculo de las probabilidades de un determinado n-grama.

Este descuento se expresa en la siguiente fórmula.

$$d_c = \frac{c^*}{c}$$

Esto se puede extender para el caso de otros n-gramas. A modo de ejemplo y para un bigrama sería:

$$c^*(w_{n-1}, w_n) = \frac{|C(w_{n-1}, w_n) + 1| \times C(w_{n-1})}{C(w_{n-1}) + V}$$

Suavizado Add-k

La idea del suavizado de Laplace se puede generalizar para que, en lugar de incrementar el valor del número de veces que aparece una palabra en 1, se incremente en un valor k que se especifique. Esto es lo que se hace con el algoritmo **suavizado add-k (add-k smoothing)**. Para el caso de bigramas, sería de la siguiente manera:

$$P_{Add-k}(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n) + k}{C(w_{n-1}) + kV}$$

En este algoritmo se tiene que especificar a priori ese valor de k . Esto es algo que también se puede considerar como un **hiperparámetro** a definir tras seguir un proceso de optimización de los resultados sobre los **datos de validación**.

Backoff e interpolación

Los algoritmos vistos hasta ahora ayudan a resolver el problema de las combinaciones de n-gramas con frecuencia 0. Ahora bien, otra consideración que se puede hacer es que cuando no se tengan casos de un determinado n-grama, se estime esa probabilidad con un n-grama más pequeño. Esto es lo que se lleva a cabo con el **algoritmo de backoff**.

Por ejemplo, trabajando con trigramas, si se quiere calcular $P(w_n|w_{n-2}w_{n-1})$ y no se tiene el trígramo $w_{n-2} w_{n-1} w_n$, se puede estimar esa probabilidad usando el bigrama $P(w_n|w_{n-1})$. De igual manera, si no se tienen casos para obtener $P(w_n|w_{n-1})$, se puede estimar con la del unígrama $P(w_n)$. Así, este algoritmo se va aplicando de manera recursiva sobre los n-gramas que no aparecen hasta encontrar uno de menor orden que sí esté.

El algoritmo de *backoff* se usa para estimar las probabilidades de los n-gramas que no aparecen en los datos de entrenamiento con las probabilidades de los n-gramas de menor orden.

Frente al algoritmo de *backoff*, existe la alternativa del **algoritmo de interpolación lineal**, con el que la probabilidad para un determinado n-grama se obtiene de hacer una suma ponderada de la probabilidad de ese n-grama junto con las de sus n-gramas de menor orden. Para un ejemplo de un trígrama sería:

$$P_{inter}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n) + \lambda_2 P(w_n | w_{n-1}) + \lambda_3 P(w_n | w_{n-2}w_{n-1})$$

Con $\sum_i \lambda_i = 1$

Estos valores λ_i se pueden considerar también como **hiperparámetros** a estimar frente a los **datos de validación**.

Los valores λ_i se pueden calcular teniendo en cuenta la información de contexto. Si para el cálculo de las probabilidades de un determinado n-grama se tienen muchos casos de n-gramas de menor orden en el corpus, esto quede reflejado en sus valores λ_i para que la probabilidad final sea mayor respecto a otros casos en los que se tengan menos casos de n-gramas de menor orden. Formalizando esto para el caso de los trigramas se tendría:

$$\begin{aligned} P_{inter}(w_n | w_{n-2}w_{n-1}) \\ = \lambda_1(w_{n-2:n-1})P(w_n) + \lambda_2(w_{n-2:n-1})P(w_n | w_{n-1}) \\ + \lambda_3(w_{n-2:n-1})P(w_n | w_{n-2}w_{n-1}) \end{aligned}$$

Una última variación del algoritmo de *backoff* es el algoritmo **Katz backoff**, donde se usan las probabilidades originales cuando ese n-grama existe en el corpus de entrenamiento, y en caso de que no exista, se busca el primer n-grama de menor orden para el que sí se tengan datos, ponderando ese valor en función de un parámetro (como en el caso de la interpolación) que dependerá de la información de contexto, como en el caso de la interpolación ya mencionado.

Suavizado de Kneser-Ney

Además de los algoritmos vistos previamente, existen soluciones más avanzadas y efectivas para hacer el suavizado, como es el caso del suavizado de Kneser-Key.

Para completar el estudio de esta sección puedes leer las **páginas 17-20** del Capítulo 3 del libro: Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall. Disponible en:

<https://web.stanford.edu/~jurafsky/slp3/3.pdf>

En el vídeo *Técnicas de suavizado* se verá qué son y cómo aplicar técnicas de suavizado para la construcción de modelos del lenguaje.



Accede al vídeo

5.5. Modelos de representación vectorial

Como se ha visto previamente, con los modelos de lenguaje se modela la probabilidad de que se tengan secuencias concretas de palabras en base a un corpus previo de entrenamiento, y esto luego es útil para distintas tareas de PLN. De esta manera, partiendo de los textos de un corpus, se hace un modelado estadístico para poder trabajar con esos textos a nivel computacional y usarlos dentro de esas aplicaciones de PLN.

Ahora bien, existen otras maneras de representar el contenido de un texto para poder trabajar con él a nivel computacional para llevar a cabo distintas tareas de NLP. Este es el caso de los **modelos de representación vectorial**, donde los textos se representan como vectores dentro de un espacio vectorial en base a la información que se extrae de las palabras que aparecen en ellos. Con esto se busca que dos textos similares tengan representaciones vectoriales también similares, por lo que esta aproximación es útil para tareas como la **recuperación de información**, donde se

puede realizar una consulta en formato texto y se busca encontrar documentos similares a ella:

$$d_j = (w_{1j}, w_{2j}, \dots, w_{tj})$$

$$q = (w_{1q}, w_{2q}, \dots, w_{nq})$$

Como podemos ver en las ecuaciones anteriores, cada documento d_j se representaría por un vector obtenido en base a las palabras que aparecen en el texto, y análogamente se representaría la consulta q . Al ser vectores dentro de un mismo espacio vectorial, los documentos con un contenido similar al de la consulta deberían tener vectores similares.

Los modelos de representación vectorial son representaciones algebraicas de los textos o documentos, de manera que cada uno de ellos se expresa mediante un vector que recoge su información. Con ello, documentos o textos similares tendrán representaciones también similares.

La cuestión es cómo construir esos vectores que modelen el contenido de los textos. Existen muchas aproximaciones para hacerlo, y en este tema presentaremos algunas de las más sencillas. Ahora bien, hoy en día existen representaciones vectoriales más complejas basadas en redes neuronales, que se verán en detalle en el siguiente capítulo.

Una de las aproximaciones más sencillas para representar vectorialmente el contenido de texto es mediante lo que se denomina **bolsa de palabras (Bag of Words, BoW)**. Con esta aproximación, se consideran las palabras del texto de manera aislada (unigramas), sin considerar información en relación con el orden de estas en el texto. De esta manera, partiendo de un corpus de varios textos, esta técnica daría como salida una matriz donde cada fila representa uno de esos textos, y cada columna representa cada una de las palabras que aparecen en el corpus.

A modo de ejemplo, si se tiene un corpus con los siguientes textos:

$$\begin{cases} d_1 = \text{La ingeniera ha viajado en verano} \\ d_2 = \text{El abogado ha trabajado en verano} \end{cases}$$

Tras aplicar las técnicas de normalización (tokenización, eliminado de *stopwords*, eliminado de mayúsculas y lematizado) quedaría:

	ingeniero	viajar	verano	abogado	trabajar
d_1	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
d_2	x_{21}	x_{22}	x_{23}	x_{24}	x_{25}

Cada documento, representado por un vector con varias componentes, se puede ver también desde la perspectiva de las matrices de rasgos que se usan en aprendizaje automático para entrenar los modelos. Así, cada componente sería un rasgo o variable, pudiéndose usar directamente estas matrices para entrenar los modelos. Esto puede utilizarse para, por ejemplo, tareas de análisis de sentimiento, donde las variables de entrada de un modelo supervisado son precisamente los elementos de la matriz BOW, y la variable de salida sería la categoría del sentimiento (identificada previamente) asociada a cada uno de esos documentos. Con estas representaciones se tiene, por tanto, un vector por texto de una dimensionalidad N donde N es el número de palabras del vocabulario del corpus. Por este motivo es importante también normalizar antes los textos de cara a reducir la dimensionalidad de los vectores.

Dentro del esquema de BoW, los valores que aparecen en la matriz se pueden calcular de distinta manera. Existen dos aproximaciones, las **locales** y las **globales**. En el caso de las locales, los valores de las componentes del vector que representa cada texto se calculan en base a sólo la información de ese texto. En el caso de las representaciones globales, las componentes del vector se construyen usando información tanto del texto en sí como del resto de textos del corpus.

Representaciones locales

Una de las representaciones más sencillas dentro de las representaciones locales es la **binaria**, donde cada componente toma un valor binario (1 o 0) dependiendo de si esa palabra aparece o no en el texto en cuestión.

$$Bin(t_i, d_j) = \begin{cases} 1 & \text{si el token } t_i \text{ aparece en el documento } d_j \\ 0 & \text{en otro caso} \end{cases}$$

Uno de los problemas de las representaciones binarias es que se ponderan por igual todas las palabras, independientemente de que aparezcan mucho o poco en un determinado texto. Por este motivo, existen otras representaciones, como las de **frecuencia de términos** (Term Frequency, TF) donde en lugar de usar valores binarios, se cuenta el número de veces que aparece la palabra en el texto en cuestión.

$$TF(t_i, d_j) = f_{ij}, \text{ siendo la frecuencia del token } t_i \text{ en } d_j$$

Ahora bien, la representación en base a frecuencias de términos también tiene sus problemas, ya que puede que no todos los textos tengan el mismo número de palabras, y por tanto, se dé más importancia a los textos más largos frente a los cortos. Para solucionar eso, se usa también la técnica **frecuencia de términos ponderada** (Weighted Term Frequency, WTF) donde se normalizan los valores de las frecuencias en base a la suma de las frecuencias dentro del documento.

$$WTF(t_i, d_j) = \frac{f_{ij}}{\sum_{t_p \in d_j} f_{pj}}$$

Representaciones globales

Junto a las representaciones locales aparecen también las **globales**, donde la información de los componentes del vector usa sólo información del propio texto, se tienen también las globales donde se usa información de todo el corpus. Un

ejemplo es la **Frecuencia de términos-Frecuencia inversa del documento** (Term Frequency–Inverse Document Frequency, TF-IDF).

Una de las utilidades de construir las componentes de los vectores con TF-IDF es la siguiente. Con las técnicas vistas previamente, como **TF**, se puede acabar dando **importancia a palabras poco representativas que aparecen mucho en cualquier tipo de texto**. Es verdad que esto ocurre principalmente con palabras como determinantes, conjunciones que se eliminarían al quitar las *stopwords*. Sin embargo, puede haber otras palabras que no sean *stopwords* y que no sean representativas por aparecer mucho. Por ejemplo, teniendo textos de sinopsis de películas, la palabra «película» podría aparecer mucho en todos ellos, pero no es representativa para modelar el contenido de esos textos. Esto precisamente se soluciona con TF-IDF, donde el valor de la frecuencia (TF) se pondera en función de la rareza de las palabras dentro del corpus de la siguiente manera:

$$TFIDF(t_i, d_j) = f_{ij} \times \log\left(\frac{N}{df(t_i)}\right)$$

Donde f_{ij} es la frecuencia del token t_i en el documento (o texto) d_j , N el número de documentos en el corpus, y $df(t_i)$ el número de documentos en el corpus donde aparece el token t_i .

En el vídeo *Modelos de representación vectorial* se revisarán las distintas técnicas de representación de textos basadas en modelos vectoriales.



Accede al vídeo

4.6. Referencias bibliográficas

Aggarwal, C. (2018). *Machine Learning for Text*. Springer Cham.

Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. New Jersey (Estados Unidos): Prentice-Hall.

Manning, C. y Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Modelos de Lenguaje

Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall. <https://web.stanford.edu/~jurafsky/slp3/14.pdf>

El capítulo 3 describe en detalle todo lo relacionado con el modelado de lenguaje mediante n-gramas, junto con las técnicas de suavizado que se han visto en este tema además de alguna otra adicional.

Test

1. Indica cuál de estas afirmaciones es verdadera sobre los modelos de lenguaje:

 - A. Se pueden usar en aplicaciones de PLN como por ejemplo para reconocimiento de voz o traducción automática.
 - B. No necesitan ningún dato de partida, sino que generan las secuencias más probables sin ningún conocimiento previo de la lengua.
 - C. Permiten modelar sólo secuencias de texto cortas, pero no largas.
 - D. Todas las anteriores.
2. Si el número de veces que aparece «el gato» en un corpus es 100, y el número de veces que aparece «el gato es grande» es 20, la probabilidad condicionada para la palabra «grande» dada la secuencia previa «el gato es» es

 - A. 0.1.
 - B. 0.2.
 - C. 0.3.
 - D. 0.4.
3. Indica cuál de estas afirmaciones es verdadera sobre la evaluación de los modelos del lenguaje:

 - A. Un ejemplo de evaluación extrínseca es la métrica de perplejidad.
 - B. Un ejemplo de evaluación intrínseca es el análisis del modelo del lenguaje desde los resultados que tiene al usarse dentro de una aplicación de PLN, como por ejemplo una de traducción automática.
 - C. Cuanto más alto sea el valor de perplejidad, mejor será el modelo de lenguaje.
 - D. Ninguna de las anteriores.

- 4.** Indica cuál de estas afirmaciones es verdadera sobre los modelos de lenguaje:
- A. Si se usan n-gramas de un orden alto (ej.: trigramas o superior), es más difícil que el modelo genere secuencias de texto fieles a las del corpus original.
 - B. Si se usan n-gramas de bajo orden (ej. unigramas), es más fácil que el modelo genere secuencias de texto fieles a las del corpus original.
 - C. Si se usan n-gramas de bajo orden (ej. unigramas), es más difícil que el modelo genere secuencias de texto fieles a las del corpus original.
 - D. Ninguna de las anteriores.
- 5.** ¿Qué pasos se pueden seguir para tener un modelo de lenguaje que sea capaz de trabajar con palabras desconocidas?
- A. 1) Usar una lista de palabras abierta que pueda cambiar. 2) Convertir las palabras del conjunto de test que no estén en la lista a UNK. 3) Obtener las probabilidades de esos tokens UNK.
 - B. 1) Usar una lista de palabras cerrada. 2) Convertir las palabras del conjunto de entrenamiento que no estén en la lista a UNK. 3) Obtener las probabilidades de esos tokens UNK.
 - C. 1) Usar una lista de palabras cerrada. 2) Convertir las palabras del conjunto de test que no estén en la lista a UNK. 3) Obtener las probabilidades de esos tokens UNK.
 - D. Ninguna de las anteriores.

6. Dado un vocabulario de 250 palabras, si la palabra «buen» aparece cien veces, y si la secuencia «buen amigo» aparece veinte veces, ¿cuál es la probabilidad $P(\text{amigo} | \text{buen})$ considerando el suavizado de Laplace?
- A. 0.04.
 - B. 0.05.
 - C. 0.06.
 - D. 0.08.
7. Dado un vocabulario de 250 palabras, si la palabra «buen» aparece 100 veces, y si la secuencia «buen amigo» aparece veinte veces, ¿cuál es la probabilidad $P(\text{amigo} | \text{buen})$ considerando el suavizado add-k con $k=5$?
- A. 0.001.
 - B. 0.019.
 - C. 0.051.
 - D. 0.133.
8. Dado un vocabulario de 250 palabras, con una suma de frecuencias totales de 10 000, si la palabra «buen» aparece cien veces, y si no se tiene la secuencia «buen amigo», ¿qué probabilidad tendría asignado ese bigrama usando el suavizado de *backoff*?
- A. 0.01.
 - B. 0.02.
 - C. 0.3.
 - D. 0.4.

9. Indica cuál de estas afirmaciones es verdadera sobre los modelos de representación vectorial:

- A. Son representaciones algebraicas de textos o documentos.
- B. Representan los textos o documentos en un espacio vectorial, de manera que dos textos similares tengan representaciones similares.
- C. A y B son correctas.
- D. Ninguna de las anteriores.

10. Indica cuál de estas afirmaciones es verdadera respecto a los modelos de representación vectorial basados en BoW:

- A. Se caracterizan por conservar la información relativa al orden de las palabras de los textos.
- B. Se caracterizan por conservar la información relativa a la sintaxis de los textos
- C. Se caracterizan por usar sólo representaciones binarias para indicar las palabras que aparecen en un determinado texto.
- D. Ninguna de las anteriores.

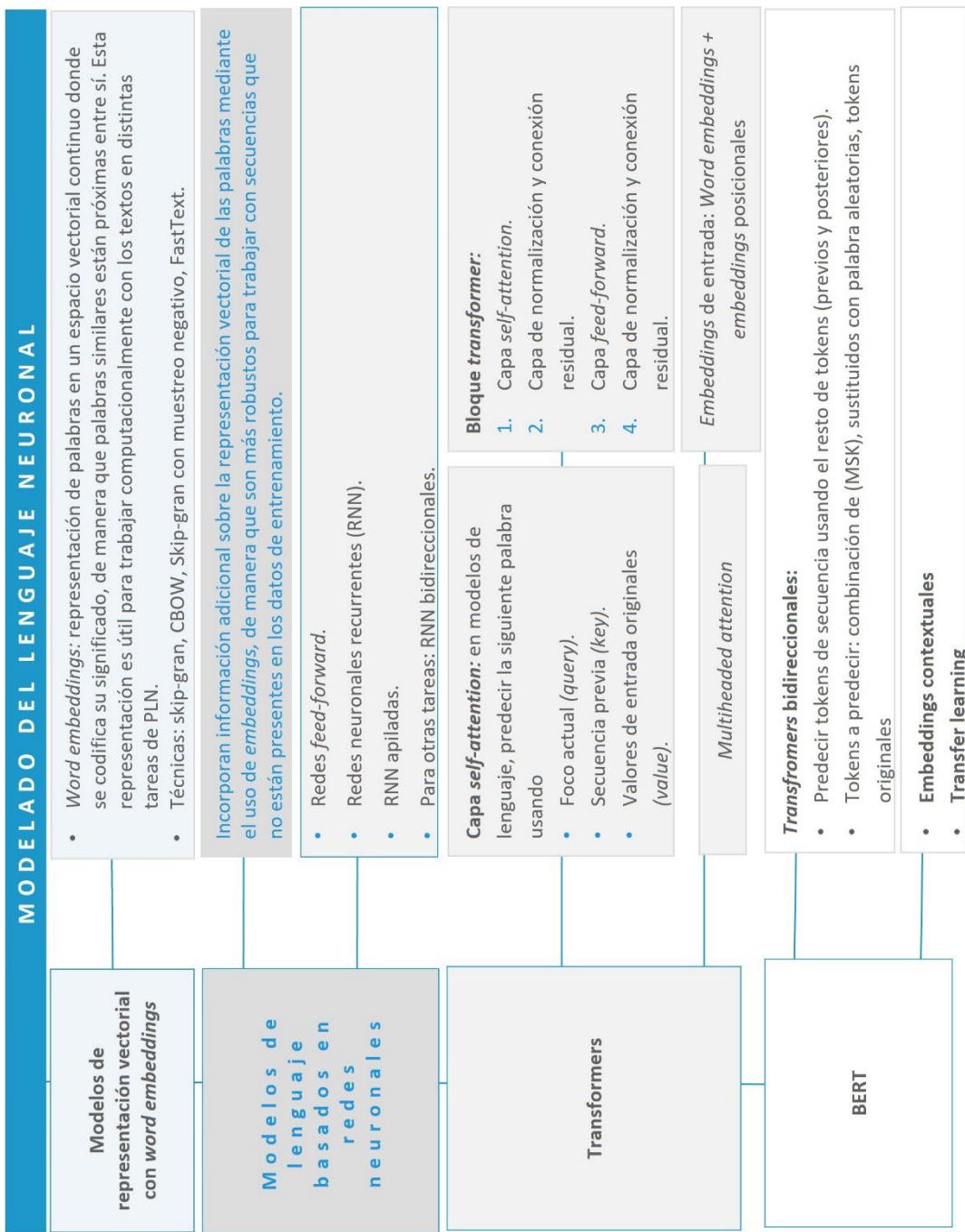
Procesamiento del Lenguaje Natural

Modelado del Lenguaje Neuronal

Índice

Esquema	3
Ideas clave	4
6.1. Introducción y objetivos	4
6.2. Modelos de representación vectorial con <i>word embeddings</i>	5
6.3. Modelos de lenguaje basados en redes neuronales	18
6.4. Transformers	28
6.5. BERT	37
6.6. Referencias bibliográficas	43
A fondo	44
Test	45

Esquema



6.1. Introducción y objetivos

A continuación, se abordará el uso de redes neuronales para **construir la representación vectorial** del significado de las palabras mediante el uso de algoritmos como Skip-gram o CBOW. También, se verá el uso de redes neuronales para **abordar tareas de modelado del lenguaje**, como las que se vieron en el tema anterior. Finalmente, se presentará una arquitectura de redes neuronales muy relevante para el PLN, los *transformers*, con los que se pueden llevar a cabo tanto tareas de modelado del lenguaje, como de representación vectorial del significado de las palabras, entre muchas otras. En relación con esto, se hablará del modelo de BERT, y cómo se puede usar para obtener *embeddings* contextuales.

Objetivos

- ▶ Entender el concepto de *word embeddings*, y cómo poder construirlos usando arquitecturas basadas en redes neuronales.
- ▶ Describir cómo se pueden usar arquitecturas de redes neuronales para llevar a cabo tareas de modelado del lenguaje.
- ▶ Describir qué son los *transformers* en el ámbito del procesamiento del lenguaje natural, cómo es su arquitectura, y qué ventajas tienen.
- ▶ Identificar las características de un modelo como BERT, describir su arquitectura, y entender su uso para la obtención de *embeddings* contextuales.

6.2. Modelos de representación vectorial con *word embeddings*

Anteriormente, se estudió cómo los **modelos de representación vectorial** son **representaciones algebraicas de los textos o documentos**, de manera que cada uno de ellos se expresa mediante un vector que recoge su información, de modo que documentos o textos similares tendrán representaciones también similares. Dentro de las distintas técnicas que se pueden usar para construir estas representaciones, se comentó cómo funciona la técnica de **bolsa de palabras** (*Bag of Words, BoW*) con la que se obtenía un vector que representa el contenido de ese texto en base a información del texto en sí (representaciones locales), o del texto en sí junto con información del corpus (representaciones globales).

Ahora bien, un **problema** que se tiene con estas representaciones es **el aspecto de la dispersión (sparsity)**. Por ejemplo, si se tiene un corpus de $V=1000$ palabras, y se quiere construir el vector de un texto de diez palabras (sin considerar *stopwords*), se tendrá un vector con 990 componentes a 0 (correspondientes a todas las palabras del vocabulario que no aparecen en ese texto). Este es precisamente el **problema** de la dispersión, el **tener representaciones vectoriales** donde casi todas las componentes estén a 0.

El problema de la dispersión de las representaciones basadas en bolsas de palabras hace referencia a cómo se cómo los vectores que representan los textos tienen muchas de sus componentes a 0.

Además, **construir representaciones vectoriales** donde se tengan tantos componentes como palabras de un vocabulario puede dar lugar a **espacios vectoriales de mucha dimensión**, lo que se traduciría en tener un gran número de variables. Esto podría dificultar luego el entrenamiento de un modelo de aprendizaje automático (mayor coste de computación y mayor propensión a tener sobreajuste).

Adicionalmente, aunque las representaciones basadas en BoW pueden servir para construir el significado del texto en su conjunto, ¿cómo se podría hacer para representar el significado no sólo del texto sino también de las palabras que lo componen? Si bien se construyen espacios vectoriales donde dos textos similares tendrán vectores cercanos, el significado de las palabras no está contemplado dentro de estos espacios.

Word Embeddings

Para solventar el problema de la dispersión, el problema de la representación en espacios vectoriales de gran dimensionalidad, y el de la representación del significado de las palabras, se proponen las representaciones densas denominadas **word embeddings**, con las que se asigna un valor numérico a las palabras en base a su semántica. De esta manera, en lugar de representar los textos directamente en un espacio vectorial, se representan primero las palabras, de manera que cada una tendrá asociado un vector numérico que recoja su valor semántico en un espacio dimensional de un tamaño predefinido. Este valor semántico (su *embedding*) se puede obtener en función de la relación contextual que cada palabra tenga con el resto de las palabras del corpus. En la siguiente imagen se muestra el ejemplo clásico de representación con vectores en tres dimensiones. Gracias a que los valores numéricos se infieren por el contexto, se pueden hacer operaciones del siguiente tipo:

$$\text{King} - \text{Man} = \text{Queen} - \text{Woman}$$

Es decir, que los vectores representan el valor semántico de que si a «rey» se le resta la contribución de «hombre» y a «reina» de «mujer» se obtendría un mismo valor que representaría el valor semántico genérico de «monarca».

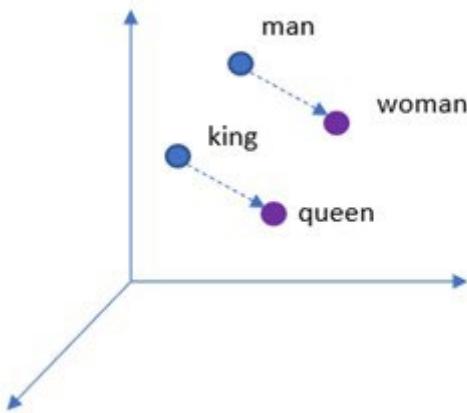


Figura 1 Ejemplo de representación con vectores en tres dimensiones. Fuente: elaboración propia.

Así, con esta aproximación, cada palabra se representa con un vector de dimensionalidad D , y cada texto se representa como una matriz $N \times D$, con N el número de palabras. Estos se pueden obtener mediante distintas técnicas, como **Skip-gram** o **CBOW (Continuous Bag of Words)**.

El término *word embeddings* dentro del PLN hace referencia a la representación de palabras en un espacio vectorial continuo donde se codifica su significado, de manera que palabras similares están próximas entre sí. Esta representación es útil para trabajar computacionalmente con los textos en distintas tareas de PLN.

Skip-gram

La obtención de estos *embeddings* mediante el algoritmo de Skip-gram consiste en entrenar un modelo que tenga como entrada las distintas palabras del vocabulario, y que trate de predecir el contexto de dichas palabras.

Este contexto se define en función de las **palabras vecinas** que tengan esas palabras de entrada. Las palabras vecinas son las que aparecen en una ventana de un tamaño concreto alrededor de cada una de las palabras del vocabulario.

El cliente ha comprado un coche rojo.

Figura 2. Ejemplo de ventana de tamaño 2 alrededor de la palabra «comprado». Fuente: elaboración propia.

Con ello, se define un conjunto de datos de entrenamiento para entrenar un modelo que trate de **predecir las palabras del contexto de cada palabra del vocabulario**.

(comprado, cliente)
(comprado, ha)
(comprado, un)
(comprado, coche)

Figura 3. Datos de entrenamiento expresados en tuplas con la palabra de entrada (azul) y las palabras del contexto (verde). Fuente: elaboración propia.

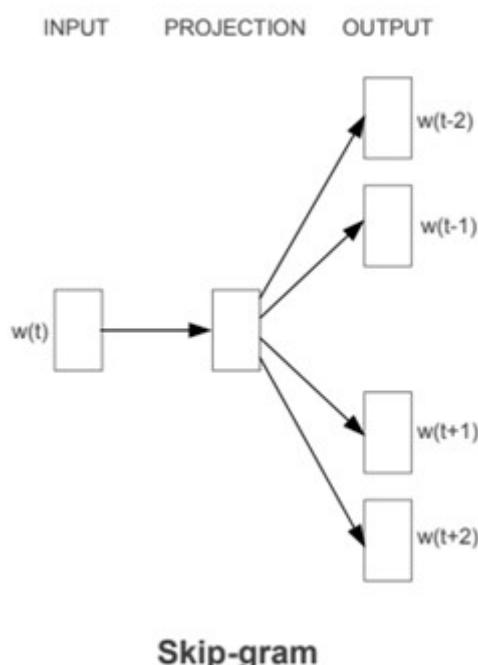


Figura 4. Esquema del algoritmo de Skip-gram para el caso de una ventana de tamaño 2. Fuente: Mikolov et al., 2013).

Así, formalmente, el algoritmo Skip-gram busca obtener lo siguiente:

$$P(w_1, w_2, \dots, w_m | w)$$

Donde w_1, w_2, \dots, w_m son las distintas palabras del contexto y w es la palabra de entrada. De esta manera, el modelo de Skip-gram para la obtención de *embeddings* es un modelo **auto-supervisado** (*self-supervised*), ya que es un modelo (dada la palabra de entrada se trata de predecir el contexto) donde los datos de entrenamiento no se obtienen de un proceso de anotación previo, sino que se obtienen automáticamente de los propios datos de entrada.

Una aproximación para construir el modelo de Skip-gram utilizando aprendizaje automático, es la siguiente:

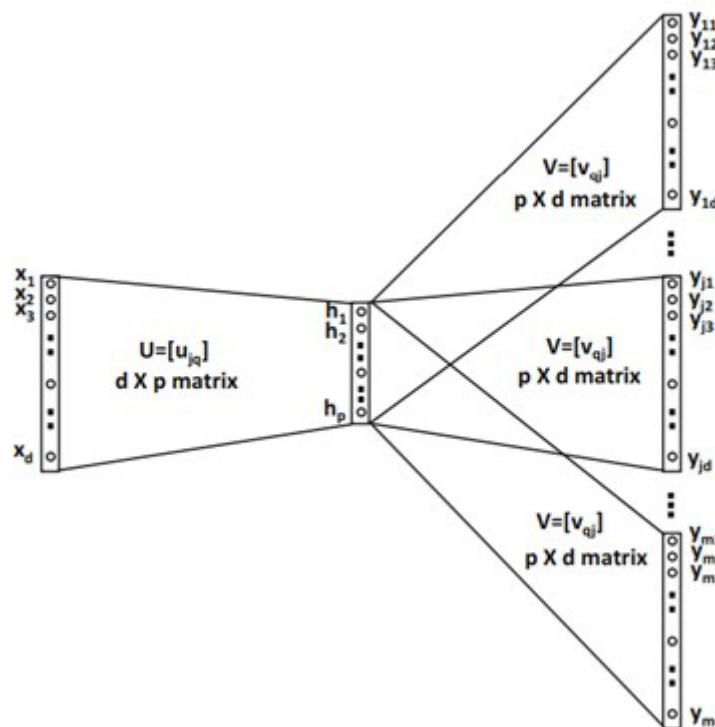


Figura 5. Arquitectura del modelo Skip-gram. Fuente: Mikolov et al., 2013.

Dada una palabra de entrada (w_q), se tiene una **red neuronal** con una capa intermedia con la que se trata de predecir si una palabra concreta w_j pertenece o no su contexto. La entrada del modelo, que corresponde a una palabra concreta del vocabulario (de tamaño d), se puede expresar mediante un vector *one-hot encoding*, donde todos los valores sean 0 menos el de la palabra en cuestión. De igual manera, el vector asociado a la salida (la palabra del contexto), también se podrá expresar mediante un vector *one-hot encoding*. Esta arquitectura tiene dos matrices de pesos

U y V, que serán compartidas para todas las combinaciones de palabras de entrada con respecto a sus respectivas m palabras de contexto, de manera que se tenga una única matriz U y una única matriz V al final del entrenamiento. La matriz U corresponde a la **matriz de embeddings**.

Es decir, que la representación vectorial del significado de las palabras se obtendrá directamente de los pesos de la red neuronal, recogidos en dicha matriz (también denominada *lookup table*).

El valor p corresponde precisamente a la dimensionalidad del espacio de representación de los vectores (es decir, al número de componentes que estos tengan). Este valor p es un parámetro que se define a priori.

El valor de salida de la capa oculta se obtendría de la siguiente manera:

$$h_q = \sum_{j=1}^d u_{jq}x_j \quad \forall q \in \{1 \dots p\}$$

Y, con ello, se podría obtener ya la salida del modelo. Ahora bien, como lo que se trata de hacer es predecir si una palabra está o no en el contexto de la palabra de entrada, lo que corresponde a un **problema de clasificación**, la capa de salida tiene incluida una función *softmax*. De esta manera, si la salida de la matriz V corresponde a un vector z, la predicción final quedaría como:

$$\hat{y} = p(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Donde se normaliza la salida para que su valor sea siempre positivo, entre 0 y 1, y corresponda a un valor de probabilidad tal que así la suma de todos los valores de salida sean 1. Esta salida es la que se compraría con el vector de *one-hot encoding* asociado a esa palabra del contexto. De esta manera, ambos se incluirían en una

función de coste (como, por ejemplo, la **entropía cruzada**) tal que se busque minimizar:

$$L = - \sum_{i=1}^m \sum_{j=1}^d y_{ij} \log (\hat{y}_{ij})$$

Donde el vector \hat{y} corresponde a la predicción, y el vector y al valor objetivo obtenido desde ese *one-hot encoding*.

Con esto, se entrenaría el modelo para todas las palabras del vocabulario y para todas las palabras de su contexto, ajustando los pesos mediante técnicas como, por ejemplo, las de **descenso de gradiente**.

Una vez que se tienen las dos matrices finales U y V , se puede obtener el *embedding* para una palabra del vocabulario d obteniendo su vector de entrada y multiplicándolo por la matriz de *embeddings* U (*lookup table*).

$$(1 \ 0 \ \dots \ 0) \times \begin{pmatrix} 0.4 & \dots & 2.1 \\ \vdots & \ddots & \vdots \\ -1.3 & \dots & 1.1 \end{pmatrix} = \begin{pmatrix} 0.4 \\ \vdots \\ 2.1 \end{pmatrix}$$

Figura 7. Ejemplo de obtención de los *embeddings* para una palabra de entrada (la primera del vocabulario) dada su *lookup table*. Fuente: elaboración propia.

En el vídeo *Word embeddings y wordvec* se verá cómo el aprendizaje profundo sirve para modelar el significado de las palabras dentro del PLN.



Accede al vídeo

CBOW

El esquema del modelo CBOW es el **inverso al del Skip-gram**. Como entrada se tienen las palabras del contexto de una determinada palabra, y como salida se tiene esa palabra. De esta manera, se plantea un modelo de clasificación que trate de predecir una palabra dado su contexto. Así, la entrada serán tuplas (como en Skip-gram) donde el primer elemento es una palabra del contexto y el segundo la palabra que se quiere predecir.

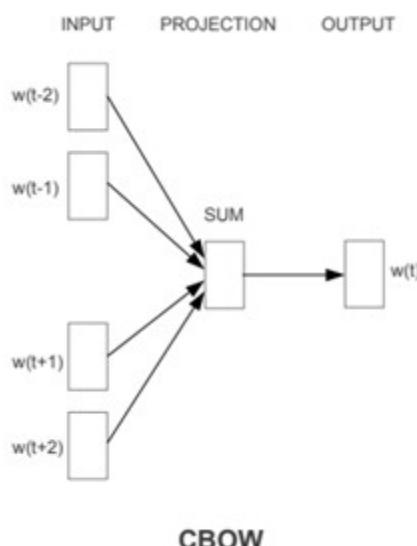


Figura 8: Esquema del algoritmo de CBOW para el caso de una ventana de tamaño 2. Fuente: Mikolov et al., 2013.

Formalmente, sería:

$$P(w | w_1, w_2, \dots, w_n)$$

De manera análoga a cómo era la arquitectura de Skip-gram, CBOW usa un modelo basado en redes neuronales donde se tiene una capa oculta. Así, se tienen dos matrices de pesos, U y V, entre la entrada y la capa oculta, y entre la capa oculta y la salida, respectivamente. Esto se ve en la siguiente imagen:

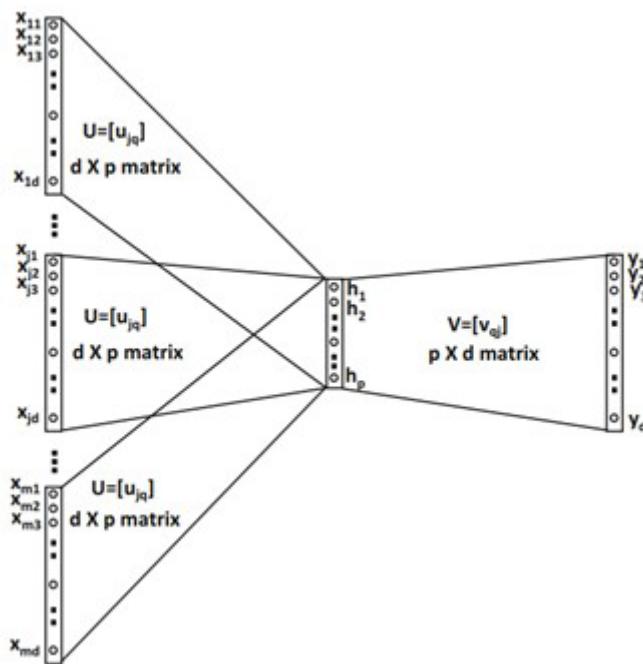


Figura 9. Arquitectura del modelo CBOW. Fuente: Mikolov et al., 2013.

Las palabras de entrada se pueden convertir en vectores binarios con una técnica como *one-hot encoding*, y lo mismo se podrá hacer sobre la palabra de salida. La capa de salida tendrá una función *softmax* para expresar dicha salida como una probabilidad. Igual que en el caso de Skip-gram, las matrices de pesos U y V son compartidas entre todas las combinaciones de palabras de contexto y palabras de salida. La matriz U final después del entrenamiento será la que se use como matriz de *embeddings*.

No obstante, en el modelo de CBOW no es necesario considerar de manera independiente las palabras de entrada del contexto, sino que se puede reducir significativamente el coste computacional considerando ese contexto como un único vector de entrada con el que se busca predecir una determinada palabra.

La manera de hacer esto es hacer *one-hot encoding* sobre todas las palabras del contexto, de manera que de la matriz de *embeddings* se obtengan los valores de los vectores asociados a todas ellas. Luego, sobre esta salida, se aplica una capa de *pooling* que agregue esos valores (por ejemplo, obteniendo el valor medio de esos vectores), y ya con ese valor agregado se obtenga la probabilidad de predicción de la palabra de salida. Un esquema de esta aproximación aparece en la imagen siguiente:

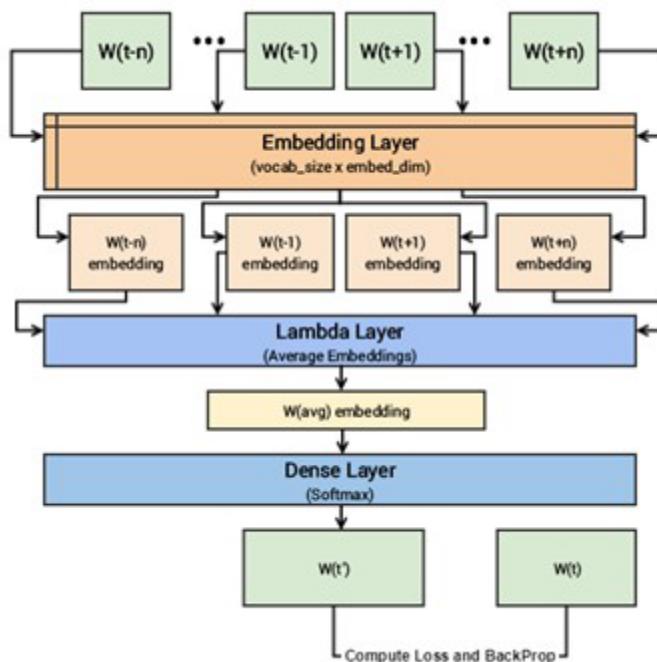


Figura 10. Arquitectura de CBOW donde se agrega el contexto de entrada en un único vector. Fuente: KD Nuggets, 2018.

Comparando Skip-gram con CBOW, aparecen algunas diferencias significativas que hacen que un modelo u otro sean mejores en distintos casos. Con el modelo de CBOW, si se tienen frases como «el agua del mar es clara» y «el agua del mar es prístina», en ambos casos se recibe un mismo contexto y se trata de predecir la palabra más probable para él.

Por este motivo, CBOW da buenos resultados para las representaciones de palabras comunes, pero no tanto para las palabras poco frecuentes, ya que ante un contexto se va a poner el énfasis en predecir las palabras que son más comunes.

Así, será más fácil en el ejemplo anterior representar clara que prística. En cambio, como con Skip-gram se recibe una palabra y se busca predecir su contexto, para los ejemplos anteriores se tratarían esas palabras, clara y prística, como palabras independientes, de manera que será más precisa la representación de prística que con CBOW. Así, Skip-gram da mejores resultados para la representación de palabras poco frecuentes.

Otra diferencia significativa entre ambos modelos es que es más rápido entrenar CBOW que Skip-gram. El motivo es el siguiente: para el caso del modelo con la capa de *pooling* de CBOW, hay menos combinaciones de entradas-salidas para entrenar el modelo.

Skip-gram con muestreo negativo

El modelo de Skip-gram visto previamente tienen una fase especialmente costosa a nivel computacional: el cálculo de la capa *softmax*. En esta capa se transforman las salidas de la red neuronal a un vector de probabilidades normalizado sobre todas las palabras del vocabulario, haciéndose este cálculo para todas las palabras de entrada sobre las que se quiere predecir el contexto. Por este motivo, existe una arquitectura de Skip-gram alternativa denominada **Skip-gram con muestreo negativo** (Skip-gram with negative sampling, SGNS).

Con la arquitectura SGNS, en vez de trabajar con todas las palabras del vocabulario en la predicción del contexto, se trabaja con un subconjunto de palabras formado por palabras que aparecen en el contexto de la palabra de entrada, y algunas palabras que no aparecen. Con esto se busca que el modelo sepa diferenciar qué palabras son del contexto y cuáles no. Para poder implementar esta arquitectura, se lleva a

cabo un cambio: en lugar de plantear una clasificación multiclase, se plantea una **arquitectura de clasificación binaria** usando una función de salida sigmoide, tal que, dada una palabra de entrada y una segunda palabra de entrada, se obtenga la predicción de si esa segunda palabra es del contexto de la primera o no. Esta idea se refleja en la siguiente arquitectura:

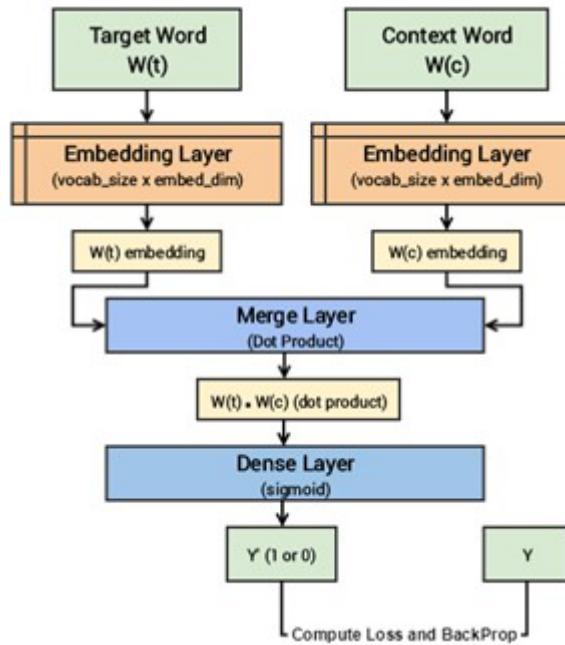


Figura 11. Ejemplo de arquitectura para SGNS. Fuente: KD Nuggets, 2018.

De esta manera, para P palabras del contexto como entrada y para N palabras obtenidas de un muestreo que no son del contexto, se obtienen P + N predicciones para una determinada palabra de entrada, con las que se calcula la siguiente función de coste que se busca optimizar:

$$-\sum_{(i,j) \in \mathcal{P}} \log \left(\frac{1}{1 + \exp(-\bar{u}_i \cdot \bar{v}_j)} \right) - \sum_{(i,j) \in \mathcal{N}} \log \left(\frac{1}{1 + \exp(\bar{u}_i \cdot \bar{v}_j)} \right)$$

Donde el valor de los sumatorios corresponde a la salida de las predicciones del modelo en función de valores de entrada. Así, con esta función se busca que esos valores P sean lo mayor posible (predicciones cercanas a 1) y los N lo más pequeños posibles (cercanos a 0).

FastText

Uno de los problemas que aparecen con las representaciones vistas hasta ahora de word2vec es **cómo representar palabras desconocidas**. La representación de las palabras se ha obtenido mediante el entrenamiento previo de una red neuronal sobre un corpus de entrenamiento, y una vez hecho esto, se usa la matriz de pesos para obtener esas representaciones.

Ahora bien, podría ocurrir que se quieran obtener representaciones de palabras que no aparecían en el corpus sin tener que reentrenar toda la red neuronal para ello.

Esto es lo que trata de solucionar el modelo de **FastText** (Bojanowski et al., 2017) mediante la construcción de representaciones vectoriales no sólo de las palabras, sino también de los distintos n-gramas de una misma palabra. Así, por ejemplo, la palabra «where» con n-gramas de tamaño 3 para sus caracteres se representaría como:

<wh, whe, her, ere, re>

Con ello, se obtienen los *embeddings* con técnicas como Skip-gram para cada uno de esos n-grama constituyentes, de manera que el *embedding* de la palabra se construye luego como la suma de todos esos *embeddings* de los constituyentes. Cuando se quieren representar palabras desconocidas, se representarán en base a la suma de los *embeddings* de los n-gramas constituyentes que sí aparecen en el corpus.

6.3. Modelos de lenguaje basados en redes neuronales

Las redes neuronales no son solo útiles para construir modelos de representación vectorial, como es el caso de la obtención de *word embeddings* mediante el uso de las técnicas vistas previamente. También son útiles para la construcción de **modelos de lenguaje** (*language model*, LM). Como ya se comentó en el tema previo, con un modelo del lenguaje se calculan las probabilidades de que desde una secuencia se continúe con un determinado elemento. Ahora bien, para el cálculo de esta probabilidad no se tienen porque considerar todos los elementos de la secuencia, sino que se puede aproximar considerándose únicamente los N últimos términos. Formalmente, esto se expresa de la siguiente manera:

$$P(w_t | w_1, \dots, w_{t-1}) \approx P(w_t | w_{t-N+1}, \dots, w_{t-1})$$

Para el caso de N = 4 sería:

$$P(w_t | w_{t-3}, w_{t-2}, w_{t-1})$$

En el tema previo se vio cómo era posible obtener estas probabilidades en base al uso de modelos estadísticos basados en n-gramas. Ahora bien, estos modelos tenían una serie de limitaciones como, por ejemplo, el cálculo de probabilidades cuando se tienen secuencias no vistas en el corpus de entrenamiento. Si, por ejemplo, se trabaja con N = 4 y en el corpus de entrenamiento se tiene una frase como

- ▶ *Tengo un perro de mascota.*

Cuando se tengan secuencias como «un perro de» se podrá predecir «mascota». Ahora bien, si aparecen frases nuevas como «tengo un gato de» no se sabrá qué palabra podrá seguir a esa secuencia. Este problema se soluciona en gran parte con

el uso de **modelos basados en redes neuronales** para la construcción de LM, ya que en ellos se incorpora el concepto de *embeddings* visto previamente, de manera que, aunque el LM no haya visto nunca la palabra «gato» en esa secuencia, a nivel de representación de *embeddings* será similar a «perro» y se podrá estimar que una palabra como «mascota» podría seguir a esa secuencia.

Los modelos de lenguaje basados en redes neuronales incorporan información adicional sobre la representación vectorial de las palabras mediante el uso de *embeddings*, de manera que son más robustos para trabajar con secuencias que no están presentes en los datos de entrenamiento.

Redes *feed-forward*

La construcción de un LM basado en redes neuronales se puede llevar a cabo mediante el uso de **distintas arquitecturas**. Una primera aproximación es mediante el uso de *redes feed-forward*, como refleja la imagen siguiente.

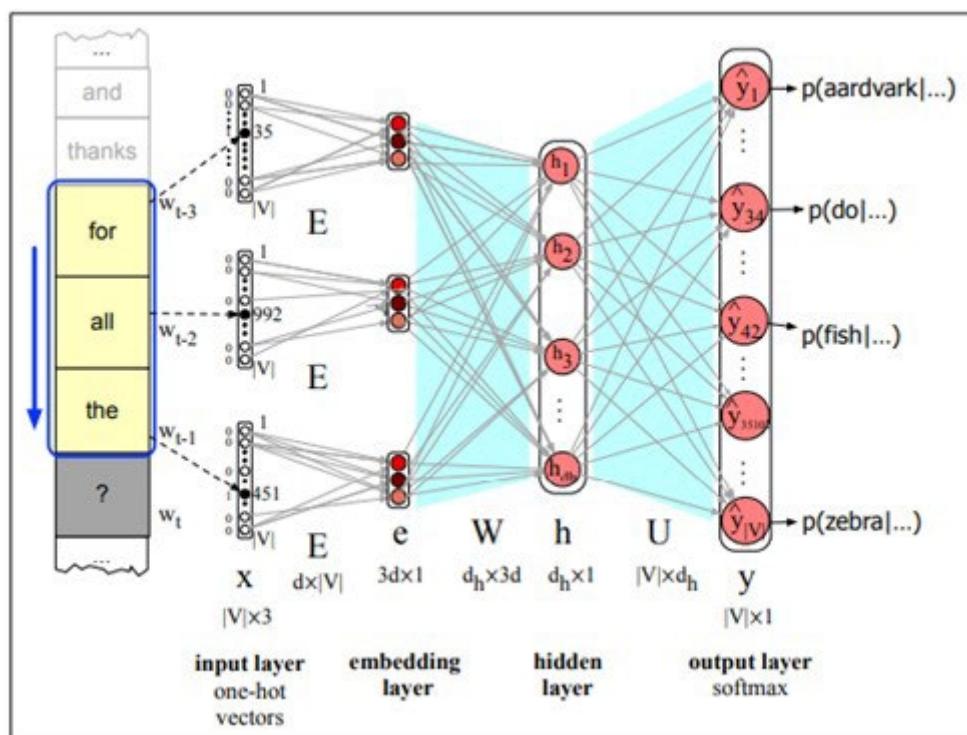


Figura 13. Arquitectura para un LM basado en *feed-forward networks*. Fuente: Jurafsky, 2021.

Como muestra la imagen, tras definir el tamaño de la secuencia, las palabras que la componen se expresan mediante sus vectores *one-hot encoding*, los cuales se usan para obtener los *embeddings* de esas palabras desde la matriz de pesos E. Estos vectores de *embeddings* se concatenan para generar un vector E final que se multiplica por la matriz de pesos W. Sobre la salida final se aplica la función de activación de la capa de salida (ej., softmax) para tener la probabilidad de que cada una de las palabras del vocabulario sigan a esa secuencia de entrada. De este modo, las ecuaciones de la red son:

$$\begin{aligned}\mathbf{e} &= [\mathbf{Ex}_{t-3}; \mathbf{Ex}_{t-2}; \mathbf{Ex}_{t-1}] \\ \mathbf{h} &= \sigma(\mathbf{We} + \mathbf{b}) \\ \mathbf{z} &= \mathbf{Uh} \\ \hat{\mathbf{y}} &= \text{softmax}(\mathbf{z})\end{aligned}$$

Un aspecto importante de esta arquitectura es la matriz de pesos E desde la que se obtienen los *embeddings*. Para los valores iniciales de esta matriz, en lugar de ser valores aleatorios, se puede usar la matriz de *embeddings* resultante de aplicar alguno de los algoritmos de *word embeddings* vistos en el apartado anterior.

Una vez que se tiene esta arquitectura junto con un corpus de entrenamiento, pasarán por ella distintas secuencias de entrada para tratar de predecir la palabra que sigue tras ellas, y en base a esta información se irán ajustando los pesos del modelo con técnicas como, por ejemplo, el **descenso de gradiente**.

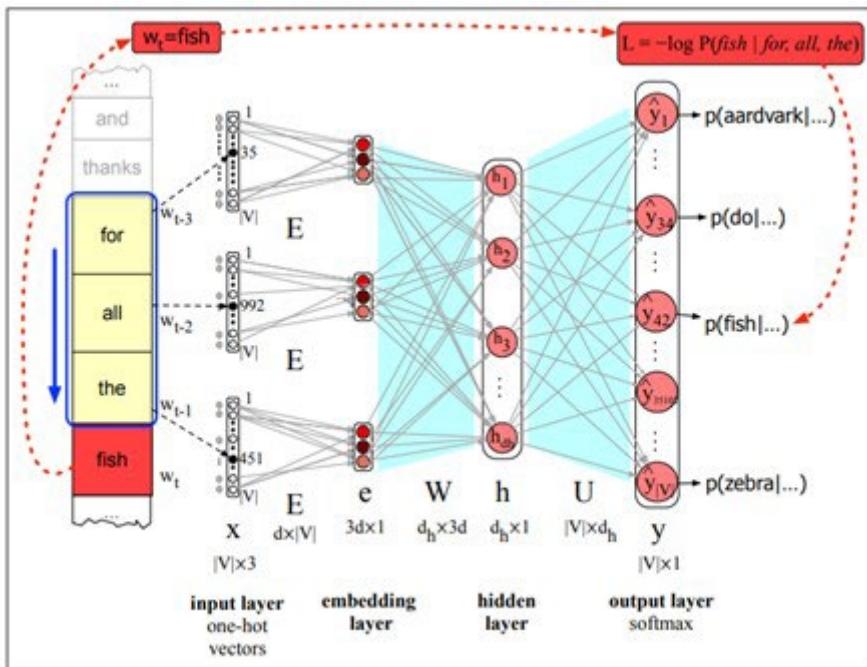


Figura 15. Cálculo de la función de coste en el LM. Fuente: Jurafsky, 2021.

Para completar el estudio de esta sección y ver en detalle las ecuaciones involucradas en el entrenamiento del modelo puedes leer el Capítulo 7 del libro: Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall. <https://web.stanford.edu/~jurafsky/slp3/7.pdf>

Redes neuronales recurrentes

Además de los modelos *feed-forward* vistos previamente, existen otras arquitecturas de redes neuronales que son útiles para el modelado del lenguaje, como las **redes neuronales recurrentes** (*recurrent neural networks, RNN*).

Las RNN se caracterizan por utilizar no sólo la información de entrada para calcular la salida, sino que también utilizan la información de los estados previos, de manera que son útiles para capturar información secuencial, como es el caso de los textos.

En la siguiente imagen se ve un ejemplo de RNN, donde x_t es la, W es la matriz de pesos, h_t está asociada a la capa oculta, y V es la matriz de pesos de salida. De esta manera, el valor de salida y_t para un instante t no depende sólo de la entrada en ese momento (x_t), sino que depende también de la información del estado previo. Esta información se recibe mediante otra matriz de pesos, U , que conecta la capa oculta del paso anterior con la capa oculta en el paso actual.

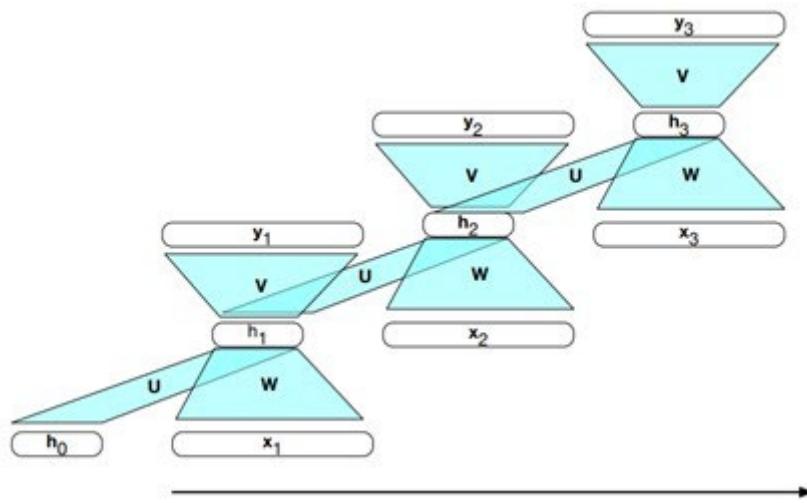


Figura 16. Ejemplo de arquitectura de RNN. Fuente: Jurafsky, 2021.

Trasladando el esquema de las RNN al problema del modelado del lenguaje, se tiene una arquitectura como la que muestra la siguiente imagen. Cada paso de la RNN corresponderá a la predicción de la siguiente palabra de la secuencia, usando información tanto de la última palabra (la palabra de entrada en el estado t), como la información previa de la secuencia (recibida desde el estado $t-1$, que contiene tanto la información de $t-1$ como la información previa a ese estado).

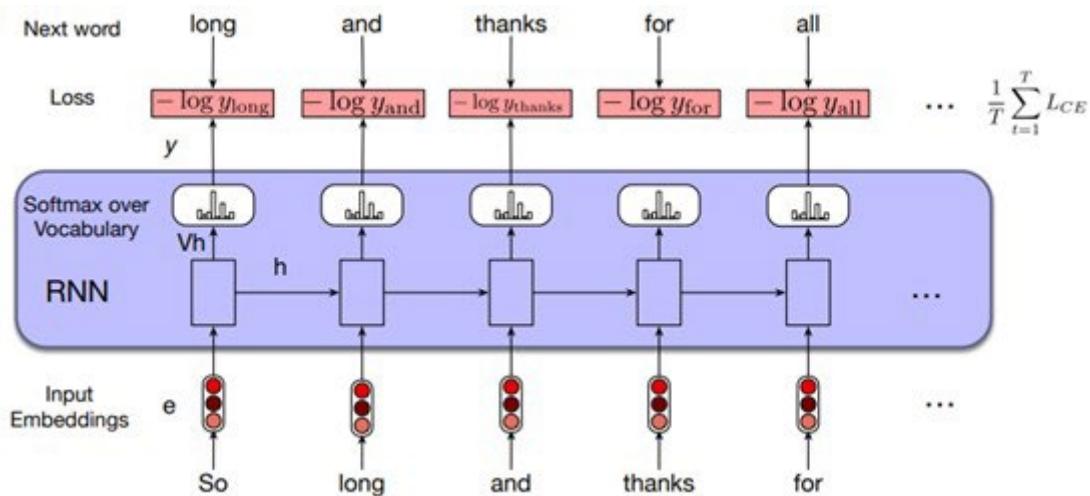


Figura 17. RNN para un LM. Fuente: Jurafsky, 2021.

El esquema de matrices es el mismo que el que se describió para la RNN en general, con una salvedad, y es que aparece una matriz adicional E , la matriz de *embeddings*, que convierte el valor del vector de la palabra de entrada (expresada mediante su *one-hot encoding*) en su vector de *embeddings* (e_t). Con ello, se tienen las siguientes ecuaciones para el vector de entrada x_t (el *one-hot encoding*) de la palabra de la secuencia en el instante t (donde g corresponde a la función de activación de la capa oculta):

$$\begin{aligned}\mathbf{e}_t &= \mathbf{E} \mathbf{x}_t \\ \mathbf{h}_t &= g(\mathbf{U} \mathbf{h}_{t-1} + \mathbf{W} \mathbf{e}_t) \\ \mathbf{y}_t &= \text{softmax}(\mathbf{V} \mathbf{h}_t)\end{aligned}$$

Como ocurría con las *feed-forward networks*, las matrices de pesos E , V y U son comunes para todas las palabras de la secuencia de entrada. La salida y_t es un vector con las probabilidades de que cada palabra del vocabulario sea la palabra siguiente de la secuencia. De esta manera, $y_t[i]$ correspondería a la probabilidad de que esa palabra fuese la siguiente en la secuencia dada las palabras previas, como se muestra a continuación:

$$P(w_{t+1} = i | w_1, \dots, w_t) = \mathbf{y}_t[i]$$

En la imagen previa también se muestra el cálculo de la función de coste en cada uno de los pasos. Al ser un modelo supervisado (auto-supervisado), se tiene la información de la palabra que realmente sigue a cada secuencia, de manera que se puede comparar con la palabra predicha para obtener el valor de la función de coste y ajustar los pesos. Utilizando la ecuación de la **entropía cruzada** con $y_t[w]$ el valor del vector de salida correcto para la palabra w e $\hat{y}_t[w]$ el valor predicho para esa palabra, se tiene el siguiente valor de la función de coste considerando todas las palabras del vocabulario.

$$L_{CE} = - \sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$$

Como el vector de salida está expresado también según su *one-hot encoding*, su valor puede ser 0 o 1, de manera que la función de coste se puede simplificar a la siguiente ecuación para calcular el error en el estado t con respecto a la predicción de la siguiente palabra de la secuencia, w_{t+1} .

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

Finalmente, en estas arquitecturas existe también un concepto, denominado **weight tying**, con el que se simplifica la estructura de la RNN, reduciendo el número de parámetros que se tienen que ajustar, considerando la matriz de salida V igual que la de los *embeddings*, E .

La intuición tras esto es que ambas matrices representan un concepto de embeddings similar, y ambas tienen una dimensionalidad que depende del tamaño del espacio de los embeddings y del tamaño del vocabulario.

Las ecuaciones quedarían:

$$\begin{aligned}\mathbf{e}_t &= \mathbf{Ex}_t \\ \mathbf{h}_t &= g(\mathbf{Uh}_{t-1} + \mathbf{We}_t) \\ \mathbf{y}_t &= \text{softmax}(\mathbf{E}^{\text{intercal}} \mathbf{h}_t)\end{aligned}$$

RNN apiladas

La arquitectura vista previamente de RNN para LM no es la única posible. En ocasiones, ocurre que el modelado de los datos puede ser muy complejo y esto requiera de arquitecturas que modelicen mejor las relaciones entre dichos datos. Por este motivo, existen también arquitecturas como las **RNN apiladas**, donde no se tienen una única capa oculta, sino que **se tienen distintas capas**, donde cada una de ellas recibe la salida de la capa previa, hasta llegar a la última que es la que generaría la predicción.

Este tipo de aproximaciones pueden dar resultados más precisos que las soluciones de una única capa al tener distintos niveles de abstracción, con los que capturar aspectos diversos de los datos de entrada.

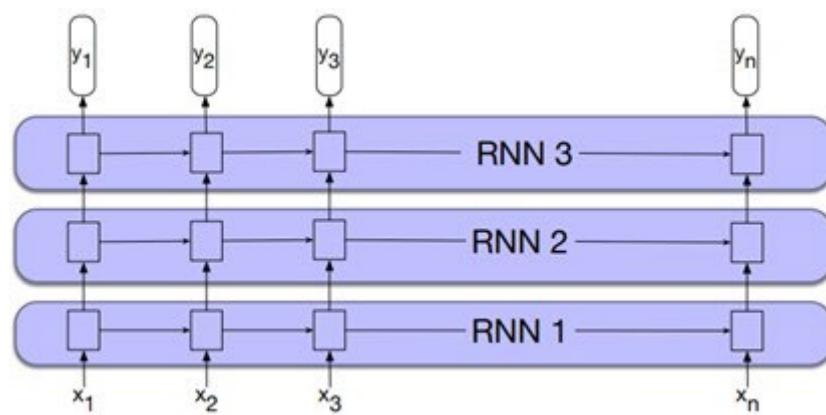


Figura 23. Arquitectura de RNN apiladas. Fuente: Jurafsky, 2021.

RNN bidireccionales

Las RNN vistas hasta ahora consideran de manera conjunta la información del estado actual y de los estados previos. No obstante, en ocasiones también es interesante considerar la información posterior a un determinado estado para tareas como el LM. Por ejemplo, si se quiere predecir la palabra herramienta en la siguiente frase:

- ▶ María buscó el gato entre sus herramientas para poder arreglar el coche.

La información previa para esa palabra «María buscó el gato entre sus» podría no ser tan útil como la información posterior «para poder arreglar el coche» para predecirla.

Así, existen también las RNN bidireccionales, donde cada estado recibe información tanto de la secuencia previa como de la posterior para hacer las predicciones.

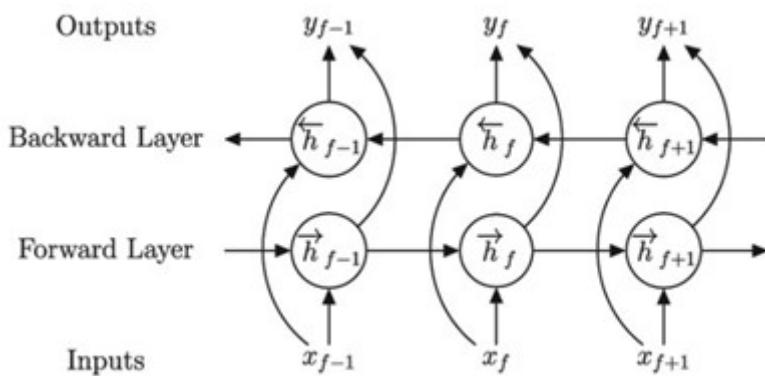


Figura 24. Arquitectura de una RNN bidireccional. Fuente: Graves et al., 2013.

Las ecuaciones asociadas a estas RNN son similares a las vistas hasta ahora, con la salvedad de que se distingue entre dos capas ocultas, las asociadas al procesamiento de la secuencia de *izquierda a derecha*, y las asociadas al procesamiento de la secuencia de *derecha a izquierda*. Para un instante f son, respectivamente, \vec{h}_f y \bar{h}_f . Con ello, una manera de construir la red es con las siguientes ecuaciones:

$$\begin{aligned}\vec{\mathbf{h}}_f &= g(\mathbf{W}_{x\vec{h}} \mathbf{x}_f + \mathbf{W}_{\vec{h}\vec{h}} \vec{\mathbf{h}}_{f-1} + \mathbf{b}_{\vec{h}}) \\ \overleftarrow{\mathbf{h}}_f &= g(\mathbf{W}_{x\overleftarrow{h}} \mathbf{x}_f + \mathbf{W}_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{\mathbf{h}}_{f-1} + \mathbf{b}_{\overleftarrow{h}}) \\ \mathbf{y}_f &= m(\mathbf{W}_{\vec{h}y} \vec{\mathbf{h}}_f + \mathbf{W}_{\overleftarrow{h}y} \overleftarrow{\mathbf{h}}_f + \mathbf{b}_y)\end{aligned}$$

Donde m , que corresponde a la función de activación sobre la capa oculta, puede ser la función *softmax* para el caso del modelado del lenguaje. En estas ecuaciones genéricas se incluyen los términos optionales de *bias* $b_{\vec{h}}$, $b_{\overleftarrow{h}}$ y b_y .

Como se puede observar, el uso de las RNN bidireccionales está condicionado a que esté disponible toda la secuencia del texto al usar tanto la información previa como la información posterior de una palabra.

Por este motivo, no son estructuras que sirvan para todas las tareas de PLN. Por ejemplo, no servirían para un **modelado de lenguaje causal** (donde se trata de predecir la siguiente palabra usando sólo la información previa de la secuencia), o para tareas de PLN como el autocompletado de frases.

Otras arquitecturas: LSTM

Las arquitecturas de RNN tienen varios problemas que se deben tener en cuenta.

Supongamos una frase como:

- ▶ Los vuelos que la aerolínea canceló estaban llenos.

En este caso, predecir una palabra como «canceló», en su tiempo verbal adecuado, puede ser factible al tener en su contexto cercano su sustantivo asociado «aerolínea». Ahora bien, la palabra «estaban» está vinculada con *vuelos*, y este sustantivo está muy lejos de su contexto cercano. Esto puede hacer que su predicción sea más complicada ya que, aunque se reciba la información previa de la secuencia, la información relevante está muy distante.

Adicionalmente, un problema común en las RNN es respecto a **retropropagar** (*backpropagate*) el error a lo largo de las distintas etapas para ajustar los pesos. Ocurre que, al ajustar los pesos en base a las derivadas parciales y a la regla de la cadena, el gradiente se irá reduciendo y tomando valores cada vez más pequeños en las etapas más lejanas, no ajustándolos adecuadamente. Este problema se conoce como **desvanecimiento de gradiente**.

Para evitar estos problemas, se han propuesto arquitecturas más complejas, como por ejemplo mediante el uso de **unidades ocultas como las LSTM** (*long short-term memory*).

Para completar el estudio de esta sección y ver más información sobre la necesidad del uso de las LSTM y su arquitectura interna, puedes leer el Capítulo 9 del libro Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*.

Prentice-Hall. <https://web.stanford.edu/~jurafsky/slp3/9.pdf>

6.4. Transformers

Ocurre que, aunque los modelos basados en RNN son útiles para modelar secuencias, como textos (y en particular para LM), tienen a su vez una serie de desventajas para tener en cuenta. Primero, aun usando estructuras avanzadas como las LSTM, no siempre se puede evitar la pérdida de información desde etapas anteriores lejanas. Segundo, la estructura de las RNN dificulta que se pueda procesar su información en paralelo.

Por este motivo, han surgido nuevos tipos de arquitecturas de redes neuronales, como los **transformers**, donde se busca conservar las ventajas de las RNN, solucionando los problemas mencionados.

Self-attention

Los *transformers* se componen de distintos bloques, donde se recibe un vector de datos de entrada (x_1, \dots, x_n) , y se genera un vector de salida (y_1, \dots, y_n) del mismo tamaño. Esto se lleva a cabo mediante distintos **bloques**, que se componen de distintas capas de redes neuronales. En concreto destaca una de ellas: la capa de **self-attention**. Esta capa de *self-attention* se describe en la siguiente imagen.

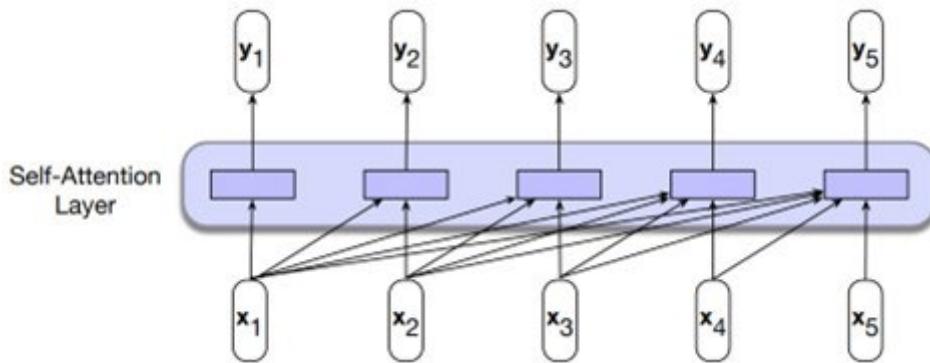


Figura 26. Capa de *self-attention*. Fuente: Jurafsky, 2021.

Como se puede observar, esta capa recibe un vector de entrada x y genera un vector de salida y del mismo tamaño. Cada valor y_i que se predice usa información del valor de entrada actual x_i , pero también recibe la información de todos los estados anteriores. Estos valores de entrada se expresan mediante un vector de *embeddings*, de manera análoga al resto de modelos vistos hasta ahora. A diferencia de las RNN, cada uno de los valores de salida se trata de manera independiente desde el punto de vista computacional, de manera que se pueden procesar en paralelo.

El cálculo de cada uno de esos valores y_i se lleva a cabo tal como indica la siguiente imagen, donde se muestra un ejemplo para el cálculo de y_3 .

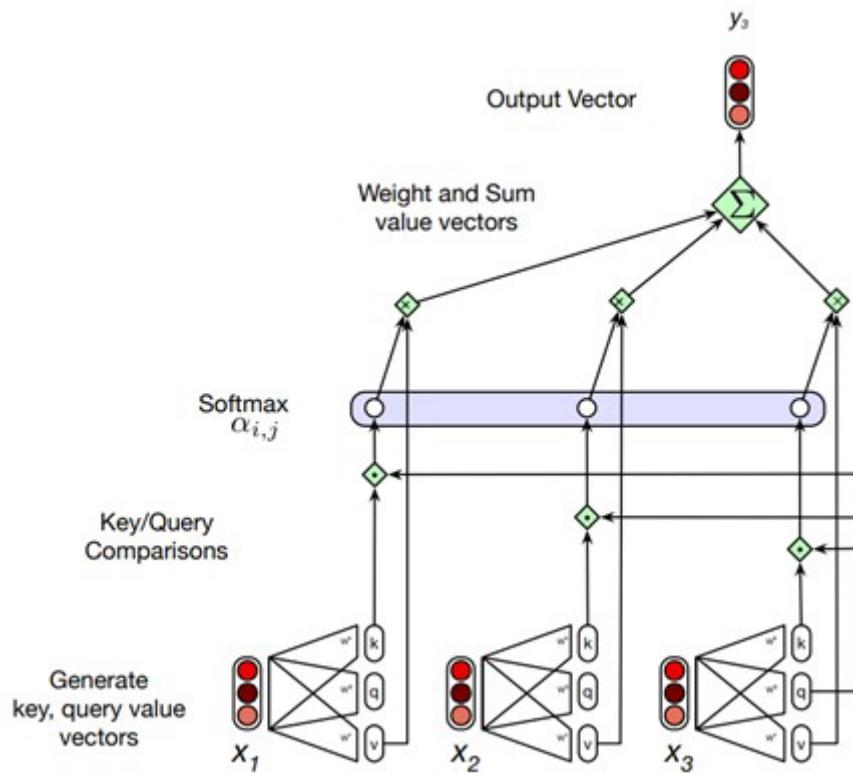


Figura 27. Cálculo de y_3 con el modelo de attention. Fuente: Jurafsky, 2021.

Como se puede observar, para el cálculo se parte de los datos de entrada, se obtiene desde ellos el vector de *embeddings* correspondiente para cada uno, y luego se usan para generar 3 vectores q , k y v , que, combinados entre sí, generan la salida final.

Estos 3 vectores son:

- ▶ El **foco actual** del modelo de *attention*, que para una salida y_i hace referencia a x_i . Este se denomina **query (q)**.
- ▶ Ese foco actual q es un vector que se va a comparar tanto con el elemento actual como con los elementos previos mediante un producto escalar entre los vectores que da como resultado el valor de su similitud. Así, Las entradas de todos los elementos dan lugar a los vectores **k (key)**, que se multiplican con **q** en cada elemento, dando como resultado un valor numérico. Todos estos valores concatenados dan lugar a un vector que recibe la capa *softmax* con la que se genera un vector de probabilidades final, con un valor $\alpha_{i,j}$ por cada elemento de entrada.

- ▶ Los valores de entrada también son usados directamente en el cálculo de la salida, en combinación con los resultados de los dos puntos anteriores (**value**, **v**). Así, cada vector de *embeddings* de entrada se usa para generar un vector **v** que se multiplica por la salida de la capa *softmax*. La suma de los elementos de este vector final da como lugar la salida de la capa de *attention* para ese valor y_i .

Trasladado a ecuaciones, y_i se obtiene de la siguiente manera, en base a las componentes $\alpha_{i,j}$ de la salida de la capa *softmax*, y a los vectores v_j obtenidos desde las entradas de los distintos elementos del modelo:

$$y_i = \sum_{j \leq i} \alpha_{ij} v_j$$

Por otro lado, los productos escalares entre el foco actual q_i y cada uno de los elementos de entrada expresados mediante un vector k_j dan un resultado como el siguiente (donde se ha normalizado en función del tamaño del vector k):

$$\text{score}(x_i, x_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

El cálculo de Q, K y V expresado matricialmente, donde se tienen los vectores para todos los elementos de entrada, son las siguientes:

$$Q = XW^Q$$

$$K = XW^K$$

$$V = XW^V$$

La matriz X corresponde a la matriz de *embeddings*, donde cada fila es el vector de *embeddings* asociado a uno de los elementos de entrada. Multiplicando esta matriz por las matrices W^Q , W^K y W^V se obtienen, respectivamente, los valores de las matrices Q, K y V. De esta manera, la salida del modelo de *self-attention* se puede expresar de la siguiente manera:

$$\text{SelfAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Ahora bien, para el cálculo de un determinado valor y_i sólo se usa la información del elemento actual y de los elementos previos, pero no de los elementos siguientes. Por este motivo, ese producto matricial anterior no se puede realizar directamente así, ya que si no se tendría en cuenta toda la secuencia para cada elemento. Para evitar esto (y poder, entre otras cosas, usar estos modelos para tareas como LM), los elementos de la matriz resultante del producto matricial de QK^T que van después del foco actual reciben un valor arbitrario que les hace ser nulos, de manera que su información no se use en el cálculo. Esto se ve en la siguiente imagen.

	$q_1 \cdot k_1$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	$q_2 \cdot k_1$	$q_2 \cdot k_2$	$-\infty$	$-\infty$	$-\infty$
N	$q_3 \cdot k_1$	$q_3 \cdot k_2$	$q_3 \cdot k_3$	$-\infty$	$-\infty$
	$q_4 \cdot k_1$	$q_4 \cdot k_2$	$q_4 \cdot k_3$	$q_4 \cdot k_4$	$-\infty$
	$q_5 \cdot k_1$	$q_5 \cdot k_2$	$q_5 \cdot k_3$	$q_5 \cdot k_4$	$q_5 \cdot k_5$

Figura 28. Ejemplo del caso de la matriz resultante de multiplicar Q por K^T , con Q y K de dimensión N x d.
Fuente: Jurafsky, 2021.

Modelo de *transformers*

Una vez conocido el funcionamiento de la capa *self-attention*, se puede entender el funcionamiento del modelo de *transformers*. El esquema de este modelo se describe en la siguiente imagen:

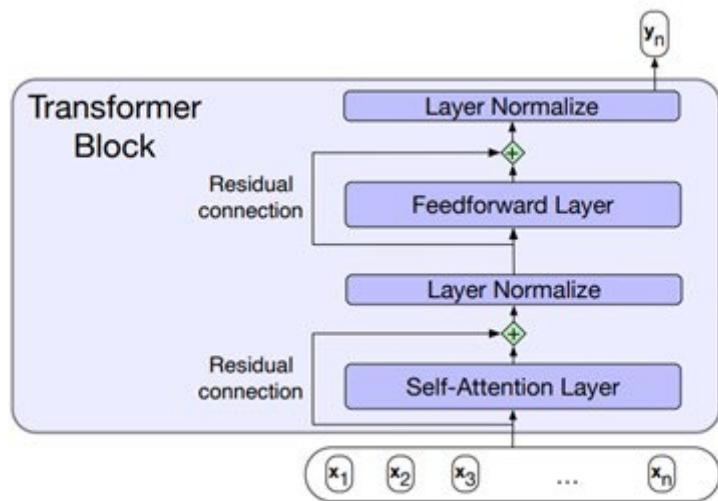


Figura 29. Esquema del modelo de *transformers*. Fuente: Jurafsky, 2021.

En este modelo, se recibe una entrada que pasa por la capa *self-attention* para generar una primera salida intermedia. Esta salida intermedia se combina de nuevo con la información de entrada mediante la **conexión residual**, de manera que el vector de entrada de la capa *self-attention* y la salida de dicha capa se combinan sumando sus valores.

Esta aproximación de la conexión residual es útil para no perder la información de las primeras capas de una red neuronal, recuperándola en fases posteriores al combinarla con la información que sale del propio procesamiento de la red.

Este vector pasa por una **capa de normalización** para asegurar que el vector tiene valores en una escala similar. El vector resultante sirve como entrada de una capa *feed-forward*, que produce un vector de salida que se vuelve a combinar con su entrada mediante otra conexión residual. El vector final se normaliza de nuevo, generando la salida final del modelo.

Expresándolo mediante ecuaciones se tiene:

$$\begin{aligned} \mathbf{z} &= \text{LayerNorm}(\mathbf{x} + \text{SelfAttn}(\mathbf{x})) \\ \mathbf{y} &= \text{LayerNorm}(\mathbf{z} + \text{FFNN}(\mathbf{z})) \end{aligned}$$

La **normalización** se suele hacer mediante el cálculo de la **media** y la **desviación típica del vector**, de manera que, por ejemplo, para un vector x sería:

$$\hat{x} = \frac{(x - \mu)}{\sigma}$$

Con μ la media, y con σ la desviación típica. Además, en lugar de usar directamente este resultado como salida de la capa de normalización, se puede ajustar con unos parámetros γ y β , que también serían parámetros que se ajustarían en la red durante la optimización del error, tal que:

$$LayerNorm = \gamma \hat{x} + \beta$$

Multiheaded Attention

En algunas ocasiones, un modelo simple de *self-attention*, como el que se ha visto previamente, no es suficiente para capturar la complejidad de los datos de entrada. Por este motivo, existen alternativas más avanzadas, como son los **modelos multiheaded**, donde, para una misma entrada, se usan distintos modelos *self-attention* (denominados *heads*), y la salida de todos ellos se combina en un único vector para generar con él la salida final. La imagen siguiente muestra esta arquitectura.

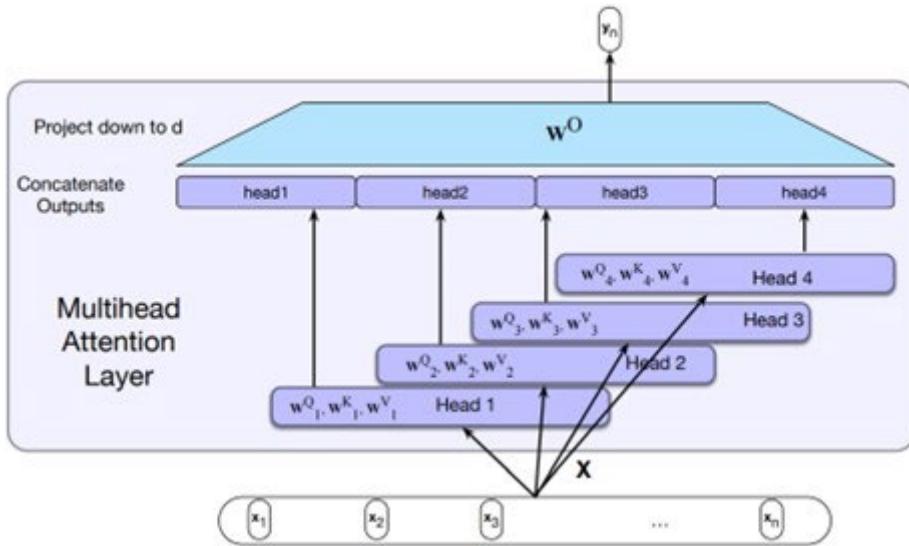


Figura 33. Modelo multiheaded attention con 4 heads. Fuente: Jurafsky, 2021.

De manera genérica, para i heads, se tienen las siguientes ecuaciones:

$$\begin{aligned} MultiHeadAttn(\mathbf{X}) &= (\mathbf{head}_1 \oplus \mathbf{head}_2 \dots \oplus \mathbf{head}_h) \mathbf{W}^O \\ \mathbf{Q} &= \mathbf{X} \mathbf{W}_i^Q ; \mathbf{K} = \mathbf{X} \mathbf{W}_i^K ; \mathbf{V} = \mathbf{X} \mathbf{W}_i^V \\ \mathbf{head}_i &= SelfAttention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \end{aligned}$$

Embeddings posicionales

En los modelos vistos previamente, cuando los datos de entrada son palabras, estas se expresan con su correspondiente vector de *embeddings*, el cual muchas veces toma como valor el resultante de modelos como Skip-gram. Ahora bien, estos vectores representan el valor semántico de la palabra en cuestión, pero no tienen en cuenta otro tipo de información, como la posición de la palabra dentro de la secuencia. Por este motivo, existen también los *embeddings posicionales*, donde se tienen vectores densos con los que representa la posición de las palabras, y se captura información como que la palabra de la posición 4 es más cercana a la 3 que a la 19.

El uso de *embeddings* posicionales en los modelos de transformers es importante ya que su arquitectura, a diferencia de las RNN, no conserva la información relativa al orden de las palabras (se tiene información de la secuencia previa a una palabra en concreto, pero no del orden de las palabras como tal).

Este problema se solventa con la combinación de los dos *embeddings* para generar el vector de entrada del modelo.

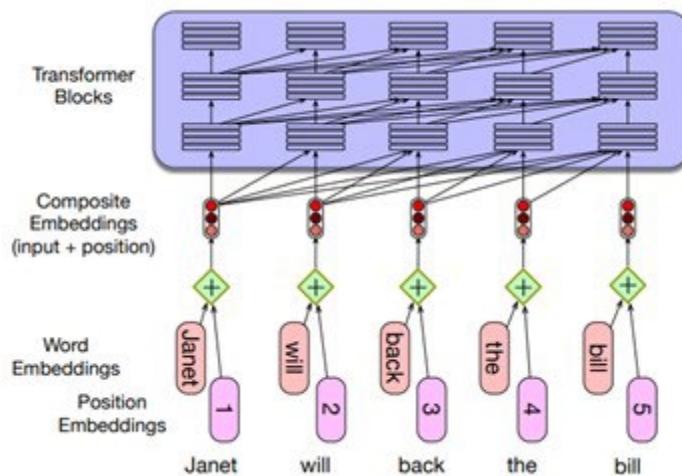


Figura 35. Ejemplo de combinación de *word embeddings* junto con *embeddings* posicionales. Fuente: Jurafsky, 2021.

Transformers como modelos de lenguaje

Los modelos de *transformers* pueden usarse, como ya se ha comentado, para **problemas de LM**. Para una secuencia de entrada, se usa una capa con el modelo de *transformers* que recibe como entrada los *embeddings* de la palabra en un estado concreto y de toda la secuencia previa, y con ello se construye un modelo en el que el vector de salida corresponda a la distribución de probabilidad sobre todas las palabras del vocabulario, donde se trate de predecir la palabra que seguirá a la secuencia. Esto se muestra en la imagen siguiente:

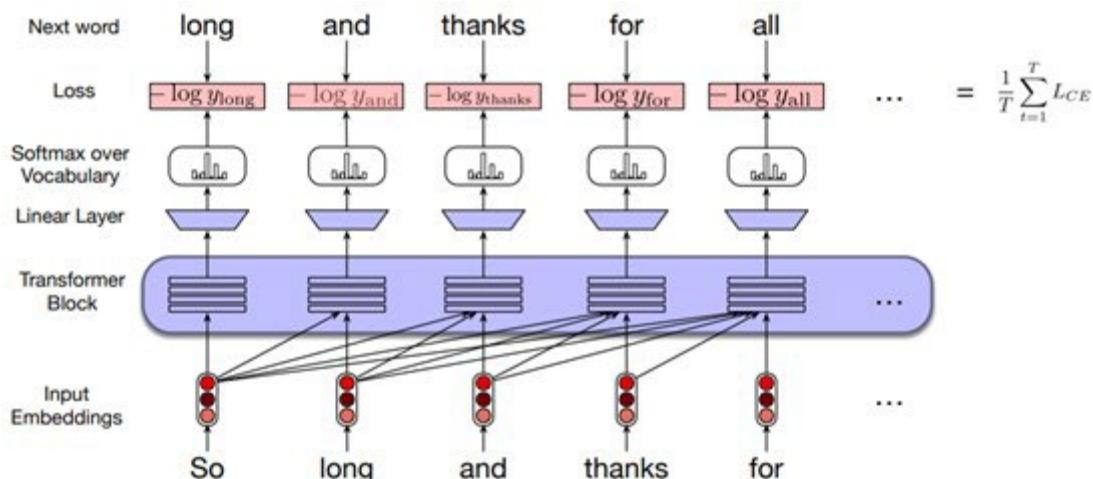


Figura 36. Uso de *transformers* para LM. Fuente: Jurafsky, 2021.

Para completar el estudio de esta sección y ver más información sobre los *transformers*, puedes leer el Capítulo 9 del libro, pág. 17-25: Jurafsky, D. y Martin, J. H. (2021). Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics. Prentice-Hall.

<https://web.stanford.edu/~jurafsky/slp3/9.pdf>

6.5. BERT

Los algoritmos que se han visto hasta ahora para generar *word embeddings* tienen en común una cosa: usando la información de un corpus, se generan **representaciones estáticas** de las palabras mediante vectores dentro de un espacio vectorial. Es decir, se busca modelar el significado de las palabras desde una perspectiva general de su uso, pero no desde el contexto concreto de una oración. Ahora bien, en muchas ocasiones es importante considerar el contexto concreto de una palabra para entender su significado. Por ejemplo:

- ▶ María buscó el gato entre sus herramientas para poder arreglar el coche.
- ▶ María buscó a Garfield, su gato doméstico, para darle de comer.

Ambas oraciones comparten una serie de palabras, como es el caso de «gato». Sin embargo, su significado cambia mucho debido al contexto concreto de uso: en el primer caso hace referencia a una herramienta y en el segundo a un animal. Es por este motivo que aparecen los ***embeddings contextuales***, con los que se busca obtener un vector de *embeddings* que represente el significado de las palabras, teniendo en cuenta tanto el conocimiento global (el corpus en general) como el local (el contexto de uso). Una forma de obtener estos *embeddings* es mediante el uso del **modelo BERT** (Bidirectional Encoder Representations from Transformers) (Delvin et al., 2019).

Transformers bidireccionales

El modelo **BERT** se basa en el concepto de **transformers bidireccionales**, que tienen en cuenta no sólo el contexto de la secuencia de izquierda a derecha, sino también de derecha a izquierda (como ocurría con las RNN bidireccionales). De esta manera, la capa de *self-attention* sería como muestra la siguiente imagen:

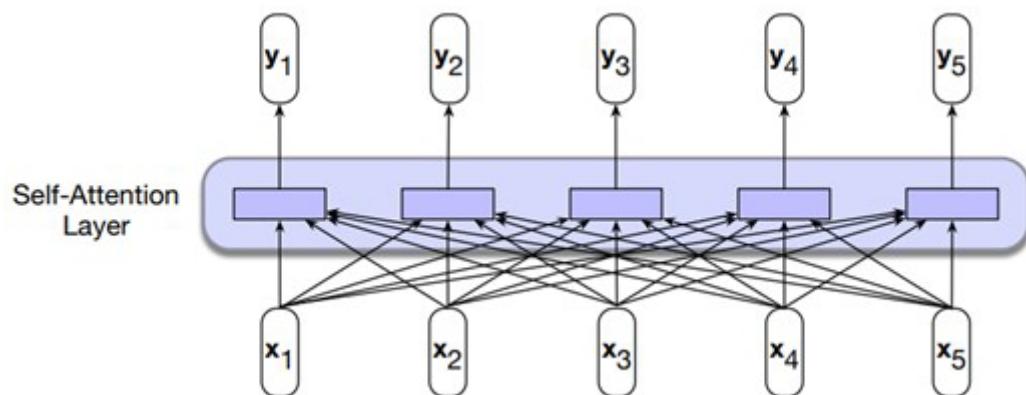


Figura 37. Capa de *self-attention* bidireccional. Fuente: Jurafsky, 2021.

Esta arquitectura no seguiría el esquema causal de izquierda a derecha con el que construir un LM para poder predecir cómo continuará una frase, pero sirve para otras tareas, como la construcción de *embeddings* contextuales. Su construcción es trivial con lo visto hasta ahora: sería el mismo cálculo matricial que en el caso no bidireccional, eliminando el paso de enmascarar la información de la matriz

resultante de QK^T asociada a las palabras siguientes a cada palabra de la secuencia.

De esta manera, sería:

	q1•k1	q1•k2	q1•k3	q1•k4	q1•k5
N	q2•k1	q2•k2	q2•k3	q2•k4	q2•k5
	q3•k1	q3•k2	q3•k3	q3•k4	q3•k5
	q4•k1	q4•k2	q4•k3	q4•k4	q4•k5
	q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

Figura 38. Matriz completa resultante del producto QK^T . Fuente: Jurafsky, 2021.

Con esto, la arquitectura de BERT sigue el esquema estándar de los *transformers*, con esta particularidad de las capas de *self-attention* bidireccionales y con las siguientes particularidades:}

- ▶ En lugar de usar palabras como dato de entrada se usan **subpalabras**. Estas se obtienen mediante un proceso de tokenización con el algoritmo *WordPierce* (Schuster and Nakajima, 2012). Así, en lugar de usar palabras, se usan partes de esas palabras. Un ejemplo de salida de este algoritmo se ve en la siguiente imagen.



Figura 39. Ejemplo de tokenización en subpalabras con WordPierce. Fuente: Google Research, 2021.

- ▶ Las capas ocultas tienen una dimensionalidad de 768.
- ▶ Se usan doce bloques de *transformers*, con doce capas *multihead attention*.

De esta manera, se tiene un modelo de 100M de parámetros.

Entrenamiento del modelo

Dado que en este caso no se busca construir un modelo que prediga la siguiente palabra de la secuencia (ajustando sus parámetros en función de cómo de cercana sea la distribución de probabilidad a la referencia de la salida), se plantea lo siguiente: omitir algunas palabras de la frase, y que el modelo trate de predecirlas, usando el resto de la secuencia (palabras previas y posteriores). Por ejemplo, se puede partir de la frase

- ▶ María buscó a Garfield, su gato doméstico, para darle de comer.

Y tener:

- ▶ María buscó a Garfield, su _____ doméstico, para darle de comer.

De esta manera, el modelo trataría de predecir la palabra «gato» dado el resto de las palabras de la secuencia. La manera con la que se lleva a cabo esto en BERT es mediante la selección de las frases de entrenamiento, y sobre cada una de ellas, seleccionar un subconjunto de tokens de entrada para usar en el entrenamiento (tratar de predecirlos usando los tokens conocidos, que sería el resto de ellos). BERT usa un 15 % de los tokens totales para este subconjunto a predecir. Sobre él se hace lo siguiente:

- ▶ El 80 % de ellos se enmascaran cambiándolos por un token [MASK], de manera que el modelo trate de predecir su valor usando la información de los tokens conocidos.
- ▶ El 10 % de ellos se reemplazan por una palabra aleatoria del vocabulario (en base a las probabilidades de los unigramas de los tokens).

- ▶ El 10 % restante de las palabras elegidas para entrenar el modelo se dejan sin cambiar.

El motivo de combinar tokens con [MASK] junto con tokens con palabras no enmascaradas es que el modelo se pueda utilizar para obtener predicciones sobre palabras aun cuando no tengan ese [MASK] (por ejemplo, para cuando se usa para hacer *fine-tuning* en el ámbito de *transfer learning*). Se usan tanto palabras aleatorias como las propias palabras en esas predicciones sin [MASK] para que haya un cierto sesgo en el modelo hacia la palabra en cuestión. Un ejemplo de entrenamiento y predicción con BERT se ve en la imagen siguiente. Como entrada, BERT usa también una combinación de los *embeddings* originales de esos tokens junto con su *embedding* posicional. Estos *embeddings*, de nuevo, se han obtenido previamente a nivel de subpalabras.

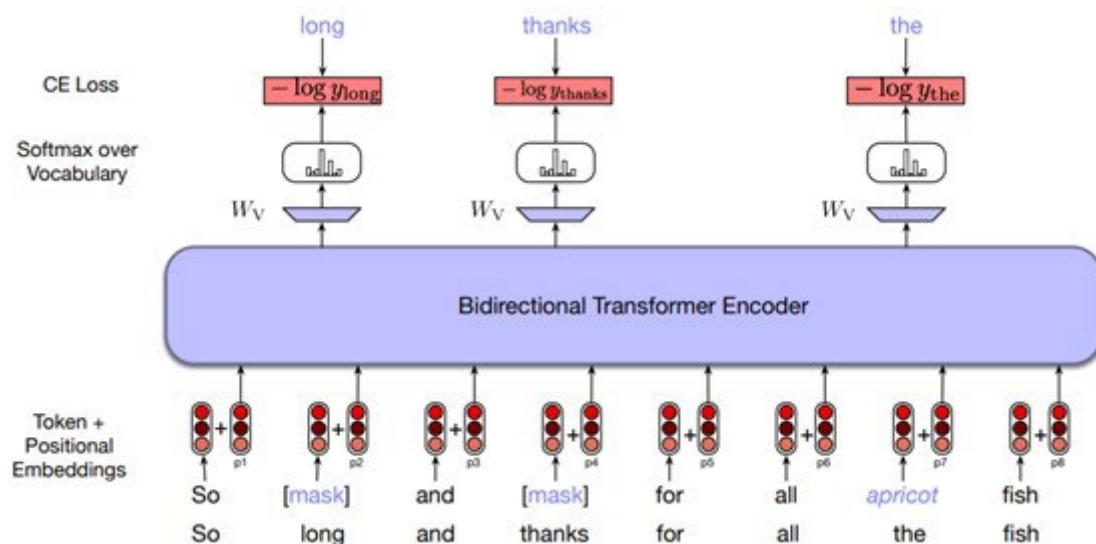


Figura 40. Ejemplo de entrenamiento en BERT. Fuente: Jurafsky, 2021.

Sobre una secuencia concreta, el modelo trata de predecir algunos de los tokens, que pueden estar enmascarados con un token genérico [MSK], o pueden estar reemplazados por una palabra aleatoria. También pueden estar sin cambiar con sus valores originales. En base a la probabilidad predicha sobre el vocabulario frente a la palabra real, se ajustan los valores del modelo.

Embeddings contextuales

Teniendo el modelo de BERT ya entrenado, se puede utilizar para obtener **embeddings contextuales de palabras**. Con estos *embeddings* se obtienen representaciones vectoriales de palabras en función de la secuencia de texto donde aparecen. Por ejemplo, dada una secuencia de tokens (x_1, \dots, x_n) , si se quiere obtener el *embedding* del token x_i , esta se puede obtener usando esa secuencia como entrada del modelo, y obteniendo el vector y_i correspondiente a la última capa. Se puede también usar la salida de distintas capas del modelo con esa entrada, y construir el vector final de *embeddings* como una media de esos vectores.

Transfer learning

La tarea de entrenar un modelo como BERT es muy costosa computacionalmente, ya que se utilizan corpus de entrada de gran volumen de cara a poder modelar lo mejor posible el conocimiento de una lengua. Por este motivo, **es habitual trabajar con modelos de BERT ya pre entrenados, y utilizarlos para distintas tareas de PLN según se necesiten**. Así, se puede usar el modelo como entrada de otro proceso, como por ejemplo para obtener directamente los *embeddings* de una oración. Esto se relaciona con el concepto de **transfer learning**, donde el **conocimiento inferido por un modelo en su entrenamiento en una tarea específica se puede trasladar a otra**.

Para completar el estudio sobre *embeddings* contextuales, BERT y *transfer learning* puedes leer el Capítulo 11 del libro: Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall.

<https://web.stanford.edu/~jurafsky/slp3/11.pdf>

En el vídeo *Transformers y BERT* se verá cómo se puede modelar el lenguaje gracias al aprendizaje profundo, viendo arquitecturas novedosas como es el caso de los *transformers*.



Accede al vídeo

6.6. Referencias bibliográficas

Aggarwal, C. (2018). *Machine Learning for Text*. Springer Cham.

Bojanowski, P., Grave E., Joulin A. y Mikolov, T. (2017). *Enriching word vectors with subword information*. TACL, 5:135–146

Devlin, J., Chang, M.-W., Lee, K. y Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL HLT*.

Graves, A., Jaitly, N., y Mohamed, A. R. (2013, December). *Hybrid speech recognition with deep bidirectional LSTM*. In 2013 IEEE workshop on automatic speech recognition and understanding (pp. 273-278). *IEEE*.

Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall.

Schuster, M. y Nakajima, K. (2012). *Japanese and korean voice search*. ICASSP.

A fondo

Hugging Face

Hugging Face. <https://huggingface.co/>

Puedes ver ejemplos de cómo trabajar con BERT usando Python con la librería Hugging Face y con los tutoriales disponibles en su página web

Test

1. Indicar cuál de estas afirmaciones es verdadera respecto a los modelos de representación vectorial:

 - A. Un modelo de *word embeddings* siempre va a generar vectores de mayor dimensión que uno basado en bolsas de palabras.
 - B. Las representaciones vectoriales basadas en bolsas de palabras son densas, mientras que las basadas en *word embeddings* son dispersas.
 - C. Con un modelo BoW se puede ver la relación entre palabras, a diferencia de un modelo de *word embeddings*.
 - D. Un modelo de *word embeddings* permite ver la similitud entre palabras e incluso hacer operaciones con los vectores que recogen el significado.
2. El modelo de Skip-gram:

 - A. Infiere el significado de una palabra usando un modelo que trata de predecir el contexto (palabras vecinas) dada esa palabra.
 - B. Infiere el significado de una palabra usando un modelo que trata de predecir la palabra dado el contexto (palabras vecinas) de esa palabra.
 - C. Infiere el significado de una palabra construyendo un vector que recoge el número de veces que aparecen algunas palabras en el texto.
 - D. Se caracterizan por usar sólo representaciones binarias para indicar las palabras que aparecen en un determinado texto.

3. El modelo de CBOW:

- A. Infiere el significado de una palabra usando un modelo que trata de predecir el contexto (palabras vecinas) dada esa palabra.
- B. Infiere el significado de una palabra usando un modelo que trata de predecir la palabra dado el contexto (palabras vecinas) de esa palabra.
- C. Infiere el significado de una palabra construyendo un vector que recoge el número de veces que aparecen algunas palabras en el texto.
- D. Se caracterizan por usar sólo representaciones binarias para indicar las palabras que aparecen en un determinado texto.

4. ¿Cuál de estos modelos servirá mejor para obtener la representación de palabras desconocidas no presentes en el corpus de entrenamiento?

- A. Skip-gram.
- B. CBOW.
- C. FastText.
- D. TF-IDF.

5. Indica cuál de las siguientes afirmaciones es falsa respecto a la construcción de modelos de lenguaje (LM) con redes neuronales:

- A. Pueden usar la información de los *word embeddings* para representar los tokens de entrada.
- B. Se construye usando tanto arquitecturas de redes neuronales recurrentes, como de redes *feed-forward*.
- C. Las redes neuronales recurrentes bidireccionales son útiles para la construcción de LM y para tareas de PLN como el autocompletado de textos.
- D. Se pueden usar estructuras como las LSTM para evitar perder información de las etapas más lejanas de la secuencia.

- 6.** Indicar cuál de estas afirmaciones es verdadera respecto capa de *self-attention*:
- A. Usa la información del foco actual (*query*, *q*) y de la secuencia hasta una etapa en concreto (*key*, *k*) para construir un vector que sirve como entrada para una capa *softmax*. La salida de esta capa se corresponde a la salida de la capa *self-attention*.
 - B. Usa la información de la secuencia hasta una etapa en concreto (*key*, *k*) para construir un vector que sirve como entrada para una capa *softmax*. La salida de esta capa se combina de nuevo con la información de entrada (*value*, *v*) para obtener la salida de la capa *self-attention*.
 - C. Usa la información del foco actual (*query*, *q*) para construir un vector que sirve como entrada para una capa *softmax*. La salida de esta capa se combina de nuevo con la información de entrada (*value*, *v*) para obtener la salida de la capa *self-attention*.
 - D. Usa la información del foco actual (*query*, *q*) y de la secuencia hasta una etapa en concreto (*key*, *k*) para construir un vector que sirve como entrada para una capa *softmax*. La salida de esta capa se combina de nuevo con la información de entrada (*value*, *v*) para obtener la salida de la capa *self-attention*.
- 7.** ¿Cuál es la estructura de un bloque de *transformers* básico?
- A. Capa *self-attention*, capa de normalización, capa *softmax*, capa normalización.
 - B. Capa *self-attention*, conexión residual, capa de normalización, capa recurrente con LSTM, conexión residual, capa normalización.
 - C. Capa *self-attention*, conexión residual, capa de normalización, capa *feed-forward*, conexión residual, capa normalización.
 - D. Capa *self-attention*, capa de normalización, capa *feed-forward*, conexión residual, capa normalización.

8. Indicar cuál de estas afirmaciones es verdadera sobre los *transformers*:

- A. Se usan *embeddings* contextuales en lugar de *word embeddings* como vectores de entrada para obtener mejores resultados.
- B. La capa de normalización es siempre estática, y no tiene parámetros que se ajusten durante el entrenamiento del modelo.
- C. Se pueden usar con una arquitectura *multiheaded*, donde se tienen varias capas *self-attention*, y cada una da una salida que se combina en un único vector final que sirve como salida de esa capa *multiheaded*.
- D. No es posible usar *transformers* como modelos causales del lenguaje ya que siempre son bidireccionales y tienen por tanto información posterior de la secuencia con respecto a cada palabra.

9. Indicar cuál de estas afirmaciones es falsa sobre el modelo BERT:

- A. Usan como entrada tokens que contienen subpalabras, en vez de palabras.
- B. Usan *transformers* bidireccionales, donde se tiene en cuenta tanto la información previa como posterior de la secuencia.
- C. Al ser bidireccional sirve para tareas de NLP, como el autocompletado de textos.
- D. Usa capas de *multiheaded attention*.

10. Indicar cuál de estas afirmaciones es verdadera sobre el modelo BERT:

- A. Permite la generación de *embeddings* contextuales, donde se obtiene el *embedding* de una palabra sólo teniendo en cuenta el contexto local de la frase.
- B. Permite la generación de *embeddings* contextuales, donde se obtiene el *embedding* de una palabra no sólo de manera global, sino teniendo también en cuenta el contexto local de la frase.
- C. Es habitual entrenar el modelo BERT desde cero cada vez que se vaya a llevar a cabo una tarea de PLN, para tener en cuenta únicamente los textos asociados a dicha tarea.
- D. BERT se entrena para predecir unos tokens del corpus de entrenamiento dado el resto de tokens, donde todos estos tokens se reemplazan por un token genérico [MSK].

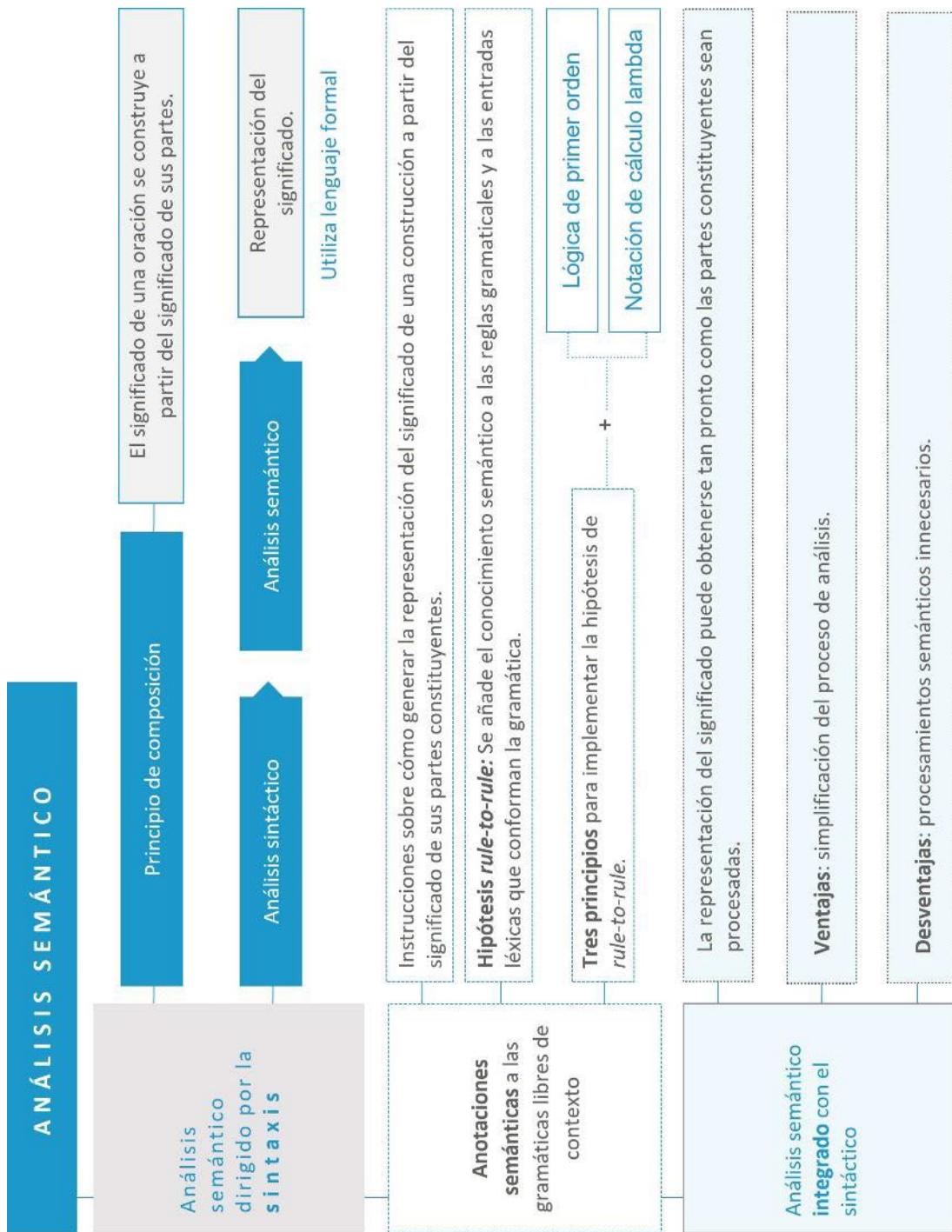
Procesamiento del Lenguaje Natural

Análisis semántico

Índice

Esquema	3
Ideas clave	4
7.1. Introducción y objetivos	4
7.2. Análisis semántico dirigido por la sintaxis	4
7.3. Anotaciones semánticas a las gramáticas libres de contexto	9
7.4. Análisis semántico integrado con el análisis sintáctico	16
7.5. Referencias bibliográficas	17
A fondo	19
Test	20

Esquema



7.1. Introducción y objetivos

Los objetivos al finalizar el estudio de los siguientes apartados son:

- ▶ Describir el funcionamiento del análisis semántico basado en el principio de composición del significado, es decir, el análisis semántico dirigido por la sintaxis.
- ▶ Modelar las anotaciones semánticas que se hace de las gramáticas libres de contexto para que la información sintáctica representada en forma de reglas se complemente con información semántica.
- ▶ Describir las características de realizar el análisis semántico en paralelo con el análisis sintáctico, al tener la semántica incorporada en la representación sintáctica

7.2. Análisis semántico dirigido por la sintaxis

El objetivo del análisis semántico es **producir una representación del significado de la oración** y, tal como se ha explicado anteriormente, **requiere de múltiples fuentes de conocimiento** como, por ejemplo, el conocimiento sobre los significados de las palabras. Entonces, **el análisis semántico dirigido por la sintaxis se basa en el principio de composición.**

La idea fundamental del principio de composición es que **el significado de una oración se construye a partir del significado de sus partes.**

Una frase se compone de palabras y las palabras son las unidades básicas de significado. Por lo tanto, se podría interpretar que, de acuerdo con el principio de

composición, el significado de una oración es el conjunto de los significados de las palabras que la componen, sin embargo, esta primera interpretación tan básica es poco útil.

El significado de una oración no se basa solamente en el significado de las palabras que la componen, sino que también en el orden y la agrupación de palabras y en las relaciones entre las palabras en la frase. Es decir, que **el significado de una frase está también basado en su estructura sintáctica.**

Por lo tanto, en el análisis semántico dirigido por la sintaxis, la composición del significado de la oración se hace a partir de la estructura sintáctica de la frase.

La Figura 1 muestra una posible estructura del proceso de análisis semántico dirigido por la sintaxis. Se observa que el resultado del análisis sintáctico de una oración sirve como entrada para el análisis semántico. De esta forma, **una frase que vaya a ser analizada pasa primero por un analizador sintáctico que extraiga la información sintáctica relativa a las relaciones estructurales entre palabras. La estructura sintáctica extraída, que normalmente se representa como un árbol sintáctico tal como se ha visto en temas anteriores, sirve para alimentar el analizador semántico, que producirá una representación del significado de la oración.**

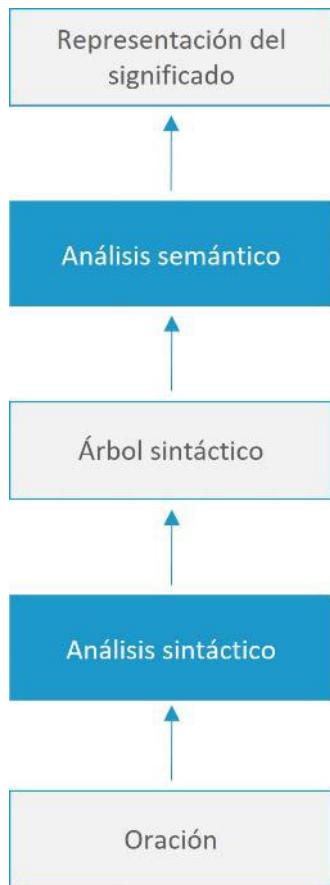


Figura 1. Estructura del proceso de análisis semántico dirigido por la sintaxis. Fuente: elaboración propia.

Cabe destacar que, en diferentes tareas de PLN presentadas en temas anteriores, se ha observado el problema de la ambigüedad. Sin embargo, en **el análisis semántico dirigido por la sintaxis se asume que la ambigüedad no es un problema** para el analizador semántico, ya que se va a tratar en otras etapas del proceso de análisis.

Por ejemplo, el analizador sintáctico será capaz de resolver las ambigüedades en los árboles sintácticos y solo proporcionará el mejor árbol sintáctico al analizador semántico. Otra opción sería que el analizador sintáctico encontrara varios posibles árboles sintácticos y se encargara de alimentar al analizador semántico con cada uno de estos posibles árboles y, una vez obtenidas las diferentes representaciones del significado, se aplicara un proceso de desambiguación en un nivel superior.

Ejemplo ilustrativo 1

Análisis semántico dirigido por la sintaxis de la frase en inglés:

- ▶ Francis likes Frasca.

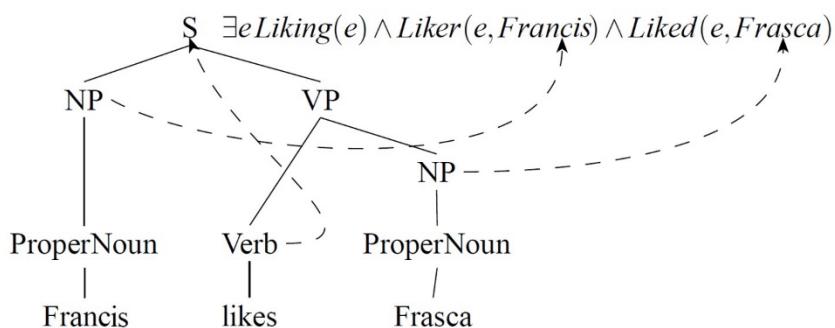


Figura 2. Árbol sintáctico anotado con la representación del significado resultante del análisis semántico de «Francis likes Frasca». Fuente: Jurafsky y Martin, 2009.

El análisis semántico de la frase «*Francis likes Frasca*» (en español, «A Francis le gusta Frasca») se realiza a partir del árbol sintáctico derivado del proceso previo de análisis sintáctico. Al árbol se le asigna la representación del significado utilizando un lenguaje formal como, por ejemplo, lógica de primer orden.

Las líneas discontinuas en la Figura 2 muestran la asociación de los nodos del árbol sintáctico con los diferentes elementos de la representación del significado a los que hacen referencia. Por ejemplo:

«Frasca» es nombre propio (*ProperNoun*), núcleo del sintagma nominal (NP) y forma parte del predicado verbal (VP), cuyo núcleo es «*likes*», el verbo (*Verb*); en el significado de la frase se representa, en lógica de primer orden, como el objeto de la expresión «*Liked(e,Frasca)*». Es decir, «Frasca» es el objeto de una relación del tipo *Liked*, donde el sujeto es el elemento *e*, del cual se sabe que es del tipo *Liking* y que, además, está relacionado con el objeto «Francis» a través de la relación *Liker*.

Para poder obtener e interpretar la representación del significado, es necesario definir formalmente utilizando lógica de primer orden todo el conocimiento necesario, es decir, qué significa una relación del tipo *Liked*, un elemento del tipo *Liking* o una relación del tipo *Liker*.

Como resultado del análisis semántico se obtiene la **representación del significado de la oración**, y esta representación se modela utilizando un lenguaje formal como puede ser la lógica de primer orden o la lógica descriptiva. Por tanto, el analizador semántico va a necesitar acceder a una descripción formal del conocimiento sobre el significado de las palabras con las que trabaja.

Esto se observa en el Ejemplo ilustrativo 1 donde se ha modelado el verbo «*like*» (gustar) a través del objeto *Liking* y las relaciones *Liked* y *Liker*. El analizador semántico, al encontrarse en el árbol sintáctico con el verbo «*like*», debe saber proporcionar la representación adecuada para ese verbo en concreto.

El algoritmo del analizador semántico interpreta y conoce cierta información general sobre los árboles sintácticos como que, entre otras cosas:

- ▶ Es el verbo el que carga con el significado principal de la oración.
- ▶ Los sintagmas que complementan el verbo son elementos que proporcionan información relacionada con el verbo.

Esta información es genérica, sin embargo, tal como se ha comentado antes, dependiendo del verbo en concreto, la representación formal del significado de la frase puede variar; en otras palabras, no hay una plantilla o modelo concreto que permita generar la representación del significado para cualquier verbo y esta depende el verbo en cuestión y cómo se ha formalizado el conocimiento.

En resumen, el analizador semántico precisa tanto de conocimiento sobre el árbol sintáctico como de conocimiento sobre el ejemplo o dominio concretos de la frase que se analiza para poder crear de forma automática la representación del significado.

Dado que hay un número infinito de posibles árboles sintácticos para una gramática, diseñar un analizador semántico específico para cada árbol es totalmente inviable. Es por eso por lo que la estructura del proceso de análisis semántico dirigido por la

sintaxis presentada en la Figura 1, que separa el análisis sintáctico del análisis semántico, no es muy común en la realidad. Lo que se hace es integrar el conocimiento semántico en el propio proceso de generación del árbol sintáctico. Para ello, se añade el conocimiento semántico a las reglas gramaticales y a las entradas léxicas que conforman la gramática, en lo que se conoce como *rule-to-rule hypothesis* (Bach, 1976).

7.3. Anotaciones semánticas a las gramáticas

libres de contexto

Una gramática libre de contexto se puede aumentar añadiéndole a las reglas anotaciones sobre el conocimiento semántico para así, poder implementar el análisis semántico guiado por la sintaxis.

Las anotaciones semánticas son instrucciones que especifican cómo generar la representación del significado de una construcción a partir de los significados de sus partes constituyentes.

Una regla con anotaciones semánticas tiene la siguiente estructura:

$$A \rightarrow \alpha_1 \dots \alpha_n \quad \{ f(\alpha_j.sem, \dots, \alpha_k.sem) \}$$

La anotación semántica que se añade a la notación básica de una regla libre de contexto se muestra en la parte derecha de los constituyentes sintácticos, dentro del símbolo de las llaves {...}. Esta notación establece que la representación del significado asignada a la construcción A , llamada $A.sem$, se puede calcular ejecutando la función f en un subconjunto de las anotaciones semánticas de los constituyentes de A . Donde $\alpha_1 \dots \alpha_n$ son las partes constituyentes de A y sus anotaciones semánticas correspondientes son $\alpha_j.sem, \dots, \alpha_k.sem$.

Esta notación es muy abstracta y, de hecho, para modelar las anotaciones semánticas es necesario utilizar algún lenguaje formal que permita representar el significado de una construcción y sus constituyentes. Es muy común utilizar lógica de primer orden y la notación de cálculo lambda para implementar las anotaciones semánticas y así, aumentar las reglas que componen la gramática.

Los principios que se utilizan para implementar la hipótesis *rule-to-rule*, es decir, anotar semánticamente las reglas de una gramática, utilizando lógica de primer orden y la notación de cálculo lambda son:

- 1 Asociar expresiones del cálculo lambda complejas, similares a una función, a las reglas léxicas.
- 2 Copiar el valor semántico del único constituyente a la construcción cuando la regla gramatical solo tiene un constituyente.
- 3 Aplicar la semántica de uno de los constituyentes de una regla gramatical a la semántica de otro de los constituyentes de la regla como si fuera una función.

Figura 3. Principios para implementar la hipótesis *rule-to-rule*. Fuente: elaboración propia.

Para explicar estos tres principios se utiliza un ejemplo porque las anotaciones semánticas de las reglas de una gramática son altamente dependientes del conocimiento sobre el ejemplo concreto para el que se va a representar el significado.

Si se quiere analizar semánticamente la frase

- Matías abrió un restaurante.

Es necesario aumentar una gramática libre de contexto con las correspondientes anotaciones semánticas. La gramática que define limitadamente el lenguaje natural del español y permite analizar la frase propuesta es la siguiente:

- a) NP → Matías
 - b) N → restaurante
 - c) Det → un
 - d) Vt → abrió
- 1) O → SN SV
 - 2) SN → NP
 - 3) SN → Det N
 - 4) SV → Vt SN

Donde:

- ▶ Las reglas de la *a*) a la *d*) son reglas léxicas, definen las categorías gramaticales y formarían parte del lexicón.
- ▶ Las reglas del 1 al 4 son reglas gramaticales y permiten definir la estructura sintagmática de la oración.

Primer principio sobre las anotaciones semánticas

Según el primer principio, las reglas léxicas se anotan con expresiones del cálculo lambda complejas. La regla *a*) hace referencia a que la palabra «Matías» es un nombre propio (*NP*). El significado de un nombre propio es el nombre en sí mismo. Entonces, en este caso, la anotación semántica más sencilla sería asignar a esta regla una constante, lo que representado en lógica de primer orden sería:

$$a) \text{NP} \rightarrow \text{Matías} \{\text{Matías}\}$$

El mismo principio se puede aplicar a otras reglas léxicas donde las anotaciones son más complejas. Por ejemplo, la regla *b*) indica que la palabra «restaurante» es un nombre (*N*). A nivel conceptual, un restaurante es un tipo de objeto porque puede haber múltiples restaurantes, todos con una serie de características comunes, con sus particularidades e identificados por diferentes nombres. Por lo tanto, un

restaurante en concreto será un objeto de la clase Restaurante, lo que se representa en lógica de primer orden como *Restaurante(r)*, donde *r* es un elemento de la clase *Restaurante*.

Sin embargo, para representar un elemento no especificado de la clase se utiliza una variable *r* en la notación lambda (λr). Entonces la regla quedaría anotada de la siguiente forma:

b) $N \rightarrow \text{restaurante} \{ \lambda r. \text{Restaurante}(r) \}$

Por consistencia con la anotación de la regla *b*), la regla *a*) se podría escribir de una forma más formal y utilizando variables de la siguiente forma:

a) $NP \rightarrow \text{Matías} \{ \lambda m. m(\text{Matías}) \}$

Además, expresiones aún más complejas en lógica de primer orden y en la notación de cálculo lambda son necesarias para anotar la regla *c*) y la regla *d*):

c) $\text{Det} \rightarrow \text{un} \{ \lambda P. \lambda Q. \exists x P(x) \wedge Q(x) \}$

d) $V_t \rightarrow \text{abrió} \{ \lambda w. \lambda z. w(\lambda x. \exists e \text{ Abierto}(e) \wedge$

$\text{PersonaQueAbre}(e, z) \wedge \text{CosaAbierta}(e, x)) \}$

Por ejemplo, en la regla *d*), que representa el verbo transitivo (V_t) «abrió»; se utiliza un objeto, *Abierto(e)*, y dos relaciones *PersonaQueAbre(e, z)* y *CosaAbierta(e, x)* para modelar el hecho de abrir *e*, la persona *z* que realiza la acción de abrir y la cosa *x* que se abre.

Segundo principio sobre las anotaciones semánticas

El segundo principio indica que en las reglas gramaticales se copia el valor semántico del único constituyente a la construcción. Una regla de este tipo es la regla 2), que indica que un único nombre propio (NP) puede constituir un sintagma nominal (SN).

Por lo tanto, aplicando este principio se copia el valor semántico del único constituyente a la construcción, es decir, que el significado del nombre propio (*NP.sem*) es el significado del sintagma nominal porque es el único elemento que lo compone.

Por ello, la regla anotada semánticamente sería la siguiente:

$$2) \text{SN} \rightarrow \text{NP} \{ \text{NP.sem} \}$$

Entonces, al seguir las reglas 2) y a), si en el análisis semántico se detecta la palabra «Matías» y se sabe que es un nombre propio (NP) y que conforma un sintagma nominal (SN), por el principio de composición se va a poder inferir que el significado del sintagma nominal es el significado del nombre propio. Por ende, el sintagma nominal (SN) «Matías» tiene como significado el concepto Matías, lo que se representaría formalmente como:

$$\text{SN.sem} = \text{NP.sem} = \{ \lambda m. m(\text{Matías}) \}$$

Tercer principio sobre las anotaciones semánticas

El tercer principio indica que, en las reglas gramaticales, se aplica la semántica de uno de los constituyentes de una regla a la semántica de otro de los constituyentes de la regla como si fuera una función. Por ejemplo, la regla 3) indica que un sintagma nominal (SN) está formado por un determinante (Det) y un nombre (N). Entonces, el significado del sintagma nominal es el resultado de aplicar el significado del determinante (*Det.sem*) al significado del nombre (*N.sem*), lo que genera la siguiente regla anotada:

$$3) \text{SN} \rightarrow \text{Det N} \{ \text{Det.sem} (\text{N.sem}) \}$$

A partir de las reglas 3), b) y c), y aplicando el principio de composición, es posible realizar la composición de los significados de la palabra «un» y la palabra

«restaurante» que forman el sintagma nominal «un restaurante» de la siguiente forma:

$$\begin{aligned} \text{Det.sem} &= \{ \lambda P. \lambda Q. \exists x P(x) \wedge Q(x) \} \\ N.sem &= \{ \lambda r. \text{Restaurante}(r) \} \\ \{ \text{Det.sem} (N.sem) \} &= \{ \lambda Q. \exists x \text{Restaurante}(x) \wedge Q(x) \} \end{aligned}$$

Por lo tanto, el sintagma nominal (SN) «un restaurante» tendría como significado $\{ \lambda Q. \exists x \text{Restaurante}(x) \wedge Q(x) \}$ en la oración a analizar, lo que se puede interpretar como que existe (\exists) un elemento x del tipo Restaurante.

El tercer principio se aplica de la misma forma a las reglas 1) y 4), dando como resultado las siguientes reglas anotadas:

- 1) O \rightarrow SN SV $\{ SN.sem (SV.sem) \}$
- 4) SV \rightarrow V_t SN $\{ Vt.sem (SN.sem) \}$

A partir de las reglas 4) y d) y la composición del significado del sintagma nominal (SN) «un restaurante» se puede obtener el significado del sintagma verbal (SV) «abrió un restaurante» de la siguiente forma:

$$\begin{aligned} Vt.sem &= \{ \lambda w. \lambda z. w(\lambda x. \exists e \text{Abierto}(e) \wedge \\ &\quad \text{PersonaQueAbre}(e, z) \wedge \text{CosaAbierta}(e, x)) \} \\ SN.sem &= \{ \text{Det.sem} (N.sem) \} = \{ \lambda Q. \exists x \text{Restaurante}(x) \wedge Q(x) \} \\ \{ Vt.sem (SN.sem) \} &= \{ \lambda z. (\exists x \text{Restaurante}(x) \wedge \exists e \text{Abierto}(e) \wedge \\ &\quad \text{PersonaQueAbre}(e, z) \wedge \text{CosaAbierta}(e, x)) \} \end{aligned}$$

El sintagma verbal (SV) «abrió un restaurante» tiene como significado la expresión:

$$\begin{aligned} &\{ \lambda z. (\exists x \text{Restaurante}(x) \wedge \exists e \text{Abierto}(e) \\ &\quad \wedge \text{PersonaQueAbre}(e, z) \wedge \text{CosaAbierta}(e, x)) \} \end{aligned}$$

Esto se puede interpretar como que existe (\exists) un elemento e del tipo *Abierto* que se relaciona a través de la relación *PersonaQueAbre* con un objeto (desconocido) z y que también se relaciona a través de la relación *CosaAbierta* con un elemento x del tipo *Restaurante*.

Para finalizar, a partir de la regla 1), del significado del sintagma nominal (SN) «Matías» y del significado del sintagma verbal (SV) «abrió un restaurante», computados previamente, se puede obtener por el principio de composición el significado de la oración «Matías abrió un restaurante» de la siguiente forma:

$$\begin{aligned} SN.sem &= NP.sem = \{ \lambda m. m(Matías) \} \\ SV.sem &= \{ Vt.sem (SN.sem) \} = \{ \lambda z. (\exists x Restaurante(x) \wedge \exists e \\ &\quad Abierto(e) \wedge PersonaQueAbre(e, z) \wedge CosaAbierta(e, x)) \} \\ O.sem &= \{ SN.sem (SV.sem) \} = \{ \exists x Restaurante(x) \wedge \exists e Abierto(e) \wedge \\ &\quad PersonaQueAbre(e, Matías) \wedge CosaAbierta(e, x) \} \end{aligned}$$

Entonces la oración (O) «Matías abrió un restaurante» tiene como significado la expresión:

$$\{ \exists x Restaurante(x) \wedge \exists e Abierto(e) \wedge \\ PersonaQueAbre(e, Matías) \wedge CosaAbierta(e, x) \}$$

Esta representación del significado de la frase se puede interpretar como que existe (\exists) un elemento e del tipo *Abierto* que se relaciona a través de la relación *PersonaQueAbre* con un objeto llamado *Matías* y que también se relaciona a través de la relación *CosaAbierta* con un elemento x del tipo *Restaurante*.

En resumen, la gramática libre de contexto anotada semánticamente, útil para que la información sintáctica representada en forma de reglas se complemente con información semántica y que permite realizar el análisis semántico de la oración del ejemplo anterior sería la siguiente:

- a) NP → Matías { $\lambda m. m(Matías)$ }
 - b) N → restaurante { $\lambda r. Restaurante(r)$ }
 - c) Det → un { $\lambda P. \lambda Q. \exists x P(x) \wedge Q(x)$ }
 - d) V_t → abrió { $\lambda w. \lambda z. w(\lambda x. \exists e Abierto(e)) \wedge$
 $PersonaQueAbre(e, z) \wedge CosaAbierta(e, x)$ }
- 1) O → SN SV { $SN.sem (SV.sem)$ }
 - 2) SN → NP { $NP.sem$ }
 - 3) SN → Det N { $Det.sem (N.sem)$ }
 - 4) SV → V_t SN { $Vt.sem (SN.sem)$ }

7.4. Análisis semántico integrado con el análisis

sintáctico

El análisis semántico se puede realizar en paralelo con el análisis sintáctico utilizando una gramática que incorpore la semántica junto con la representación sintáctica.

En el vídeo *Análisis semántico integrado con el análisis sintáctico* se verá la integración del análisis sintáctico con el análisis semántico a través de un ejemplo práctico.



Accede al vídeo

Las gramáticas libres de contexto, presentadas en la sección anterior, permiten realizar el análisis sintáctico y semántico simultáneamente. Esto marca una diferencia con la técnica explicada al inicio de este tema, que requiere del resultado del análisis sintáctico como entrada para poder realizar el análisis semántico.

La implementación del análisis semántico integrado con el sintáctico es posible gracias a que la representación de significado para una construcción se puede obtener tan pronto como todas sus partes constituyentes sean procesadas.

Entonces, la principal ventaja de utilizar este enfoque integrado radica en que si se encuentra una construcción semántica que no tiene sentido cuando se está creando la representación de significado, se puede considerar que el árbol sintáctico correspondiente tampoco va a ser válido.

Por tanto, las consideraciones semánticas que se tienen en cuenta durante el análisis sintáctico permiten simplificar este tipo de análisis.

Sin embargo, este hecho también conlleva asociado una de las principales desventajas de integrar la semántica directamente en el analizador sintáctico. Puede ser que se dedique un esfuerzo considerable al análisis semántico de componentes de la oración que al final no contribuyan a desarrollar un árbol sintáctico consistente. De esta forma, puede ser que se hayan realizado operaciones en el análisis semántico que son totalmente innecesarias para un análisis sintáctico exitoso.

Por desgracia no existe una respuesta a la cuestión de si la ganancia obtenida de incorporar la semántica ya desde el inicio en el análisis sintáctico compensa el coste de realizar algún procesamiento semántico innecesario. Por lo que no se puede afirmar que haya un enfoque mejor que otro para realizar el análisis semántico.

7.5. Referencias bibliográficas

Bach, E. (1976). An extension of classical transformational grammar. In Problems of Linguistic Metatheory (Proceedings of the 1976 Conference). Michigan State University.

Jurafsky, D. y Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall.

A fondo

SenSem

Vàzquez, G., Alonso, L., Capilla, J. A., Castellón, I. y Fernández, A. (2006). SenSem: sentidos verbales, semántica oracional y anotación de corpus. *Procesamiento del Lenguaje Natural*, 37, 113-119.
https://www.researchgate.net/publication/39435212_SenSem_sentidos_verbales_se_mantica_oracional_y_anotacion_de_corpus

El artículo presenta el proyecto SenSem que tenía como objetivo describir y representar el comportamiento léxico, sintáctico y semántico de los verbos del español. En el desarrollo de este proyecto construyeron dos recursos: un corpus de oraciones asociadas a su interpretación sintáctico-semántica y un léxico donde cada sentido verbal se asocia a un conjunto de ejemplos anotados del corpus.

Syntactic and Semantic Processing

Alshawi, H. (Ed.). (1992). Syntactic and semantic processing. *The Core Language Engine* (pp. 129-147). MIT Press.

Recomendamos la lectura del capítulo 7 de este libro. En él se describe cómo se realiza el análisis sintáctico y el análisis semántico en el Core Language Engine (CLE), un sistema para el procesamiento del lenguaje natural implementado en Prolog. En este sistema, el análisis semántico se basa en el paradigma del análisis semántico dirigido por la sintaxis

Test

1. Indica las afirmaciones correctas sobre el análisis semántico dirigido por la sintaxis:

 - A. Utiliza como entrada el análisis sintáctico de la oración.
 - B. Obtiene como resultado el árbol semántico de la oración.
 - C. El analizador semántico trata con el problema de la ambigüedad.
 - D. Se basa en el principio de composición.

2. Indica las afirmaciones correctas sobre el principio de composición:

 - A. El significado de una oración se obtiene directamente del orden y la agrupación de palabras y de las relaciones entre las palabras en la frase.
 - B. La idea fundamental es que el significado de una oración se construye a partir del significado de sus partes.
 - C. El significado de una oración se obtiene directamente del conjunto de los significados de las palabras que la componen.
 - D. El significado de una oración se basa en su estructura sintáctica.

3. Indica las afirmaciones correctas sobre una estructura típica en cascada para realizar el análisis semántico dirigido por la sintaxis:

 - A. Se realiza primero el análisis sintáctico y con su resultado el análisis semántico.
 - B. En la segunda etapa se obtiene la información relativa a las relaciones estructurales entre palabras.
 - C. En la primera etapa se obtiene el árbol sintáctico de la oración.
 - D. El resultado de la segunda etapa es la representación del significado de la oración, por ejemplo, utilizando lógica de primer orden.

4. Indica las afirmaciones correctas sobre el analizador semántico que interpreta el árbol sintáctico cuando se realiza el análisis semántico dirigido por la sintaxis con una estructura de dos etapas:

- A. Precisa de conocimiento genérico sobre los árboles sintácticos.
- B. Precisa de conocimiento sobre el ejemplo concreto o dominio de la frase que se analiza.
- C. Precisa de conocimiento concreto sobre el árbol sintáctico específico de la frase que se analiza.
- D. Es una buena praxis que se diseña específicamente para cada árbol sintáctico.

5. Indica las afirmaciones correctas sobre las anotaciones semánticas a las reglas gramaticales:

- A. Son instrucciones de cómo generar el significado de una oración a partir de los significados de las palabras que la componen.
- B. Permiten integrar el conocimiento semántico en el propio proceso de generación del árbol sintáctico.
- C. Representan el conocimiento semántico que permite realizar el análisis semántico guiado por la sintaxis.
- D. Se basan en la conocida como hipótesis *rule-to-rule*.

6. Indica las afirmaciones correctas sobre la notación utilizada para anotar semánticamente una regla libre de contexto:
- A. Las anotaciones semánticas se modelan utilizando algún lenguaje formal que permita representar el significado de una construcción y sus constituyentes.
 - B. La anotación semántica se añade a la regla en la parte derecha de los constituyentes sintácticos de la regla dentro de paréntesis.
 - C. La representación del significado de una construcción se puede calcular como una función aplicada a un subconjunto de las anotaciones semánticas de los constituyentes de la construcción.
 - D. La lógica de primer orden y la notación de cálculo lambda son el método más común para implementar las anotaciones semánticas.

Feedback.

7. Indica las afirmaciones correctas sobre el primero de los principios que se utilizan para anotar semánticamente las reglas de una gramática:
- A. Las reglas gramaticales se anotan con expresiones complejas de la lógica de primer orden y del cálculo lambda.
 - B. Las reglas léxicas se anotan con expresiones complejas de la lógica de primer orden y del cálculo lambda.
 - C. Una regla léxica se puede anotar con la siguiente representación del significado $\{\lambda x. C(x)\}$.
 - D. Una regla léxica se puede anotar con la siguiente representación del significado $\{\lambda x. x(x)\}$.

- 8.** Indica las afirmaciones correctas sobre el segundo de los principios que se utilizan para anotar semánticamente las reglas de una gramática:
- A. Las reglas gramaticales que tienen un único constituyente en la construcción se anotan copiando el valor semántico del constituyente.
 - B. Las reglas gramaticales que tienen varios constituyentes en la construcción se anotan copiando el valor semántico de uno de los constituyentes.
 - C. Se tiene la regla gramatical $A \rightarrow B \{ B.sem \}$ y la regla léxica $A \rightarrow b \{ b.sem \}$, entonces $A.sem = B.sem = b.sem$.
 - D. Se tiene la regla gramatical $A \rightarrow B \{ B.sem \}$ y la regla léxica $B \rightarrow b \{ b.sem \}$, entonces $A.sem = B.sem = b.sem$.
- 9.** Indica las afirmaciones correctas sobre el tercero de los principios que se utilizan para anotar semánticamente las reglas de una gramática:
- A. Las reglas gramaticales que tienen varios constituyentes en la construcción se anotan aplicando la semántica de uno de los constituyentes a la semántica de otro de los constituyentes como si fuera una función.
 - B. Las reglas gramaticales que tienen varios constituyentes en la construcción se anotan copiando el valor semántico de uno de los constituyentes.
 - C. La regla gramatical $A \rightarrow B C$ se puede anotar con la anotación semántica $\{ B.sem (C.sem) \}$.
 - D. La regla gramatical $A \rightarrow B C$ se puede anotar con la anotación semántica $\{ C.sem (B.sem) \}$.

10. Indica las afirmaciones correctas sobre el análisis semántico integrado con el análisis sintáctico:

- A. El análisis semántico requiere del resultado del análisis sintáctico como entrada.
- B. El análisis sintáctico y el análisis semántico se realizan simultáneamente en paralelo.
- C. Si se encuentra una construcción semántica que no tiene sentido, se puede considerar que el árbol sintáctico correspondiente tampoco va a ser válido.
- D. Se utiliza una gramática anotada semánticamente para realizar el análisis sintáctico.

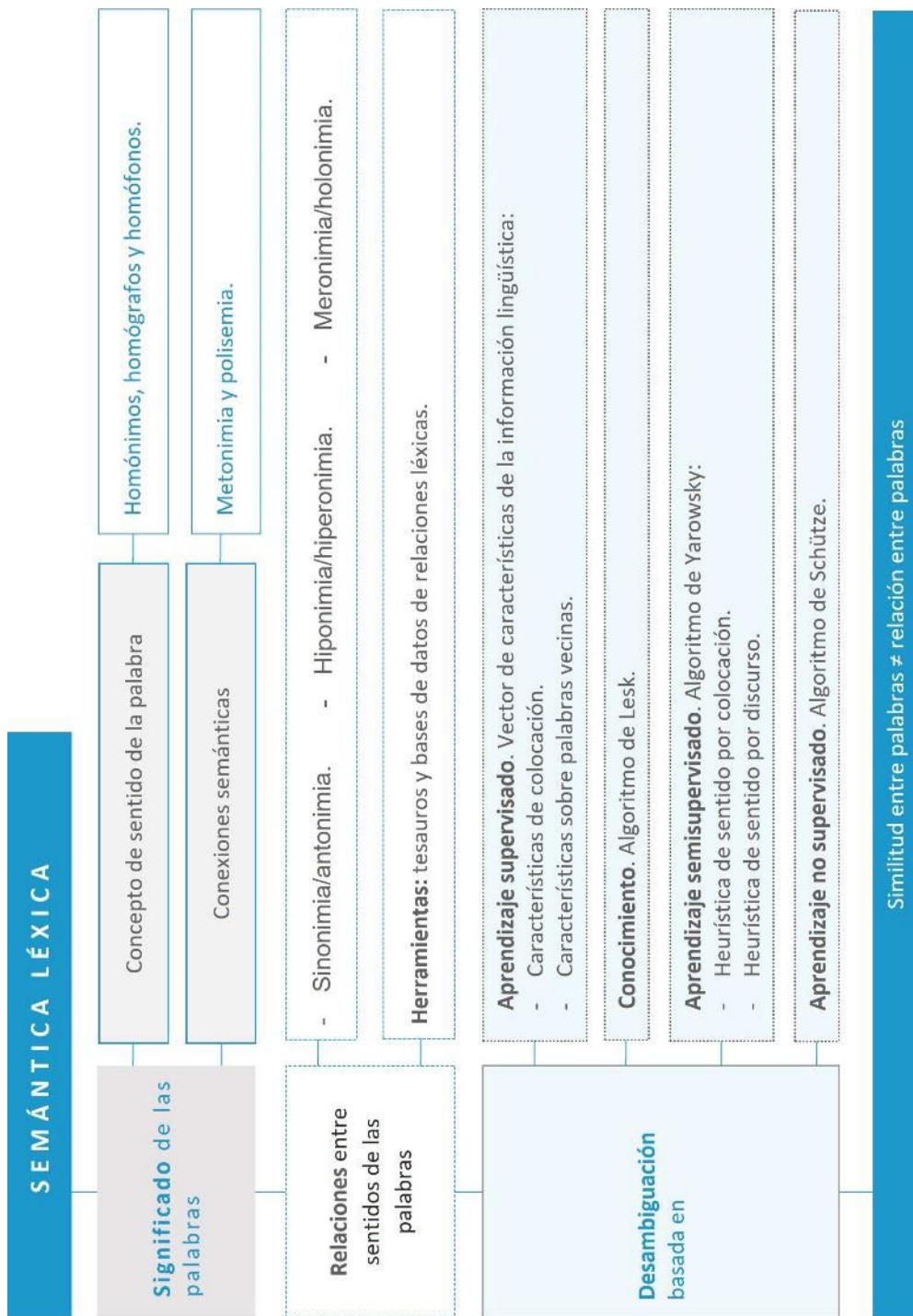
Procesamiento del Lenguaje Natural

Semántica léxica

Índice

Esquema	3
Ideas clave	4
8.1. Introducción y objetivos	4
8.2. Significado de las palabras	4
8.3. Relaciones entre sentidos de las palabras	8
8.4. Desambiguación del sentido de las palabras	11
8.5. Similitud entre palabras	24
8.6. Referencias bibliográficas	29
A fondo	33
Test	35

Esquema



8.1. Introducción y objetivos

Los objetivos al finalizar el estudio de este tema son:

- ▶ Definir los conceptos relevantes en el ámbito de la semántica léxica, campo de la lingüística que estudia el significado de las palabras.
- ▶ Identificar los diferentes tipos de relaciones entre los sentidos y significados de las palabras: homonimia, metonimia, sinonimia, antonimia, hiponimia, etc.
- ▶ Desambiguar el sentido de las palabras aplicando técnicas de aprendizaje automático, ya sean de aprendizaje supervisado, no supervisado o semisupervisado.
- ▶ Calcular la similitud entre palabras utilizando tesauros.

8.2. Significado de las palabras

La Real Academia Española (RAE, s. f. a) en su diccionario de la lengua española define **semántica léxica** como: «1. f. Rama de la semántica que estudia el significado de las palabras, así como las diversas relaciones de sentido que se establecen entre ellas».

La forma más sencilla de representar el significado de una palabra sería utilizando el **lema**. Tal como se vio anteriormente, **se llama lema a la raíz de la palabra** y, por lo tanto, es la forma gramatical de la palabra que se utiliza en los diccionarios. Por ejemplo, para la palabra *bancos* el lema es la forma «banco».

Un lema puede tener varios significados: en el caso del lema «banco» puede referirse al tipo de asiento donde pueden sentarse varias personas o a la empresa que realiza

operaciones financieras, entre otras definiciones que aparecen en el diccionario de la RAE. A cada uno de los significados del lema «banco» se le llama un sentido de la palabra o simplemente sentido.

Un sentido es una representación de uno de los aspectos del significado de una palabra.

Cada uno de los sentidos de una palabra se puede representar añadiendo un superíndice a la forma ortográfica del lema. Entonces, los dos sentidos de la palabra «banco» se representarían como «banco¹» y «banco²», respectivamente. Los diferentes sentidos de una palabra (en el ejemplo, el banco como un tipo de asiento («banco¹») y como una entidad financiera («banco²»)) normalmente no tienen ninguna relación a nivel de significado entre sí, por eso se dice que «banco¹» y «banco²» son homónimos, término que se aplica a cosas que tienen el mismo nombre.

Si una palabra tiene varios sentidos que no están relacionados entre sí, se dice que los sentidos son homónimos y a la relación entre dos de estos sentidos se le llama relación de homonimia.

Además, en el caso del *banco* como un tipo de asiento («banco¹») y como una entidad financiera («banco²») se escriben de la misma forma y, por lo tanto, se dice que estos dos usos son **homógrafos**: según la RAE (s. f. b), una palabra «que tiene la misma grafía que otra».

Existe también otra forma en que dos palabras pueden ser homónimas, sería el caso de palabras que se escriben de forma diferente, pero que se pronuncian igual. Por ejemplo, la palabra *vello* (en referencia al pelo del cuerpo) y la palabra *bello* (hermoso) son **homónimas**: según la RAE (s. f. c), una palabra «que se pronuncia como otra, pero tiene diferente origen o significado muy distante».

Además, *vello* (pelo del cuerpo) y *bello* (hermoso) son dos lemas que suenan igual por lo que se dice que son **homófonos**: según la RAE (s. f. d), una palabra «que **suena igual que otra, pero que tiene distinto significado y puede tener distinta grafía**».

- ▶ La homofonía es una de las causas de que se cometan errores ortográficos en la vida real y que también afecta el procesamiento del lenguaje natural cuando se realiza el reconocimiento automático del habla.
- ▶ De forma similar, la homografía afecta a la tarea de conversión de texto al habla.
- ▶ La homonimia es la principal causa de diferentes errores que se dan en el procesamiento del lenguaje natural. Para tratarla se aplican técnicas de desambiguación del sentido de la palabra.

La desambiguación del sentido de la palabra es la tarea de determinar qué sentido de una palabra se usa en un contexto particular.

A diferencia de lo que se acaba de exponer, **a veces existe una conexión semántica entre los sentidos de las palabras**. Volviendo al ejemplo de la palabra «banco», puede existir un tercer significado que sería la oficina donde opera la entidad financiera («banco³»). Este último significado sí está relacionado con «banco²» (la entidad financiera); estos dos sentidos, «banco²» y «banco³», tienen una conexión semántica. Sin embargo, «banco³» (la oficina de la entidad financiera) sigue sin tener ninguna relación semántica con «banco¹» (el tipo de asiento).

Por otro lado, **la oficina bancaria («banco³»)** coge el nombre de la propia organización, es decir, **la entidad financiera («banco²»)**. Lo mismo pasa con otros ejemplos como la palabra «universidad», que hace referencia a la institución de enseñanza superior, «universidad¹», pero también a los edificios de las cátedras y oficinas de la institución universitaria, «universidad²». **Este tipo de relación semántica entre los sentidos de las palabras se llama metonimia.**

La metonimia es una forma de emplear una palabra en un sentido distinto al que propiamente le corresponde, pero con el que tiene algún tipo de conexión.

Concretamente, la metonimia «consiste en designar algo con el nombre de otra cosa tomando el efecto por la causa o viceversa, el autor por sus obras, el signo por la cosa significada, etc. Por ej., las canas por la vejez; leer a Virgilio, por leer las obras de Virgilio» (RAE, s. f. e). Entonces, el sentido «banco³» (la oficina de la entidad financiera) tiene una **relación de metonimia** con «banco²» (la entidad financiera), así como «universidad¹» (institución de enseñanza superior) con la «universidad²» (los edificios de las cátedras y oficinas de la institución universitaria).

La relación semántica de la metonimia es un tipo particular de polisemia, el nombre genérico que se utiliza para definir cualquier relación semántica entre dos significados, ya que la palabra polisemia se refiere a la pluralidad de significados.

Si una palabra tiene varios sentidos que están relacionados entre sí, se dice que los sentidos son polisémicos y a la relación entre dos de estos sentidos se le llama relación de polisemia.

En el vídeo *Significado de las palabras* nos centraremos en la semántica léxica, la parte de la lingüística que estudia el significado de las palabras y las relaciones de sentido que se forman en torno a ellas.



Accede al vídeo

Los diccionarios son los repertorios donde se recogen, según un orden determinado, las palabras de una lengua acompañadas de su definición o explicación. Incluyen una descripción detallada y comprensible para un humano de los diferentes sentidos de las palabras. En el procesamiento del lenguaje natural, el formato que siguen los

diccionarios tradicionales no es el más útil para obtener y entender los significados de las palabras.

- PARA EL PROCESAMIENTO AUTOMÁTICO ES NECESARIO**
1. Definir el sentido de una palabra a través de su relación con los sentidos de otras palabras.
 2. Agrupar los sentidos de las palabras.

Figura 1. Requisitos a la hora de elaborar un repertorio de los sentidos de las palabras en PLN. Fuente: elaboración propia.

Por ejemplo, para el procesamiento del lenguaje natural puede ser útil definir que «rojo» y «azul» son lemas del mismo tipo, concretamente del tipo color. Además de establecer que pertenecen al mismo grupo y son complementarios, lo que indica que algo que es rojo no puede ser azul. También puede ser interesante definir que la sangre es roja y el mar es azul. Si se tiene una base de datos suficientemente exhaustiva con estas relaciones entre los sentidos de las palabras será posible realizar tareas semánticas muy sofisticadas.

8.3. Relaciones entre sentidos de las palabras

Las relaciones más comunes que se dan entre los sentidos de las palabras son la **sinonimia** y la **antónima**. También destacan la **hipónima** y la **hiperónima**, y, aunque sea menos común, la **merónima**.

Cuando los sentidos de dos palabras diferentes y, por extensión, los significados de sus dos lemas son idénticos o casi idénticos, se puede decir que hay una relación de **sinonimia** entre ambos. Por ejemplo, las palabras *empezar* y *comenzar* son sinónimos ya que, si se intercambian en una frase, esta mantiene prácticamente el mismo significado.

Una palabra es un sinónimo de otra si tiene el mismo significado o un significado muy parecido.

La **antonimia** es la relación contraria. Los antónimos son palabras que tienen significados opuestos como, por ejemplo, *claro* y *oscuro* o *antes* y *después*. Dos sentidos pueden ser antónimos si definen una oposición binaria o están en extremos opuestos de alguna escala. Este es el caso de *largo* y *corto*, *rápido* y *lento* o *grande* y *pequeño*, porque se encuentran en extremos opuestos de la escala de longitud, de tiempo y de tamaño, respectivamente. Además, se puede definir un grupo de antónimos llamados **reversos**, que describen cambios o movimientos en direcciones opuestas como *abierto* y *cerrado* o *subir* y *bajar*.

Una palabra es un antónimo de otra si expresa una idea opuesta o contraria.

Un sentido es un hipónimo de otro si el primer sentido es más específico que el segundo, es decir, si el primero es una subclase del segundo. Por ejemplo, *coche* es hipónimo de *vehículo*; *perro* lo es a *animal* y *mango* lo es a *fruta* porque el coche es un tipo de vehículo, el perro un tipo de animal y el mango un tipo de fruta.

Una palabra es un hipónimo de otra si su significado incluye el significado de la otra palabra.

La relación contraria a la hiponimia es la **hiperonimia**. Entonces, se dice que *vehículo* es hiperónimo de *coche*, *animal* es hiperónimo de *perro* y *fruta* es hiperónimo de *mango* porque la clase que representa los vehículos incluye como miembros a todos los coches, la que representa a los animales incluye a los perros y la que representa a las frutas incluya a los mangos.

Una palabra es un hiperónimo de otra si su significado está incluido en el significado de la otra palabra.

Formalmente, se define que un sentido *A* es hipónimo de un sentido *B* si todo lo que es *A* también es *B*, entonces ser un *A* implica ser un *B*, lo que en lógica de primer orden se escribiría como:

$$\forall x A(x) \Rightarrow B(x)$$

La hiponimia suele ser una **relación transitiva**, si *A* es hipónimo de *B* y *B* es un hipónimo de *C*, entonces *A* es un hipónimo de *C*.

En las ontologías, representaciones semánticas basadas en lógica descriptiva, se modelan las relaciones de hiponimia e hiperonimia a través de la jerarquía IS-A. La expresión *A* IS-A *B* indica que un sentido *A* es hipónimo de un sentido *B*, o que un sentido *B* es hiperónimo de un sentido *A*.

Una última relación destacada es la **meronimia** que define una relación del tipo **parte-todo** entre los sentidos de las palabras. Por ejemplo, una rueda es una parte de un coche, por lo que se dice que *rueda* es merónimo de *coche* o, al contrario, que *coche* es **holónimo** de *rueda*.

Una palabra es un merónimo de otra si su significado mantiene con el significado de la otra palabra una relación de la parte respecto al todo.

En el vídeo *Relaciones entre sentidos de las palabras* se estudiarán las diferentes relaciones entre los sentidos que pueden tener las palabras.



Accede al vídeo

Para poder realizar tareas de procesamiento del lenguaje natural es necesario recoger las relaciones entre los sentidos de las palabras de un diccionario. De hecho, en literatura se utiliza el término **tesauro**, *thesaurus*, *thesauri* o **tesoro** para referirse a los **diccionarios que contienen una lista de palabras con sus sinónimos y sus**

antónimos. En lingüística, los términos que conforman un tesauro se relacionan entre sí para mostrar las relaciones entre significados.

Las llamadas **bases de datos de relaciones léxicas** contienen un conjunto de lemas, cada uno de los cuales está anotado con el posible conjunto de sentidos de la palabra. Cada uno de los sentidos contiene, además de su definición en un formato tipo glosario y una lista de sinónimos. No es obligatorio que las bases de datos de relaciones léxicas contengan la pronunciación los lemas, sin embargo, es imprescindible que estas contengan detalles de las relaciones entre lemas.

La Figura 1 muestra un ejemplo de algunas relaciones entre sentidos en WordNet (Fellbaum, 1998), la que es la principal base de datos léxica para procesamiento del lenguaje natural en inglés.

Relation	Also Called	Definition	Example
Hypernym	Superordinate	From concepts to superordinates	<i>breakfast</i> ¹ → <i>meal</i> ¹
Hyponym	Subordinate	From concepts to subtypes	<i>meal</i> ¹ → <i>lunch</i> ¹
Instance Hypernym	Instance	From instances to their concepts	<i>Austen</i> ¹ → <i>author</i> ¹
Instance Hyponym	Has-Instance	From concepts to concept instances	<i>composer</i> ¹ → <i>Bach</i> ¹
Member Meronym	Has-Member	From groups to their members	<i>faculty</i> ² → <i>professor</i> ¹
Member Holonym	Member-Of	From members to their groups	<i>copilot</i> ¹ → <i>crew</i> ¹
Part Meronym	Has-Part	From wholes to parts	<i>table</i> ² → <i>leg</i> ³
Part Holonym	Part-Of	From parts to wholes	<i>course</i> ⁷ → <i>meal</i> ¹
Substance Meronym		From substances to their subparts	<i>water</i> ¹ → <i>oxygen</i> ¹
Substance Holonym		From parts of substances to wholes	<i>gin</i> ¹ → <i>martini</i> ¹
Antonym		Semantic opposition between lemmas	<i>leader</i> ¹ ⇔ <i>follower</i> ¹
Derivationally		Lemmas w/same morphological root	<i>destruction</i> ¹ ⇔ <i>destroy</i> ¹
Related Form			

Figura 2. Relaciones entre sentidos en WordNet. Fuente: Jurafsky y Martin, 2009.

8.4. Desambiguación del sentido de las palabras

Los métodos para el análisis semántico composicional que se han estudiado en el tema anterior no tienen en cuenta que una palabra puede tener más de un significado. De hecho, esos métodos obvian la ambigüedad léxica que es una realidad existente y es un problema importante en el procesamiento del lenguaje natural.

La desambiguación del sentido de las palabras es la tarea de seleccionar el sentido correcto para una palabra. Los algoritmos de desambiguación del sentido toman como entrada una palabra en su contexto y una lista de posibles significados de esa palabra y devuelven como salida el sentido correcto para ese uso concreto de la palabra.

Existen diferentes opciones para implementar un algoritmo de desambiguación del sentido de las palabras. La primera opción se basa en aplicar técnicas de aprendizaje automático supervisado para aprender un modelo clasificador que permita desambiguar las palabras. Estos algoritmos de desambiguación basados en aprendizaje supervisado requieren tener un corpus de palabras etiquetadas con sus sentidos correctos para poder entrenar el clasificador. Tener acceso a este tipo de datos etiquetados puede ser complejo y es por eso por lo que, aunque los algoritmos de desambiguación basados en aprendizaje supervisado sean los que proporcionan mejores resultados, a veces no se utilizan por el elevado coste asociado a la obtención de los datos etiquetados.

Si no se dispone de un corpus etiquetado, una alternativa es utilizar diccionarios, tesauros u otras bases de conocimiento para realizar un entrenamiento indirecto, aplicando algoritmos de aprendizaje supervisado débil.

Este tipo de métodos de desambiguación se llaman algoritmos de desambiguación basados en conocimiento y no utilizan textos que hayan sido etiquetados manualmente, sino que utilizan grandes bases de conocimiento.

Otra opción a los algoritmos de desambiguación basados en aprendizaje supervisado y a los basados en conocimiento es aplicar técnicas de aprendizaje semisupervisado o bootstrapping. Estos algoritmos de desambiguación basados en aprendizaje semisupervisado no requieren de grandes recursos lingüísticos generados a mano, es decir, ni de un gran conjunto de datos de entrenamiento ni de un gran diccionario,

sino que, para funcionar es suficiente con tener un pequeño conjunto de datos de entrenamiento etiquetados a mano.

Una última alternativa, que pretende evitar la tarea compleja y costosa de construir un corpus de palabras etiquetadas con los sentidos para la desambiguación de las palabras, se basa en técnicas de aprendizaje no supervisado.

Los algoritmos basados en este tipo de aprendizaje realizan la desambiguación sin utilizar definiciones de los sentidos de las palabras por humanos.

El conjunto de sentidos de cada palabra se crea automáticamente a partir de las instancias de la palabra disponibles en los datos de entrenamiento. Esta tarea se llama también inducción del sentido de las palabras.

Desambiguación basada en aprendizaje supervisado

Los algoritmos de desambiguación basados en el aprendizaje automático supervisado utilizan un conjunto de instancias etiquetadas para entrenar un clasificador. Una vez entrenado, este sirve para predecir el mejor sentido de las palabras ambiguas. Por lo tanto, el resultado del entrenamiento es un modelo clasificador capaz de asignar etiquetas de sentido a las palabras no etiquetadas que aparecen en un contexto determinado.

En una primera opción de implementación de los algoritmos de aprendizaje supervisado para aprender desambiguaciones, se parte de un conjunto de palabras ambiguas y se empieza preseleccionando un subconjunto de sentidos posibles para estas. Así, para cada palabra, se selecciona un número de instancias de la aparición de esa palabra en un corpus. Estas instancias se etiquetan a mano con el sentido correcto, según el contexto en el que aparecen en el corpus. Una vez se tienen todos los ejemplos etiquetados, estos se utilizan para entrenar el clasificador.

Como en este caso se trabaja con un conjunto reducido de palabras y el conjunto reducido de sentidos, se dice que el algoritmo de aprendizaje supervisado se entrena con una **muestra léxica**, con lo que aprende un modelo clasificador que permite desambiguar una única palabra o, como máximo, un conjunto reducido de palabras concretas, algo que puede ser poco práctico.

En una **segunda opción** de implementación de los algoritmos de aprendizaje supervisado, se utiliza un corpus de palabras ya etiquetadas con su sentido para aprender la desambiguación de un texto entero. Es decir, que se aprende a desambiguar todas las palabras del texto y no solo algunas palabras concretas, como era el caso de utilizar una muestra léxica.

El algoritmo de desambiguación que utiliza un **lexicón** para entrenar requiere de un conjunto muy grande de etiquetas porque cada lema tiene su propio set de etiquetas. Gestionar este conjunto enorme puede representar un problema y reducir la aplicabilidad de estas técnicas de aprendizaje supervisado basadas en un corpus etiquetado.

Los algoritmos de desambiguación basados en el aprendizaje supervisado ya sean los que para entrenar utilizan una muestra léxica etiquetada a mano o los que utilizan un corpus ya etiquetado, tienen que extraer características de texto y, a partir de ellas, asignar etiquetas con el sentido correcto.

Entonces, el primer paso en los algoritmos de aprendizaje supervisado es extraer características que sean predictivas de los sentidos de las palabras.

Los algoritmos modernos de desambiguación del sentido extraen información del contexto que rodea la palabra a desambiguar (Weaver, 1955). Para ello se utiliza una ventana que incluye la palabra ambigua y las palabras anteriores y posteriores, también llamadas palabras de contexto, y se extrae un **vector de características de la información lingüística** de las palabras de contexto contenidas en la ventana.

Dicho vector consta de valores numéricos o nominales y contiene dos tipos de características:

- ▶ Características de colocación.
- ▶ Características sobre las palabras vecinas.

Características de colocación

Codifican información sobre la posición de las palabras de contexto y su relación con la palabra ambigua. Algunas características típicas que se pueden extraer de las palabras de contexto incluyen la palabra en sí, la raíz de la palabra y su categoría gramatical o morfosintáctica.

Así, estas características codifican información léxica y gramatical que a menudo puede ayudar a identificar con precisión el sentido de la palabra.

Por ejemplo, un vector de características de colocación extraído de una ventana con dos palabras a la derecha y a la izquierda de la palabra objetivo, donde las características son las palabras, sus categorías gramaticales y los pares de palabras, tendría la siguiente forma:

$$[w_{i-2}, POS_{i-2}, w_{i-1}, POS_{i-1}, w_{i+1}, POS_{i+1}, w_{i+2}, POS_{i+2}, w_{i-2}^{i-1}, w_i^{i+1}]$$

Donde:

- ▶ w_{i-2} es la palabra dos posiciones a la izquierda de la palabra ambigua.
- ▶ POS_{i-2} es la categoría gramatical de la palabra w_{i-2} .
- ▶ w_{i-2}^{i-1} es el par de palabras w_{i-2} y w_{i-1} .

Ejemplo ilustrativo 1

Vector de características de colocación para desambiguar la palabra «*bass*» (bajo) en la oración:

- ▶ An electric guitar and bass player stand off to one side, not really part of the scene, just as a sort of nod to gringo expectations perhaps (del corpus WSJ).

Se define una ventana con dos palabras a la derecha y a la izquierda de la palabra «*bass*»:

<i>guitar</i>	<i>and</i>	<i>bass</i>	<i>player</i>	<i>stand</i>
w_{i-2}	w_{i-1}		w_{i+1}	w_{i+2}

Tabla 1. Ventana con la palabra «*bass*» como palabra objetivo. Fuente: elaboración propia.

Para calcular algunas de las características se realiza el análisis morfosintáctico y se identifican las siguientes categorías gramaticales de las palabras de contexto:

<i>guitar</i>	<i>and</i>	<i>bass</i>	<i>player</i>	<i>stand</i>
NN	CC		NN	VB

Tabla 2. Categorías gramaticales de las palabras contexto. Fuente: elaboración propia.

Como características se utilizan las propias palabras de contexto, sus categorías gramaticales y los pares de palabras posicionadas antes y después de la palabra «*bass*». El vector de características de colocación sería:

[*guitar*, NN, *and*, CC, *player*, NN, *stand*, VB, *and* *guitar*, *player* *stand*]

Características sobre las palabras vecinas

El segundo tipo de características codifican información sobre la llamada *bag-of-words* (bolsa de palabras), es decir, sobre un conjunto de palabras sin considerar su orden y que rodean a la palabra ambigua. Las características que se extraen de una bolsa de palabras son eficaces para capturar el tema general del discurso en el que

ocurre la palabra ambigua. Además, el tema del discurso tiende a identificar a su vez los sentidos de las palabras que son específicas en un cierto contexto.

En el enfoque más simple de bolsa de palabras, el contexto de una palabra ambigua se representa como un vector de características donde cada característica binaria indica si una palabra de un vocabulario aparece o no en el contexto. El vocabulario lo conforman un subconjunto de palabras útiles o de uso frecuente y preseleccionadas del corpus de entrenamiento. La región de contexto se define como una ventana de tamaño fijo donde la palabra ambigua se encuentra en el centro y que contiene un número generalmente pequeño de palabras situadas a la izquierda y a la derecha de la palabra ambigua.

Ejemplo ilustrativo 2

Vector de características sobre las palabras vecinas para desambiguar la palabra *bass* (bajo) en la oración:

- ▶ An electric guitar and bass player stand off to one side, not really part of the scene, just as a sort of nod to gringo expectations perhaps (del corpus WSJ).

Se define la siguiente *bag-of-words* (bolsa de palabras) con las doce palabras que aparecen más frecuentemente en frases donde se encuentra la palabra *bass* en el corpus WSJ:

[fishing, big, sound, player, fly, rod, pound, double, runs, playing, guitar, band]

Se utiliza una ventana de tamaño nueve: ventana que incluye cuatro palabras a la izquierda y cuatro a la derecha de la palabra ambigua. Por lo tanto, las palabras que quedarían dentro de la ventana son:

An electric guitar and bass player stand off to

En la ventana aparecen las palabras «*guitar*» y «*player*» y no aparecen ninguna de las otras palabras que forman parte de la bolsa (*bag-of-words*). Entonces el vector de características sobre las palabras vecinas sería:

[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]

Cualquier conjunto de datos de entrenamiento puede servir para extraer tanto características de colocación como características sobre las palabras vecinas y entrenar un clasificador de sentidos de las palabras con un algoritmo de aprendizaje automático supervisado. Sin embargo, el buen funcionamiento del clasificador va a depender de la cantidad de datos de entrenamiento con los que se cuente, es decir, de la disponibilidad de datos etiquetados con los sentidos. Debido a este problema, algunos investigadores han empezado a usar la Wikipedia como fuente de datos de entrenamiento (Mihalcea, 2007) (Ponzetto y Navigli, 2010).

Desambiguación basada en conocimiento

Los algoritmos de desambiguación que utilizan bases de conocimiento ya sean diccionarios o tesauros, para realizar un entrenamiento indirecto aplican algoritmos de aprendizaje supervisado débil.

- ▶ En el caso de utilizar diccionarios, se aplica el **algoritmo de Lesk**, que se presenta a continuación.
- ▶ En el caso de utilizar un tesauro, se aplican **métodos basados en gráficos** (Agirre, López de Lacalle y Soroa, 2014) (Navigli y Lapata, 2010).

El llamado **algoritmo de Lesk** se refiere en realidad a una familia de algoritmos que seleccionan el sentido de la palabra para el cual su definición en el diccionario comparte la mayor cantidad de palabras con los vecinos de la palabra ambigua.

La versión más simple del algoritmo de Lesk llamada **algoritmo de Lesk simplificado** (Kilgarriff y Rosenzweig, 2000) compara la firma de la palabra ambigua (*signature*), que se corresponde con su definición en el diccionario, con las palabras de contexto

(*context*), es decir, con las palabras vecinas a la palabra ambigua. El pseudocódigo se presenta en la Figura 3, donde la función COMPUTOVERLAP devuelve el número de palabras que tienen en común los conjuntos de palabras *signature* y *context*.

```
function SIMPLIFIED LESK(word, sentence) returns best sense of word
    best-sense ← most frequent sense for word
    max-overlap ← 0
    context ← set of words in sentence
    for each sense in senses of word do
        signature ← set of words in the gloss and examples of sense
        overlap ← COMPUTOVERLAP(signature, context)
        if overlap > max-overlap then
            max-overlap ← overlap
            best-sense ← sense
    end
    return(best-sense)
```

Figura 3. Pseudocódigo del algoritmo de Lesk Simplificado. Fuente: Jurafsky y Martin, 2009.

Ejemplo ilustrativo 3

Funcionamiento del algoritmo de Lesk simplificado para desambiguar la palabra «*bank*» (banco) en la oración

- ▶ The bank can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.

Del diccionario WordNet se obtienen las siguientes definiciones para la palabra «*bank*»:

bank ¹	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	"he cashed a check at the bank"; "that bank holds the mortgage on my home"
bank ²	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	"they pulled the canoe up on the bank"; "he sat on the bank of the river and watched the currents"

Figura 4. Definiciones de la palabra «*bank*» en WordNet. Fuente: WordNet, s. f.

Se observa que en la definición y los ejemplos del sentido «*bank¹*» aparecen dos palabras, «*deposits*» y «*mortgage*», que también aparecen en el contexto de la palabra a desambiguar, es decir, en la frase bajo análisis.

Por el contrario, ni en la definición ni en los ejemplos del sentido «bank²» aparecen palabras relevantes que también aparezcan en el contexto de la palabra a desambiguar. Es importante notar que los determinantes (por ejemplo, «the»), preposiciones, conjunciones y pronombres no se tienen en cuenta por no aportar información relevante.

Por lo tanto, el algoritmo de Lesk simplificado escogería el sentido «bank¹», que hace referencia a la entidad financiera, como sentido correcto para la palabra «bank» en la frase que se ha analizado.

En el vídeo *Algoritmo de Lesk simplificado* se verá cómo busca desambiguar el sentido de una palabra en una oración utilizando el contexto de esta.



Accede al vídeo

La versión original del algoritmo de Lesk (Lesk, 1986) utiliza para el entrenamiento una experiencia incluso más indirecta que la versión simple del algoritmo. De hecho, la versión original lo que hace es comparar la firma de la palabra ambigua con las firmas de cada una de las palabras del contexto.

En el vídeo *Versión original del algoritmo de Lesk* se estudiará un ejemplo del funcionamiento de este algoritmo.



Accede al vídeo

El principal problema de la versión original y la versión simplificada del algoritmo de Lesk es que las entradas del diccionario para las palabras ambiguas son cortas y puede que no se dé la oportunidad de que se solapen con las palabras del contexto (Lesk, 1986).

La solución es añadir todas las palabras en las oraciones del corpus etiquetado para un sentido de la palabra al cómputo de la firma de ese sentido. Esta versión del

algoritmo se llama **Corpus Lesk** y es la que proporciona mejores resultados (Kilgarriff y Rosenzweig, 2000) (Vasilescu, Langlais y Lapalme, 2004). Además, el algoritmo de **Corpus Lesk** lo que hace es **aplicar un peso a cada palabra coincidente en lugar de contar el número de palabras**. Aplicar un peso sirve para dar menos relevancia a las palabras poco importantes, pero que se repiten mucho como, por ejemplo, **los determinantes, preposiciones, conjunciones y pronombres**.

Para calcular los pesos en el algoritmo de Corpus Lesk se utiliza una medida llamada **inverse document frequency (IDF)** y que se define para la palabra i según la siguiente fórmula:

$$idf_i = \log \frac{N_{doc}}{nd_i}$$

Donde:

- ▶ N_{doc} es el número total de «documentos», es decir, definiciones y ejemplos.
- ▶ nd_i es el número de estos documentos que contienen la palabra i .

Se puede combinar el algoritmo de Lesk y los enfoques de aprendizaje automático supervisado añadiendo nuevas características similares a las computadas con el algoritmo de Lesk a la bolsa de palabras. Por ejemplo, las definiciones y las frases de ejemplo para el sentido de una palabra del diccionario se podrían usar en la bolsa de palabras utilizada en el aprendizaje supervisado (Yuret, 2004).

Desambiguación basada en aprendizaje semisupervisado

Los algoritmos de desambiguación basados en aprendizaje semisupervisado o **bootstrapping** no requieren de grandes recursos lingüísticos generados a mano, sino que para funcionar solo necesitan un pequeño conjunto de datos de entrenamiento etiquetados a mano que se irá ampliando durante el proceso de aprendizaje.

Uno de estos algoritmos más conocidos es el **algoritmo de Yarowsky**, que permite **aprender un clasificador para una palabra ambigua** (Yarowsky, 1995). El algoritmo de Yarowsky parte de un conjunto pequeño de datos de entrenamiento etiquetados (Λ_0) que contiene instancias para cada sentido de la palabra y un corpus no etiquetado mucho más grande (V_0).

En la primera iteración, el algoritmo de Yarowsky entrena un clasificador utilizando las instancias etiquetadas de Λ_0 y, a continuación, utiliza este clasificador para etiquetar el corpus no etiquetado V_0 . El algoritmo selecciona entonces los ejemplos de V_0 para los que tiene mayor confianza en la clasificación, los elimina de V_0 (pasando a llamarse ahora el conjunto de datos no etiquetados V_1) y los añade al conjunto de datos de entrenamiento (que ahora pasa a llamarse Λ_1).

En la siguiente iteración, el algoritmo entrena un nuevo clasificador con Λ_1 , utiliza el clasificador para etiquetar V_1 y extrae un nuevo conjunto de datos de entrenamiento Λ_2 , y así sucesivamente. Con cada iteración, el corpus de los datos de entrenamiento Λ crece y el corpus de datos sin etiquetar V disminuye. El proceso se repite hasta que se alcanza una tasa de error suficientemente baja o hasta que no quedan más ejemplos no etiquetados.

El conjunto inicial de datos de entrenamiento Λ_0 se puede etiquetar a mano, escogiendo un conjunto de instancias aleatorias del corpus no etiquetado V_0 (Hearst, 1991) o se puede hacer uso de la ayuda de una heurística para seleccionar las mejores instancias (Yarowsky, 1995).

El algoritmo original de Yarowsky propone dos posibles heurísticas: la de un sentido por colocación y la de un sentido por discurso.

Heurística de un sentido por colocación

Se basa en la idea de que ciertas palabras o frases muy relacionadas con el sentido analizado no tienden a ocurrir con los otros sentidos. Entonces se define el conjunto

de datos de entrenamiento escogiendo una única colocación para cada uno de los sentidos.

Heurística de un sentido por discurso

Se basa en la idea de que una palabra ambigua que aparece varias veces en un texto o discurso aparece a menudo con el mismo sentido (Gale, Church y Yarowsky, 1992). Esta heurística proporciona mejores resultados para los casos de homonimia que para los de polisemia (Krovetz, 1998) y es muy importante en el procesamiento del lenguaje porque muchas veces las tareas de desambiguación mejoran si se da un sesgo y se resuelve la ambigüedad de la misma manera en un mismo discurso.

Desambiguación basada en aprendizaje no supervisado

Los algoritmos de desambiguación basados en aprendizaje no supervisado aprenden los sentidos de las palabras a partir de un conjunto de datos de entrenamiento sin necesidad de disponer de definiciones para los sentidos que sean entendibles por parte de las personas. Estos algoritmos de desambiguación utilizan algún método estándar de agrupamiento, también llamado *clustering*, y una medida de distancia para determinar la similitud entre clústeres. De hecho, el método más usado en tareas lingüísticas es el **agrupamiento aglomerativo** que va fusionando sucesivamente los clústeres más similares.

El algoritmo de Schütze (1992) (1998) representa cada palabra como un vector de contexto para una bolsa de palabras y, entonces, entrena el algoritmo siguiendo tres pasos:

1. Para cada aparición w_i de la palabra w en un corpus, se calcula un vector contexto c .
2. Se utiliza un algoritmo de agrupamiento para agrupar los vectores de contexto c en un número predefinido de clústeres. De hecho, cada uno de los clústeres define un sentido de la palabra w .

3. Se calcula el centroide de cada clúster. Cada centroide s_j es un vector que representa un sentido de la palabra w .

Dado que este es un algoritmo de clasificación no supervisado, no se conoce el nombre de cada uno de los sentidos de la palabra w ; por lo que simplemente se les llama «el sentido j de la palabra w ».

Para eliminar la ambigüedad de una instancia particular t de la palabra w utilizando el algoritmo de Schütze se aplican otros tres pasos:

1. Se calcula el vector contexto c para la instancia t .
2. Se recuperan todos los vectores de sentido s_j de la palabra w .
3. Se asigna t al sentido representado por el vector de sentido s_j que está más cerca de t .

8.5. Similitud entre palabras

Una de las relaciones entre palabras que más se usa en el procesamiento del lenguaje natural es la **sinonimia**. La sinonimia es una **relación binaria entre dos palabras**: las palabras son sinónimas o no lo son. Sin embargo, se puede utilizar una métrica más relajada que permita calcular la similitud entre palabras, llamada **distancia semántica**.

Dos palabras son más similares si comparten más características de su significado, es decir, si son casi sinónimos. Por el contrario, dos palabras son menos similares o tienen mayor distancia semántica si comparten menos elementos de su significado.

Aunque se hable de similitud entre palabras, realmente la similitud debe medirse entre los sentidos de las palabras. Además, se debe distinguir entre **similitud entre**

palabras y relación entre palabras. Dos palabras son similares si son casi sinónimos y una puede sustituir a la otra en un contexto dado, sin embargo, algunas palabras pueden estar relacionadas sin ser similares.

Por ejemplo, las palabras *coche* y *gasolina* están estrechamente relacionadas, pero no son similares; mientras que las palabras *coche* y *bicicleta*, además de estar relacionadas, son más similares. De hecho, los antónimos son palabras que están relacionadas, pero que no son similares en absoluto.

Es importante destacar que muchos de los algoritmos que calculan la similitud entre palabras, lo que en realidad hacen es utilizar una medida de relación entre palabras y no únicamente de similitud, aunque típicamente siguen llamándose medidas de similitud.

Para medir la similitud entre palabras o, mejor dicho, la relación entre sentidos de las palabras, existen dos tipos de algoritmos.

- ▶ Los primeros calculan la similitud entre palabras utilizando la estructura de un tesauro y se estudian en este tema.
- ▶ Los segundos calculan la similitud entre palabras aplicando métodos de distribución y encontrando directamente palabras que tienen distribuciones similares en un corpus.

Similitud entre palabras basada en tesauros

Los algoritmos que calculan la similitud entre palabras utilizando tesauros miden la distancia entre dos sentidos en un tesauro en línea como, por ejemplo, en WordNet. Estos algoritmos utilizan la estructura jerárquica del tesauro para definir la similitud entre palabras.

En principio, se podría medir la similitud utilizando cualquier **conocimiento** existente en el tesauro, como podría ser la meronimia o el glosario. Sin embargo, en la práctica,

los algoritmos de similitud entre palabras basados en tesauros utilizan en general solo la jerarquía de relaciones de hiperonimia/hiponimia.

Los algoritmos más simples que utilizan tesauros se basan en la hipótesis de que los sentidos de las palabras son más similares si existe un camino más corto entre ellos en la representación gráfica del tesauro (Quillian, 1969). Entonces, un sentido es lo más parecido a sí mismo, luego a sus padres o hermanos y luego, es menos similar a los sentidos de las palabras que están lejos en el gráfico.

Por lo que se define la longitud del camino entre dos sentidos, representados por los nodos c_1 y c_2 , del gráfico del tesauro como $pathlen(c_1, c_2)$ y se calcula añadiendo una unidad al número de aristas existentes en el camino más corto entre los nodos c_1 y c_2 .

Entonces, se define la medida de similitud basada en la longitud del camino como:

$$sim_{path(c_1, c_2)} = \frac{1}{pathlen(c_1, c_2)}$$

Sin embargo, en la mayoría de las aplicaciones no se dispone de datos de entrada con los sentidos etiquetados, por lo que es necesario que el algoritmo de similitud pueda proporcionar la similitud entre las palabras en lugar de entre sentidos. Se puede aproximar dicha similitud entre palabras (requeriría de un proceso de desambiguación de sentidos) utilizando el par de sentidos de las propias palabras, que son las que dan el máximo valor de similitud de los sentidos (Resnik, 1995). Por lo que formalmente se define la similitud entre palabras a partir de la similitud entre sentidos de la siguiente forma:

$$wordsim(w_1, w_2) = \max_{\substack{c_1 \in senses(w_1) \\ c_2 \in senses(w_2)}} sim(c_1, c_2)$$

El algoritmo que mide la similitud basada en la longitud del camino hace la suposición implícita de que cada eslabón, en la representación gráfica del tesauro, presenta una distancia uniforme. En la práctica, este supuesto no es apropiado porque las relaciones en un nivel más profundo de la jerarquía representan a menudo una distancia más cercana, mientras que otras relaciones más arriba en la jerarquía representan una distancia más amplia. La solución pasa por normalizar las distancias en función a la profundidad de la jerarquía (Wu y Palmer, 1994) o asociar una distancia diferente a cada una de las aristas del tesauro.

Otros algoritmos que utilizan tesauros para calcular la similitud entre palabras se basan en la estructura del tesauro, pero también incluyen información probabilística derivada de un corpus. Son los llamados **algoritmos de similitud entre palabras basados en la cantidad de información** y utilizan medidas de teoría de la información para extraer información del corpus.

Según Resnik (1995), se define $P(c)$ como la probabilidad de que una palabra seleccionada al azar en un corpus sea una instancia del concepto c , es decir, una variable aleatoria de las palabras asociadas con cada concepto. Cualquier palabra del tesauro es un descendiente del concepto de raíz, llamado *root*, lo que implica que $P(\text{root}) = 1$. Intuitivamente, cuanto más bajo se encuentre un concepto en la jerarquía, menor será su probabilidad.

Entonces, para calcular las probabilidades, se hace un recuento del corpus y cada palabra en el corpus cuenta como una ocurrencia de cada concepto que sea su ancestro.

Esto formalmente se define como:

$$P(c) = \frac{\sum_{w \in \text{words}(c)} \text{count}(w)}{N}$$

Donde:

- ▶ $\text{words}(c)$ es el conjunto de palabras descendientes del concepto c .
- ▶ N es el número total de palabras en el corpus que también están presentes en el tesauro.

A partir de la teoría de la información, se define la **cantidad de información (IC)** de un concepto c como:

$$IC(c) = -\log P(c)$$

Además, a partir de la teoría de grafos se utiliza la definición de **ancestro común más bajo** de dos conceptos (LCS, lowest common subsumer en inglés). El $LCS(c_1, c_2)$ es el nodo más bajo en la jerarquía que tiene como descendientes a los conceptos c_1 y c_2 , es decir que es el ancestro (padre, abuelo, etc.) más cercano de ambos conceptos.

Existen diferentes medidas de similitud entre palabras calculadas a partir de la cantidad de información de un nodo. De hecho, la similitud entre dos palabras está relacionada con la información mutua entre las palabras, cuanta más información comparten las dos palabras, más similares van a ser. Resnik (1995) propone estimar la información mutua entre dos palabras como la cantidad de información del ancestro común más bajo de los dos nodos que representan las palabras. Por lo tanto, la **medida de similitud de Resnik** se calcula con la siguiente fórmula:

$$\text{sim}_{resnik(c_1, c_2)} = -\log P(LCS(c_1, c_2))$$

Otras medidas de similitud que amplían las ideas de Resnik son la medida de similitud de Lin (Lin, 1998) y la distancia de Jiang-Conrath (Jiang y Conrath, 1997).

En el vídeo *Cálculo de la similitud entre palabras utilizando la estructura jerárquica de un tesauro* se explicará cómo se calcula la similitud entre palabras utilizando la estructura jerárquica de un tesauro.



Accede al vídeo

8.6. Referencias bibliográficas

Agirre, E., López de Lacalle, O. y Soroa, A. (2014). *Random walks for knowledge-based word sense disambiguation*. *Computational Linguistics*, 40(1), 57-84.

Fellbaum, C. (Ed.). (1998). *WordNet: An Electronic Lexical Database*. MIT Press.

Gale, W. A., Church, K. W. y Yarowsky, D. (1992). *One sense per discourse*. En *Proceedings DARPA Speech and Natural Language Workshop* (pp. 233-237). New Jersey, Estados Unidos.

Hearst, M. A. (1991). Noun homograph disambiguation. *Proceedings of the 7th Conference of the University of Waterloo Centre for the New OED and Text Research*, 1-19.

Jiang, J. J. y Conrath, D. W. (1997). Semantic similarity based on corpus statistics and lexical taxonomy. En *Proceedings of International Conference Research on Computational Linguistics (ROCLING X)*. Taiwan: ACL.

Jurafsky, D. y Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall.

Kilgarriff, A. y Rosenzweig, J. (2000). Framework and results for English SENSEVAL. *Computers and the Humanities*, 34, 15-48.

Krovetz, R. (1998). *More than one sense per discourse*. Princeton, Estados Unidos: NEC Labs America.

Lesk, M. E. (1986). Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. En *Proceedings of the 5th International Conference on Systems Documentation* (pp. 24-26). Association for Computing Machinery.

Lin, D. (1998). An information-theoretic definition of similarity. En *Proceeding ICML '98 Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 296-304). Morgan Kaufmann Publishers.

Mihalcea, R. (2007). Using wikipedia for automatic word sense disambiguation. *NAACL-HLT 07*, 196-203.

Navigli, R. y Lapata, M. (2010). An experimental study of graph connectivity for unsupervised word sense disambiguation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4), 678–692.

Ponzetto, S. P. y Navigli, R. (2010). Knowledge-rich word sense disambiguation rivaling supervised systems. En *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics* (pp. 1522-1531). Association for Computational Linguistics.

Quillian, M. R. (1969). The teachable language comprehender: A simulation program and theory of language. *Communications of the ACM*, 12(8), 459-476.

RAE. (S. f. d). Homófono. En *Diccionario de la lengua española* (actualización de la 23^a ed.). <http://dle.rae.es/?id=KbZUpzR>

RAE. (S. f. b). Homógrafo. En *Diccionario de la lengua española* (actualización de la 23^a ed.). <http://dle.rae.es/?id=Kbl2I5o>

RAE. (S. f. c). Homónimo. En *Diccionario de la lengua española* (actualización de la 23^a ed.). <http://dle.rae.es/?id=Kbrilov>

RAE. (S. f. e). Metonimia. En *Diccionario de la lengua española* (actualización de la 23^a ed.). <http://dle.rae.es/?id=P7kP7xl>

RAE. (S. f. a). Semántica. En *Diccionario de la lengua española* (actualización de la 23^a ed.). <http://dle.rae.es/?id=XVRDns5>

Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. *International Joint Conference for Artificial Intelligence (IJCAI-95)* (pp. 448-453).

Schütze, H. (1992). Dimensions of meaning. En *Proceedings of Supercomputing '92* (pp. 787-796). IEEE Press.

Schütze, H. (1998). Automatic word sense discrimination. *Computational Linguistics*, 24(1), 97-124.

Vasilescu, F., Langlais, P. y Lapalme, G. (2004). Evaluating variants of the lesk approach for disambiguating words. En *4th International Conference on Language Resources and Evaluation* (pp. 633-636). ELRA.

Weaver, W. (1955). Translation. En W. N. Locke y A. D. Boothe (Eds.), *Machine Translation of Languages* (pp. 15-23). MIT Press.

Wu, Z. y Palmer, M. (1994). Verb semantics and lexical selection. En *Proceedings of the 32th Annual Meetings of the Associations for Computational Linguistics* (ACL-94) (pp. 133-138). ACL.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. ACL-95, 189-196.

Yuret, D. (2004). Some experiments with a Naive Bayes WSD system. *Senseval-3: 3rd International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*. <http://www.aclweb.org/anthology/W04-0864>

Zhong, Z. and Ng, H. T. (2010). It makes sense: A wide-coverage word sense disambiguation system for free text. En *Proceedings of the ACL 2010 System Demonstrations* (pp. 78-83). Association for Computational Linguistics.

Una orientación semántica al análisis de sentimientos

Turney, P. D. (2002). Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 417-424.
<http://www.aclweb.org/anthology/P02-1053.pdf>

El artículo presenta un método basado en el aprendizaje no supervisado para clasificar revisiones y utiliza una orientación semántica para el análisis de sentimientos. A la hora de calcular la orientación semántica de una oración, es decir, determinar si el sentimiento recogido en la frase es positivo o negativo, se aplica un algoritmo de similitud entre palabras basado en la cantidad de información y la información mutua. De hecho, lo que se hace es calcular la similitud de cada adjetivo y adverbio de la frase con una palabra de referencia positiva («*excellent*», excelente en inglés) y una de referencia negativa («*poor*», malo en inglés) y, a partir de esos valores, se asigna una valoración global a la frase que indica la orientación semántica de la oración.

Word Sense Disambiguation

Manning, C. y Schütze, H. (1999). Word sense disambiguation. En *Foundations of Statistical Natural Language Processing* (pp. 229-264). MIT Press.
<https://github.com/shivamms/books/blob/master/nlp/Foundations%20of%20Statistica l%20Natural%20Language%20Processing%20-%20Christopher%20D.%20Manning.pdf>

El capítulo siete del libro describe varios algoritmos probabilísticos para la desambiguación de los sentidos de las palabras. Los algoritmos presentados se basan tanto en el aprendizaje supervisado como en el no supervisado.

Test

1. Indica las afirmaciones correctas sobre el significado de las palabras:

 - A. El sentido es la representación de uno de los aspectos del significado de una palabra.
 - B. El significado de una palabra se representa a través de su lema.
 - C. La semántica léxica estudia el significado de las palabras y las relaciones de sentido entre ellas.
 - D. Cada uno de los significados de una palabra se llama sentido.
2. Indica las afirmaciones correctas sobre las palabras y sus sentidos:

 - A. Dos palabras son homófonas si suenan igual, tienen distinto significado y se escriben igual.
 - B. Dos sentidos de una palabra son homógrafos porque se escriben de la misma forma.
 - C. Dos palabras son homónimas si se pronuncian igual, pero se escriben de forma diferente.
 - D. Dos sentidos de una palabra son homónimos si están relacionados entre sí.
3. Indica las afirmaciones correctas sobre las relaciones semánticas entre sentidos:

 - A. A veces puede existir una conexión semántica entre los sentidos de las palabras.
 - B. La metonimia es una forma de emplear una palabra en un sentido distinto al que propiamente le corresponde, pero con el que tiene alguna conexión.
 - C. La polisemia es una relación semántica entre los sentidos de una misma palabra.
 - D. La polisemia es un tipo particular de metonimia.

- 4.** Indica las afirmaciones correctas sobre las relaciones entre sentidos de las palabras:
- A. La sinonimia es la relación contraria a la antonimia.
 - B. Existe una relación de antonimia entre los sentidos de dos palabras si sus significados son idénticos o casi idénticos.
 - C. Un tesauro es un diccionario que contiene una lista de palabras con sus sinónimos y sus antónimos.
 - D. Las bases de datos de relaciones léxicas contienen un conjunto de lemas, el conjunto de sentidos de cada lema, su definición y una lista de sinónimos. Sin embargo, no es obligatorio que contengan la pronunciación de los lemas ni detalles de las relaciones entre lemas.
- 5.** Indica las afirmaciones correctas sobre las relaciones entre sentidos de las palabras:
- A. Una palabra es un hiperónimo de otra si su significado incluye el significado de la otra palabra.
 - B. Una palabra es un hipónimo de otra si su significado está incluido en el significado de la otra palabra.
 - C. Las relaciones de hiponimia e hiperonimia se modelan a través de la jerarquía IS-A en las ontologías.
 - D. Una palabra es un merónimo de otra si su significado mantiene con el significado de la otra palabra una relación de la parte respecto al todo.

- 6.** Indica las afirmaciones correctas sobre los algoritmos de desambiguación del sentido de las palabras basados en aprendizaje supervisado:
- A. El clasificador se puede entrenar para desambiguar algunas palabras concretas de una muestra léxica o un texto entero.
 - B. Utilizan un tesoro que contiene las relaciones entre sentidos de las palabras para poder entrenar el clasificador.
 - C. Requieren tener un corpus de palabras etiquetadas con sus sentidos correctos para poder entrenar el clasificador.
 - D. El vector de características lingüísticas de las palabras de contexto con las que se entrena el clasificador puede constar de características de colocación y características sobre las palabras vecinas.
- 7.** Indica las afirmaciones correctas sobre los algoritmos de desambiguación del sentido de las palabras basados en conocimiento:
- A. Aplican el algoritmo de Lesk para seleccionar el sentido cuya definición en el diccionario comparte la mayor cantidad de palabras con el contexto.
 - B. Utilizan un tesoro que contiene las relaciones entre sentidos de las palabras para poder entrenar el clasificador.
 - C. Requieren tener un corpus de palabras etiquetadas con sus sentidos correctos para poder entrenar el clasificador.
 - D. Realizan un entrenamiento indirecto aplicando algoritmos de aprendizaje supervisado débil.

- 8.** Indica las afirmaciones correctas sobre los algoritmos de desambiguación del sentido de las palabras basados en aprendizaje semisupervisado:
- A. Aplican el algoritmo de Yarowsky para en cada iteración entrenar un clasificador a partir del corpus etiquetado, utilizar el clasificador para clasificar las instancias no etiquetadas y añadir a los datos de entrenamiento los ejemplos para los que se tenga mayor confianza en la clasificación.
 - B. Utilizan un tesauro que contiene las relaciones entre sentidos de las palabras para poder entrenar el clasificador.
 - C. Requieren tener un corpus de palabras etiquetadas con sus sentidos correctos para poder entrenar el clasificador.
 - D. Para seleccionar las instancias a etiquetar en el conjunto inicial se debe utilizar una heurística, por ejemplo, la heurística de un sentido por colocación o la heurística de un sentido por discurso.
- 9.** Indica las afirmaciones correctas sobre los algoritmos de desambiguación del sentido de las palabras basados en aprendizaje no supervisado:
- A. Utilizan un método estándar de agrupamiento, normalmente el algoritmo de agrupamiento aglomerativo, y una medida de distancia para determinar la similitud entre clústeres.
 - B. Utilizan un tesauro que contiene las relaciones entre sentidos de las palabras para poder entrenar el clasificador.
 - C. Requieren tener un corpus de palabras etiquetadas con sus sentidos correctos para poder entrenar el clasificador.
 - D. También se llama inducción del sentido de las palabras porque el conjunto de sentidos de cada palabra se aprende automáticamente.

10. Indica las afirmaciones correctas sobre la similitud entre palabras:

- A. Los algoritmos que calculan la similitud entre palabras miden una relación binaria, es decir si las palabras son casi-sinónimos o no lo son.
- B. Los algoritmos que calculan la similitud entre palabras utilizando la estructura de un tesauro se basan en la hipótesis de que los sentidos de las palabras son más similares si existe un camino más corto entre ellos.
- C. Muchos de los algoritmos que calculan la similitud entre palabras realmente lo que hacen es utilizar una medida de relación entre palabras y no de similitud.
- D. Existen algoritmos que calculan la similitud entre palabras utilizando la estructura de un tesauro e información probabilística derivada de un corpus como, por ejemplo, la cantidad de información.

Procesamiento del Lenguaje Natural

Aplicaciones del procesamiento del lenguaje natural

Índice

Esquema	3
Ideas clave	4
9.1. Introducción y objetivos	4
9.2. Traducción automática	5
9.3. Autocompletado y generación automática de resúmenes	17
9.4. Análisis de sentimientos	19
9.5. <i>Question Answering</i>	22
9.6. Reconocimiento automático del habla y text-to-speech	28
9.7. Referencias bibliográficas	36
A fondo	37
Test	38

Esquema

A P L I C A C I O N E S D E L P R O C E S A M I E N T O D E L L E N G U A J E N A T U R A L	
Traducción automática	<ul style="list-style-type: none">• Divergencias del lenguaje: la traducción entre dos lenguas no equivale a traducción de palabras individuales.• Modelo encoder-decoder: generar la representación de la secuencia entrada del <i>encoder</i> (contexto) y obtener la traducción de esa entrada desde el <i>decoder</i>.• Diseño de encoder-decoder: usando RNN o modelos de <i>transformers</i>.• Otros aspectos: evaluación de la traducción considerando fidelidad, fluencia y posibles sesgos.
Autocompletado y generación automática de resúmenes	<ul style="list-style-type: none">• Arquitectura: modelo <i>encoder-decoder</i> tanto para autocompletado como para generación de resúmenes.• Diferencias: en el caso del autocompletado se tiene un LM causal, en el caso de la generación de resúmenes no es necesario.• Entrenamiento: usando el concepto de <i>teacher forcing</i>.
Análisis de sentimientos	<ul style="list-style-type: none">• Análisis de sentimiento: detectar el sentimiento expresado por el autor de un texto hacia un determinado objeto, pudiendo ser por ejemplo, un sentimiento negativo o positivo.• Preprocesado: obtención de rasgos (ej.: BoW) desde textos.• Entrenamiento: algoritmos de ML de clasificación supervisados.• Otras posibilidades: uso de lexícones con sentimientos a nivel de palabra para inferir el sentimiento del texto.
Question Answering	<ul style="list-style-type: none">• IR: técnicas de recuperación de información. <i>Ad hoc retrieval</i>, donde se reciben consultas de usuarios y se busca responder a ellas utilizando un corpus.• Técnicas: representación de consultas y documentos con BoW con <i>embeddings</i>.• IR para <i>factoid QA</i>: dada una consulta y un texto, encontrar el pasaje del texto que responde a la pregunta.
Reconocimiento automático del habla y text-to-speech	<ul style="list-style-type: none">• Dificultades de un ASR: tamaño del vocabulario, hacia quién se está comunicando el usuario, el canal de comunicación y el ruido, y los acentos u otras características.• Arquitectura ASR: modelo <i>encoder-decoder</i> extrayendo rasgos de la señal acústica (entrada) y generando como salida el texto asociado.• Arquitectura TTS: texto de entrada, normalizado (vocalización), generación del espectrograma y módulo de vocoding.

9.1. Introducción y objetivos

A continuación, se introducirá algunas de las aplicaciones industriales más habituales basadas en PLN. En primer lugar, se presentarán los **sistemas de traducción automática**, donde se hablará de algunas de las técnicas basadas en redes neuronales más actuales, cómo poder evaluar los resultados obtenidos, así como otros aspectos éticos para tener en cuenta en el desarrollo del modelo, como es el tema de los sesgos. Posteriormente, se hablará de los **sistemas de autocompletado de textos y de generación automática de resúmenes**.

También, incluye una **presentación al problema del análisis de sentimientos**, así como una explicación sobre elementos para tener en cuenta en su desarrollo. Tras ello, se verá una de las **aplicaciones más relevantes de PLN** en la industria: los sistemas de *Question Answering*, con los que se busca responder a consultas de los usuarios en base a la información disponible en una colección de documentos.

Finalmente, se verán las aplicaciones de PLN tanto para generar texto desde audio, como para generar audio desde texto.

Objetivos

- ▶ Entender cómo funciona un sistema de traducción automática, así como contar con las nociones clave de qué algoritmos basados en redes neuronales se pueden usar para diseñarlo.
- ▶ Saber cómo se construyen los sistemas de autocompletado y de generación automática de resúmenes, y qué algoritmos basados en redes neuronales se pueden usar para desarrollarlos.

- ▶ Conocer los conceptos básicos del problema de análisis de sentimientos, así como entender algunos de los elementos clave necesarios para construir un sistema que pueda clasificar sentimientos en un texto.
- ▶ Entender cómo funciona un sistema de *Question Answering* en el contexto del campo de la recuperación de información, y qué algoritmos de PLN se pueden usar para desarrollarlos.
- ▶ Conocer cómo funcionan los sistemas de reconocimiento automático del habla y los de *text-to-speech*.

9.2. Traducción automática

En este capítulo se va a presentar la tarea de PLN de traducción automática (*Machine Translation, MT*), junto con algunos aspectos para tener en cuenta y las arquitecturas habituales que se usan para implementarla. El problema que se busca abordar con MT es, partiendo de un texto en una lengua, poder generar automáticamente su traducción a otra lengua. Ahora bien, como se comentará a continuación, una traducción precisa no pasa por convertir las palabras individuales de un texto en un lenguaje a su equivalente en otro, sino que se debe tener en cuenta toda la información de la secuencia del texto para hacer su traducción.

Divergencias del lenguaje

De cara a las tareas de MT es importante tener en cuenta que, a pesar de que existen elementos universales del lenguaje comunes a todas las lenguas, existen también diferencias entre ellas. Estas diferencias pueden ser a nivel de palabras concretas, lo que hace referencia a aspectos idiosincráticos o léxicos (como, por ejemplo, palabras que significan cosas distintas en según qué lengua), o pueden ser a nivel sistemático afectando a toda la estructura de una frase (por ejemplo, el orden en el que se sitúan los verbos y los objetos directos). Estas diferencias hacen que la traducción de un texto de una lengua a otra sea una tarea compleja, donde no es suficiente con

traducir las palabras individualmente, sino que hay que tener en cuenta todo el contexto y estructura de la frase.

Para completar el estudio de esta sección y ver en detalle los distintos aspectos que diferencias entre sí las lenguas, puedes leer el Capítulo 10 del libro: Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall. <https://web.stanford.edu/~jurafsky/slp3/10.pdf>

Modelo encoder-decoder

En el ámbito de PLN se han propuesto arquitecturas específicas de aprendizaje profundo de cara a poder realizar tareas de MT teniendo en cuenta las singularidades de cada lengua y las diferencias que existen entre ellas cuando se quieren realizar traducciones. Este es el caso de la **arquitectura encoder-decoder**, también conocida como **sequence-to-sequence (seq2seq)**, que **también** es **utilizada en** otras tareas de PLN como la **generación de resúmenes o la generación de respuestas**.

El esquema de una arquitectura *encoder-decoder* aparece en la siguiente imagen. Se puede apreciar que **estos modelos reciben la secuencia de datos de entrada dentro de una primera capa (*encoder*) con la que se genera una **representación (contexto)**. Esta representación sirve de entrada para la segunda **capa (decoder)** con la que se genera una secuencia de salida.**

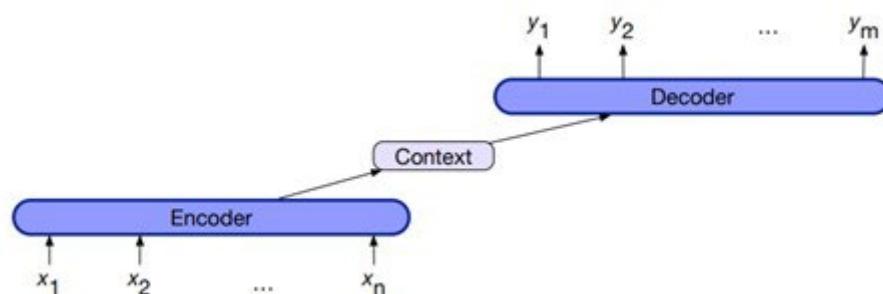


Figura 1. Esquema *encoder-decoder*. Fuente: Jurafsky, 2021.

Modelo *encoder-decoder* con RNN

Este esquema se puede construir con distintos modelos de aprendizaje profundo, como es el caso de las Redes Neuronales Recurrentes (*Recurrent Neural Networks*, RNN). Esta aproximación se muestra en la siguiente imagen. Como se puede apreciar, la secuencia de texto original sirve como entrada de la capa *encoder* convirtiendo cada palabra en un vector (ej., su *word embedding*). La última salida de la capa *encoding* corresponderá al vector de la última etapa, h_n . Este vector, que representa el *contexto* de la secuencia de entrada, se usará como entrada de la capa *decoder*, donde se construirá un modelo del lenguaje (*language model*, LM) con el que se prediga cada palabra dada la información de la secuencia hasta ese momento de la frase que se quiere traducir, junto con la información del «contexto». Un ejemplo simplificado de esto aparece en la siguiente imagen:

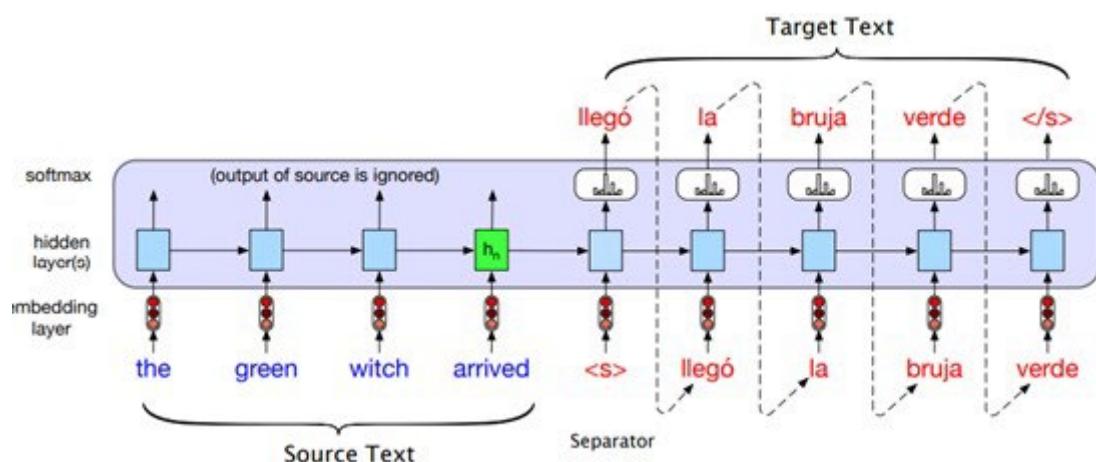


Figura 2. Arquitectura de MT basada en un esquema *encoder-decoder* con RNN. Fuente: Jurafsky, 2021.

Más en detalle, si la capa *decoder* es un LM, formalmente se expresaría de la siguiente manera:

$$p(y) = p(y_1)p(y_2|y_1)p(y_3|y_1, y_2)\dots P(y_m|y_1, \dots, y_{m-1})$$

Donde la probabilidad de tener una palabra en concreto depende de la secuencia previa. Ahora bien, como se ha comentado, se dispone también de la información del «contexto» obtenida desde la capa del *encoder*, de manera que quedaría lo siguiente:

$$p(y|x) = p(y_1|x)p(y_2|y_1,x)p(y_3|y_1,y_2,x)\dots P(y_m|y_1,\dots,y_{m-1},x)$$

Donde el vector x hace referencia a la secuencia del texto de origen. Con ello, el modelo LM predeciría la siguiente palabra dado el vector del texto original y la secuencia del texto traducido hasta ese momento. La imagen previa muestra como el token que conecta ambos modelos es un token separador $< s >$ a partir del cual se empieza a predecir la secuencia traducida. Este modelo simplificado estaría considerando el vector del contexto como entrada directa solo en la primera etapa de la capa *encoder*. Ahora bien, esto podría hacer que la información del contexto fuese perdiendo relevancia a medida que se avanza por las etapas del modelo *decoder*. Para evitar esto, una versión más sofisticada de la arquitectura consideraría el vector del contexto como entrada de todas las etapas de la capa *decoder*, tal como muestra la siguiente imagen.

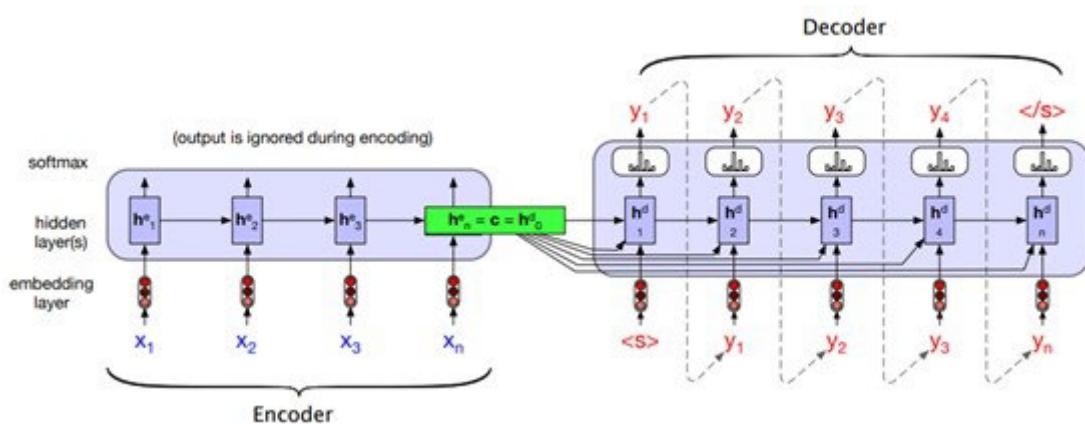


Figura 5. Arquitectura de MT donde el contexto sirve de entrada en todas las etapas del *decoder*. Fuente: Jurafsky, 2021.

Formalizado con ecuaciones, se tiene lo siguiente:

$$\begin{aligned}\mathbf{c} &= \mathbf{h}_n^e \\ \mathbf{h}_0^d &= \mathbf{c} \\ \mathbf{h}_t^d &= g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c}) \\ \mathbf{z}_t &= f(\mathbf{h}_t^d) \\ y_t &= \text{softmax}(\mathbf{z}_t)\end{aligned}$$

$$\hat{y}_t = \text{argmax}_{w \in V} P(w|x, y_1 \dots y_{t-1})$$

Donde los superíndices *e* y *d* hacen referencia al modelo de *encoder* o *decoder* respectivamente, *g* y *f* son funciones de activación, y cada salida del modelo *decoder* corresponde a la salida de aplicar la función *softmax*, donde la palabra que se predeciría correspondería a la que tiene mayor probabilidad (\hat{y}_t).

Como se puede observar, entrenar un modelo de MT con esta arquitectura requiere de un corpus donde existan frases originales y sus correspondientes frases traducidas. El ajuste de los pesos se haría calculando la función de coste (ej., entropía cruzada) resultante de comparar las palabras que se deberían haber predicho en cada fase del *decoder* con respecto a las palabras que ha predicho el modelo.

Ahora bien, el modelo, tal y como se ha descrito hasta ahora, estaría considerando la **palabra previa predicha** como entrada junto con el vector de salida de la capa oculta de la etapa anterior y el vector del contexto. El problema que tiene esta aproximación es que, si la predicción no es correcta, se estaría usando información que no es correcta para hacer la predicción, y el modelo tendería a irse desviando cada vez más de la frase correcta. Por este motivo se ha propuesto el uso del concepto de **teacher forcing**, con el que en la entrada de cada etapa durante el entrenamiento del modelo no corresponde a la salida predicha en la etapa previa, sino que se usa la palabra objetivo que se tendría que haber predicho. El esquema del entrenamiento con el modelo simplificado se ilustra en la siguiente imagen:

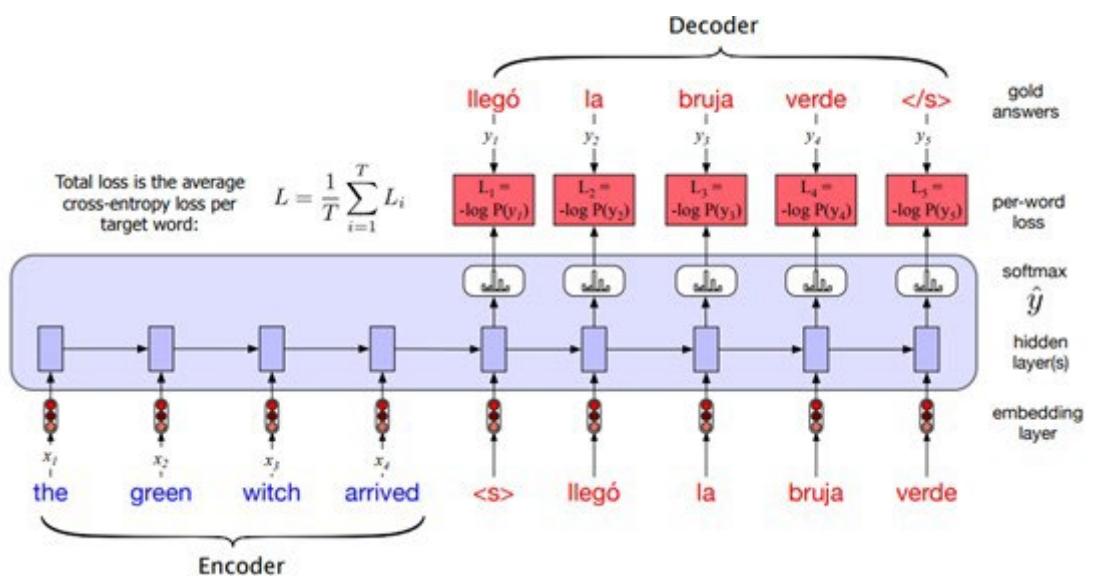


Figura 8. Esquema de entrenamiento del modelo para MT usando *teacher forcing*. Fuente: Jurafsky, 2021.

Modelo *encoder-decoder* con *attention*

Una **limitación** que tiene el **esquema previo** de *encoder-decoder* es que el **decoder** solo recibe la información de la **última etapa** del *encoder*, sin que se reciba directamente la información que hay en el resto de las etapas. Para solventar esta limitación, se ha **propuesto** el uso del mecanismo de ***attention*** dentro de estos modelos. Con ello, en lugar de usar como contexto un vector estático resultante de la **última etapa** del *encoder*, se construye uno en base a la suma ponderada de los vectores de todas las etapas del *encoder*. Esto se ilustra en la siguiente imagen.

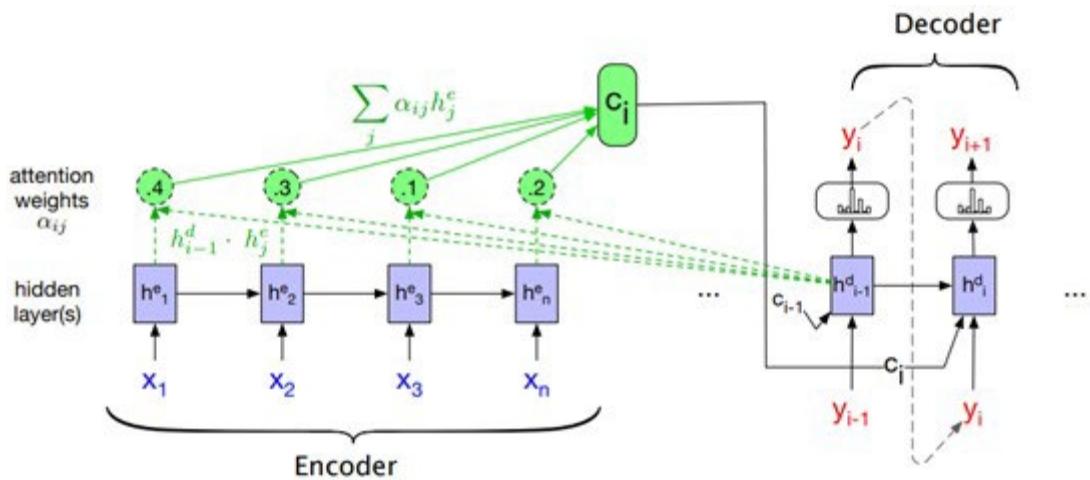


Figura 9. Esquema de *encoder-decoder* con mecanismo de *attention*. Fuente: Jurafsky, 2021.

Como se puede observar, la suma ponderada depende un parámetro α_{ij} , que para una etapa i del *decoder* se ajustará en función de una métrica de similitud entre el vector de la etapa j del *encoder* y el vector de la etapa $i-1$ del *decoder*. De esta manera, en el vector del *contexto* se dará más importancia a unas etapas u otras del *encoder* en función de la etapa concreta del *decoder*. El valor de este parámetro, normalizado con una función *softmax* y para una etapa i del *decoder* es:

$$\begin{aligned}\alpha_{ij} &= \text{softmax(score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) \forall j \in e) \\ &= \frac{\exp(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e))}{\sum_k \exp(\text{score}(\mathbf{h}_{i-1}^d, \mathbf{h}_k^e))}\end{aligned}$$

De manera que el vector del *contexto* correspondiente es:

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_j^e$$

La métrica de similitud (el *score*) se puede calcular de distintas maneras. Puede ser, por ejemplo, el producto escalar entre el vector de cada una de las etapas del *encoder* y la etapa correspondiente del *decoder*:

$$score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{i-1}^d \cdot \mathbf{h}_j^e$$

El valor del *score* con este producto escalar también se puede ajustar usando una matriz de pesos propia, W_s , de manera que quedaría:

$$score(\mathbf{h}_{i-1}^d, \mathbf{h}_j^e) = \mathbf{h}_{i-1}^d \mathbf{W}_s \mathbf{h}_j^e$$

Modelo *encoder-decoder* con *transformers*

La arquitectura de un modelo de MT también se puede construir con *transformers*, donde se mantiene el esquema *encoder-decoder*. El *encoder* se implementa mediante modelos de *multiheaded attention* bidireccionales, con los que se construye el vector del contexto. Por otro lado, el *decoder* se implementa con un esquema *multiheaded attention* como un LM causal. La siguiente imagen ilustra el esquema del modelo:

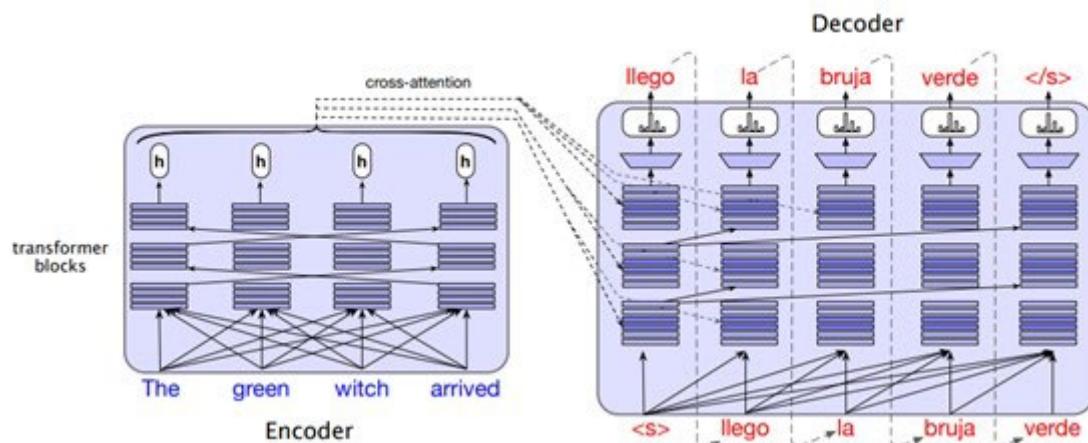


Figura 14. Modelo *encoder-decoder* con *Transformers*. Fuente: Jurafsky, 2021.

Cada bloque de *transformers* del modelo de *decoder* (tres bloques en la arquitectura anterior) va a estar compuesto de las siguientes capas:

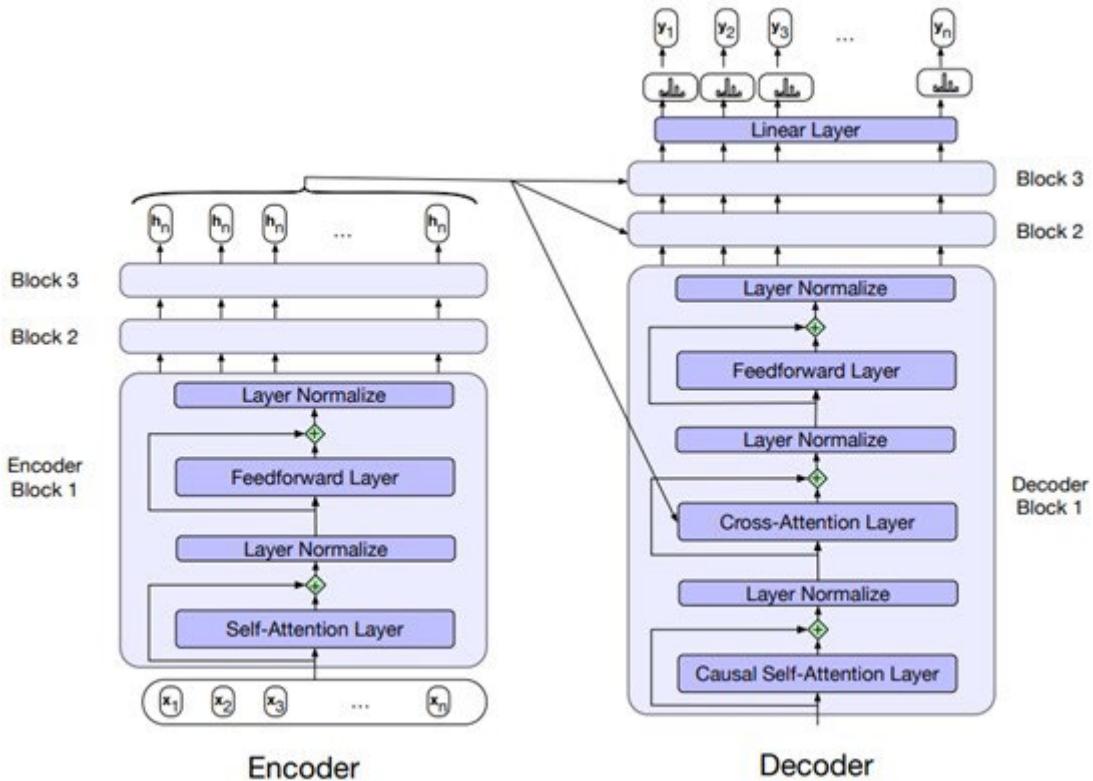


Figura 15. Esquema en detalle de un bloque de *transformers* del modelo *decoder*. Fuente: Jurafsky, 2021.

Como se puede observar, **hay dos tipos de modelos de attention**. Por un lado, aparece la **capa causal**, donde se sigue el esquema habitual de modelo *self-attention* en el que la entrada se basa en la secuencia recorrida hasta ese momento. Ahora bien, también aparece una **capa cross-attention**, donde el valor **Q (query)** corresponde a la información de salida de la capa previa, pero **K (key)** y **V (value)** corresponden al **vector de contexto** obtenido del *encoder*. De manera que, si W^Q , W^K y W^V son las matrices de pesos para las entradas Q, K y V respectivamente, la salida de esta capa (normalizada y tras aplicar la función *softmax*) sería:

$$\mathbf{Q} = \mathbf{W}^Q \mathbf{H}^{dec[i-1]}; \quad \mathbf{K} = \mathbf{W}^K \mathbf{H}^{enc}; \quad \mathbf{V} = \mathbf{W}^V \mathbf{H}^{enc}$$

$$CrossAttention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

Con ello, la capa de *cross attention*, permite que el modelo *decoder* reciba como información para la traducción la información asociada al modelo *encoder* sobre el texto original.

De cara al entrenamiento del modelo, se suele aplicar el concepto de **teacher forcing**, igual que se hizo con las RNN.

Evaluación de un modelo de MT

Desde el punto de vista de la evaluación de los modelos de MT se analizan dos aspectos. Por un lado, la **adecuación o fidelidad**, que mide cómo de bien se representa la frase traducida de referencia con la traducción obtenida automáticamente. Por otro, la **fluencia** que mide si la traducción es clara, legible, natural, gramáticamente correcta.

La **fidelidad** se puede medir con métricas como **character F-score** (chrF), que compara el solape en n-gramas que hay entre la frase traducida con el modelo de MT y la traducción original. El cálculo es similar a la métrica de F-score de los modelos de ML: se calcula el porcentaje de n-gramas de la traducción de referencia que aparecen en la frase traducida automáticamente (hipótesis), y viceversa. Esto se hace todos los n-gramas anteriores (es decir, se calcula para bigramas y para unigramas). Posteriormente, se hace la media de estos valores, y se calcula chrP (que tiene el valor calculado de la traducción automática frente a la referencia) y chpR (que tiene el valor calculado de la referencia frente a la traducción automática). Especificando un parámetro β , el cálculo de chrF sería:

$$\text{chrF}\beta = (1 + \beta^2) \frac{\text{chrP} \cdot \text{chrR}}{\beta^2 \cdot \text{chrP} + \text{chrR}}$$

La **métrica chrF** es útil especialmente para comparar dos modelos de traducción frente a un mismo conjunto de datos, más que como métrica para analizar un

modelo de manera aislada o para comparar dos modelos entrenados en conjuntos de datos de distintas lenguas.

Existen otro tipo de métricas con las que, en lugar de **comparar** las traducciones obtenidas con la referencia a nivel de cadenas de caracteres, se comparan a nivel de **embeddings**. Esto ofrece mayor flexibilidad en las comparaciones, ya que una traducción puede ser correcta a pesar de no seguir exactamente el orden de los elementos de la frase de referencia. Esta aproximación también está bien para comparar traducciones donde puede que se hayan obtenido sinónimos de palabras, pero que también serían válidos como traducción. Un ejemplo es la métrica **BERTScore**, con la que se obtienen los embeddings de las palabras de la traducción de referencia \tilde{x} y las de la traducción obtenida x_i y se comparan ambos vectores, de manera que se pueden calcular métricas de precisión y *recall* de la siguiente manera:

$$R_{BERT} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\tilde{x}_j \in \tilde{x}} x_i \cdot \tilde{x}_j \quad P_{BERT} = \frac{1}{|\tilde{x}|} \sum_{\tilde{x}_j \in \tilde{x}} \max_{x_i \in x} x_i \cdot \tilde{x}_j$$

Sesgos en los modelos

Un aspecto importante a tener en cuenta en los modelos de PLN en general, y en los de MT en particular, es el tema de los sesgos. Como cualquier modelo de aprendizaje automático, **un modelo de MT aprende a traducir en base a un conjunto de datos de origen que puede contener distintos tipos de sesgos que se propagarán al modelo y a sus predicciones**. Por ejemplo, si se tiene un modelo de MT con el que se quieren traducir estos textos:

Hungarian (gender neutral) source	English MT output
ő egy ápoló	she is a nurse
ő egy tudós	he is a scientist
ő egy mérnök	he is an engineer
ő egy pék	he is a baker
ő egy tanár	she is a teacher
ő egy vesküvőszervező	she is a wedding organizer
ő egy vezérigazgató	he is a CEO

Figura 19. Ejemplo de traducciones sesgadas entre textos con género neutro (en húngaro) y textos con el género especificado (inglés). Fuente: Jurafsky, 2021.

Se puede observar cómo partiendo de un texto de género neutro, se tiene una tendencia a asignar un género concreto a algunas profesiones y otro a otras, cuando esta especificación del género no aparece en las frases de origen que se quieren traducir. Este es un claro **ejemplo de sesgos de género** en los modelos: debido a que en los **textos de origen no hay un equilibrio entre las referencias a determinadas profesiones y las referencias a determinados géneros** (ej., hay muchos más textos que hablan de *engineers* junto con *he* que con *she*), el modelo aprende en base a este sesgo y lo usa en sus predicciones.

De cara a tratar con este problema de sesgos en los modelos de PLN y MT, existen distintas **soluciones**, desde la **detección a nivel de datos de origen, a nivel del entrenamiento del modelo, o a nivel de sus predicciones**, hasta distintas maneras de **mitigarlo**. Esto se enmarca dentro del ámbito de la **Inteligencia Artificial Responsable** (Responsible Artificial Intelligence, RAI), concretamente dentro del ámbito de las técnicas de **Fairness**.

Para ampliar el tema de los sesgos en el ámbito de PLN se puede completar el estudio con el Capítulo 10 del libro, pg. 25: Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall.

<https://web.stanford.edu/~jurafsky/slp3/10.pdf>

9.3. Autocompletado y generación automática de resúmenes

Los bloques de *transformers* se pueden usar también para construir modelos de **autocompletado de texto**. Siguiendo con la arquitectura de un *transformer* para la construcción de un LM causal, **se entrena un modelo que trate de predecir, dada una secuencia de N tokens, los M tokens siguientes**. Esto se refleja en la siguiente imagen:

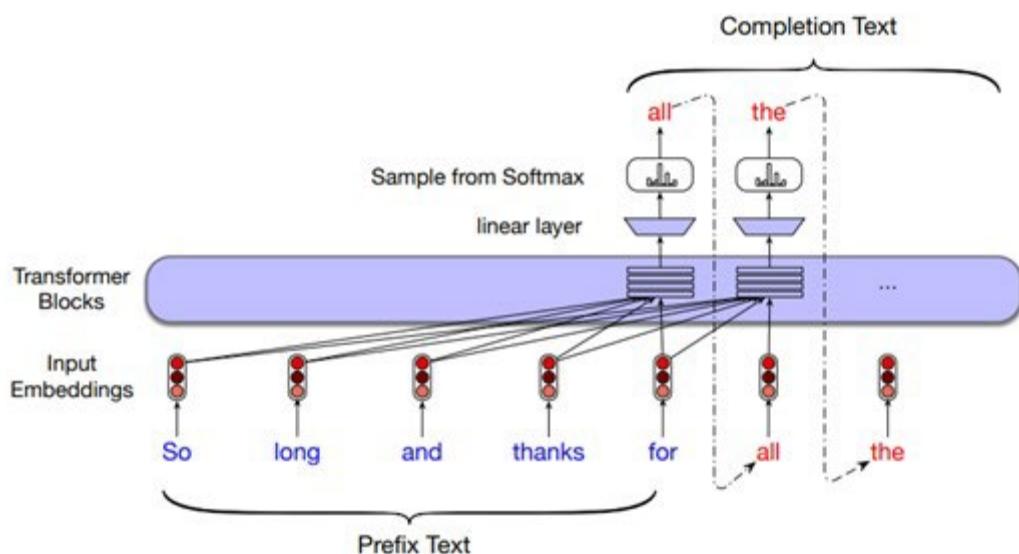


Figura 20. Arquitectura para un modelo de autocompletado de texto basado en *transformers*. Fuente: Jurafsky, 2021.

Como se puede observar, dada la secuencia:

- ▶ *So long and thanks for.*

El modelo va prediciendo uno a uno los siguientes tokens, generando así el texto autocompletado. Una vez que genera el primer token, este junto con la secuencia de entrada, se usan para predecir el siguiente token, y así sucesivamente.

Otro ejemplo de uso del bloque de *transformers* es para la construcción de modelos para tareas de **generación automática de resúmenes**, usando una arquitectura

similar a la descrita previamente. Suponiendo que se tienen pares de textos originales y sus correspondientes resúmenes, tal que (x_1, \dots, x_m) es la secuencia de texto original, y (y_1, \dots, y_n) el resumen correspondiente, se pueden concatenar ambas secuencias, usando un token que delimita una de la otra. La secuencia final sería, por tanto:

$$(x_1, \dots, x_m, \delta, y_1, \dots, y_n)$$

Con δ el token que delimita una de la otra, m el tamaño de la secuencia del texto original, n la del texto resumido, y $n + m + 1$ la secuencia usada en el modelo. Con ello, usando un LM causal con *transformers*, se tratarán de predecir ambas secuencias como una única. Una vez entrenado un modelo con los datos de entrenamiento, se podrá usar para, dado un texto de origen, obtener la secuencia que le seguiría tras el token final δ , que correspondería a la secuencia del texto resumen. Estos modelos se suelen entrenar con técnicas de **teacher forcing**. Una ilustración de la arquitectura aparece en la siguiente imagen:

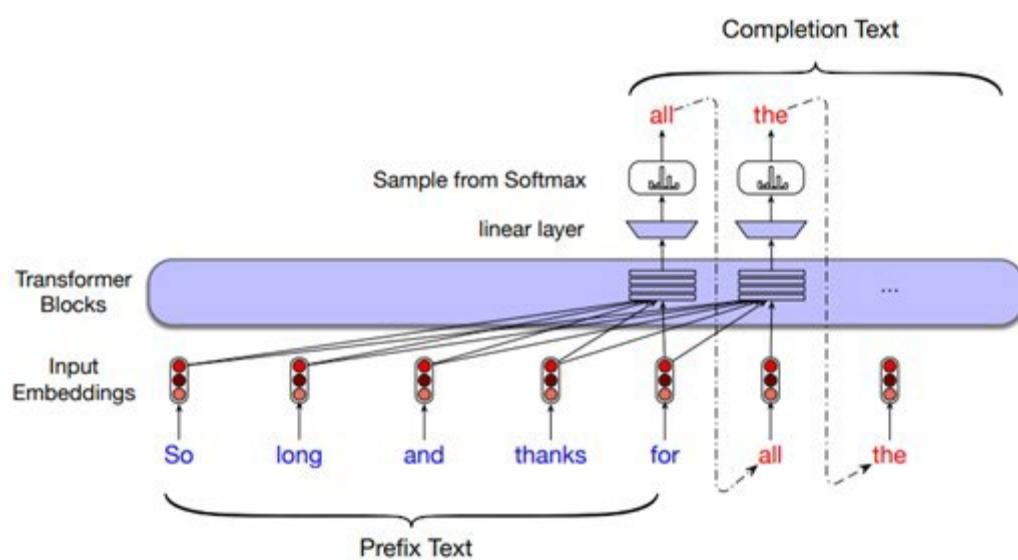


Figura 20. Arquitectura para un modelo de resumen de textos basado en *transformers*. Fuente: Jurafsky, 2021.

9.4. Análisis de sentimientos

El análisis de sentimientos es una tarea dentro del ámbito de PLN con el que se busca **detectar el sentimiento expresado por el autor de un texto hacia un determinado objeto**, pudiendo ser, por ejemplo, un sentimiento negativo o positivo.

Así, el objetivo de la clasificación de sentimientos a nivel computacional es usar distintos algoritmos que permitan hacer este análisis de manera automática.

Por ejemplo, dadas las siguientes frases, el objetivo sería detectar que en la primera el sentimiento es negativo, y en la segunda, positivo:

- ▶ Este producto es terrible. No cumple con las expectativas y lo he tenido que devolver.
- ▶ Magnífica compra y muy buena atención por parte del vendedor. El producto me ha encantado.

Una manera de hacer esa detección automática es **mediante el uso de algoritmos supervisados de aprendizaje automático**. Para ello, es necesario partir de un corpus en el que se tengan ejemplos de frases etiquetadas con su correspondiente **sentimiento** (por ejemplo, positivo o negativo), y se trate de predecir ese sentimiento en base a rasgos (*features*) obtenidas de cada uno de los textos. Existen distintas maneras de obtener esos rasgos, como por ejemplo con las técnicas de representación vectorial de bolsas de palabras (BoW). De esta manera, se transformarían los textos de entrada en una matriz donde cada fila representa uno de los textos y cada uno de los rasgos correspondería a una de las palabras del vocabulario, como se muestra en el siguiente ejemplo:

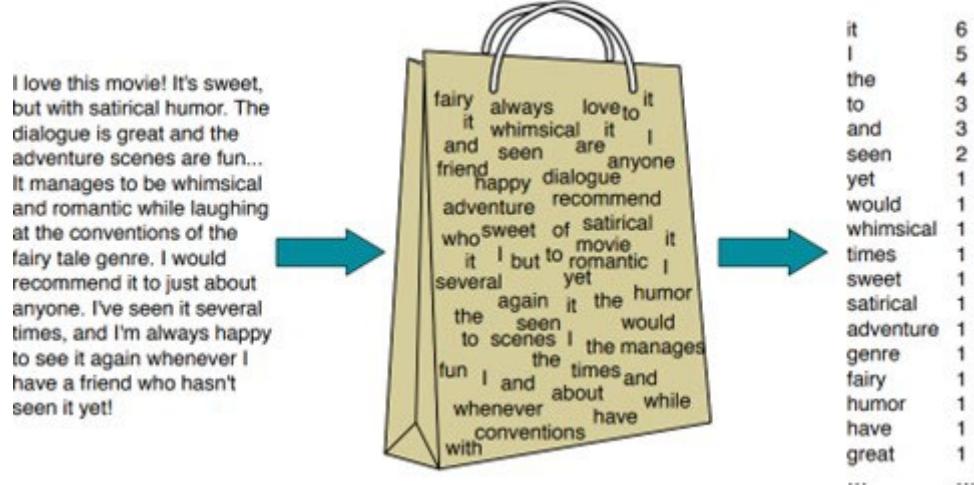


Figura 21. Ejemplo de BoW sobre un texto. Fuente: Jurafsky, 2021.

Teniendo esta matriz, y teniendo las clases (los sentimientos) que se quieren predecir, se pueden usar distintos algoritmos de aprendizaje supervisado para predecir el sentimiento asociado sobre cualquier texto de entrada.

El modelado de los textos de entrada y la construcción de los rasgos asociados no sólo se tiene por qué realizar con técnicas como BoW, sino que también se puede realizar con técnicas basadas en *word embeddings*.

Además de esta aproximación, que exige disponer de un corpus con textos etiquetados con su correspondiente sentimiento, existen otras posibilidades basadas en **lexicones**, donde la información de los sentimientos se tiene a nivel de palabras **individuales**. Partiendo de algún lexicón disponible, se trata de inferir el sentimiento asociado a todo el texto desde el sentimiento que tienen sus palabras individuales, apoyándose en el principio de composición.

Una opción simple de aplicar esto es mediante el uso de **reglas de clasificación** definidas a priori, que asocian como sentimiento principal de un texto aquél que aparezca más veces entre las palabras que lo componen. Así, se puede calcular una métrica de *sentimiento positivo* f^+ contando las palabras w del texto que tengan sentimiento positivo θ_w^+ , y lo mismo para el *sentimiento negativo* f^- con las palabras

de sentimiento negativo θ_w^- . Con ello, obteniendo los cocientes $\frac{f^+}{f^-}$ y $\frac{f^-}{f^+}$, comparándolos con un valor umbral λ , se decide si clasificar ese texto como positivo o negativo.

$$f^+ = \sum_{w \text{ s.t. } w \in \text{positivelexicon}} \theta_w^+ \text{count}(w)$$

$$f^- = \sum_{w \text{ s.t. } w \in \text{negativelexicon}} \theta_w^- \text{count}(w)$$

$$\text{sentiment} = \begin{cases} + & \text{if } \frac{f^+}{f^-} > \lambda \\ - & \text{if } \frac{f^-}{f^+} > \lambda \\ 0 & \text{otherwise.} \end{cases}$$

La representación de un texto en base al sentimiento de las palabras individuales que lo componen (obtenido desde un lexicón), **también sirve como rasgos adicionales** de un modelo de aprendizaje supervisado para el caso en el que se disponga de un corpus de entrada con sentimientos principales anotados en textos.

Así, además de los rasgos obtenidos en base a técnicas como BoW o *word embeddings*, se pueden incorporar rasgos adicionales basados en la información que proporcionan esos diccionarios.

Puedes completar el estudio sobre la clasificación de sentimientos y el uso de diccionarios con el Capítulo 20 del libro: Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall.

<https://web.stanford.edu/~jurafsky/slp3/20.pdf>

9.5. Question Answering

Un sistema de *Question Answering* (QA) tiene como **objetivo poder responder a consultas o preguntas que formule un usuario**. Estos sistemas forman parte de los buscadores de información, asistentes virtuales de manera que un usuario realice una consulta y el sistema pueda proporcionar una respuesta a la misma.

La mayoría de los sistemas de QA tratan de responder a un tipo de preguntas concretas: las preguntas sobre hechos (*factoid questions*), que hacen referencia a consultas sobre datos o hechos particulares, y que se pueden responder de manera directa.

Un ejemplo de preguntas de este tipo sería consultas como «¿Dónde se encuentra el Coliseo?» o «¿A qué edad se puede conducir legalmente en USA?». Este tipo de preguntas es el caso **contrario a las preguntas abiertas**, donde no se puede responder con una única respuesta (como un dato concreto o un hecho particular). Un ejemplo de preguntas de este segundo tipo sería «¿Cómo aprender a tocar el piano?»

Dentro de los **paradigmas** que se pueden utilizar para construir un sistema de QA existen diversas opciones, como se verá a continuación.

Information retrieval

Information Retrieval (IR) es un campo que incluye distintas técnicas de recuperación de información, y donde un caso concreto es el de **ad hoc retrieval**, donde se reciben consultas de usuarios (como textos de entrada) y se busca responder a ellas utilizando un **corpus de datos** (ej., una colección de documentos). Un ejemplo de esta arquitectura aparece en la siguiente imagen.

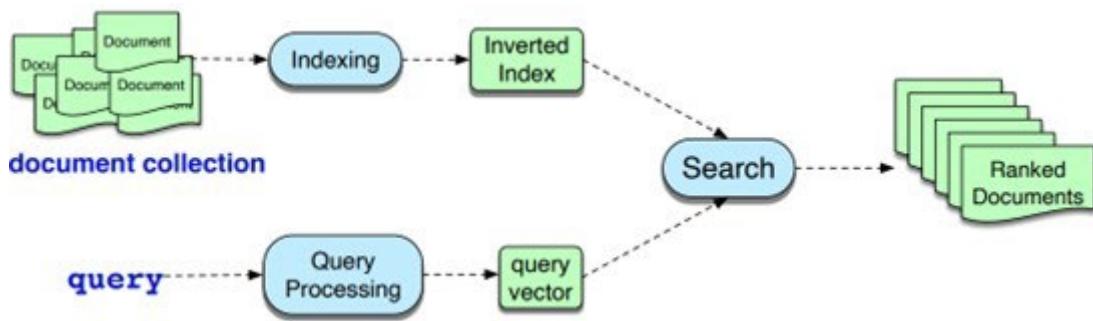


Figura 23. Esquema de un sistema de IR. Fuente: Jurafsky, 2021.

Como se puede observar, se recibe una **consulta (query)** de un usuario, y se expresa mediante un vector (con alguna de las técnicas de «representación vectorial» ya vistas). En el ejemplo básico de un IR, se usarían técnicas basadas en bolsas de palabras (BoW, *bag-of-words*) para la representación de la *query* de entrada y de los documentos de la colección, para usar después una métrica de similitud vectorial y encontrar el documento más cercano a la *query*. Usando una representación basada en TF-IDF, la métrica de similitud entre una *query* q y un documento d se puede aproximar como:

$$\text{score}(q, d) = \sum_{t \in q} \frac{\text{tf-idf}(t, d)}{|d|}$$

Donde, dado que la *query* tendrá menos términos que un documento, sólo se consideran los términos del documento que aparecen en la *query*, normalizando el resultado en función de los términos que tenga el documento. La arquitectura anterior contiene un elemento adicional, el **bloque inverted index**, con el que se hace que la búsqueda de documentos sea más eficiente. En lugar de consultar siempre toda la colección para una *query* de entrada, se usa un diccionario intermedio que tiene como claves los distintos términos, y asociado a ellas los documentos que tienen esos términos, junto con información adicional como la frecuencia de los términos en esos documentos. De esta manera, frente a esa *query* se seleccionan sólo los documentos relevantes y se calcula la similitud respecto a ellos.

IR con vectores densos

El esquema previo de QA usando técnicas basadas en BoW tiene como problema que solo se recuperarán documentos que tengan palabras que coincidan exactamente con la *query* del usuario. Ahora bien, no se considerarían otras circunstancias como, por ejemplo, cuando se usen sinónimos de palabras.

Una manera de hacer esto es mediante el uso de *embeddings* obtenidos desde modelos preentrenados, como BERT. BERT proporcionaría el vector de *embedding* para una palabra concreta. De cara a representar todo el texto como un único vector, se pueden usar técnicas de *pooling*, de manera que se combinen los valores de los vectores de cada palabra en un único vector que represente el texto. Así, representando con un vector la *query* y con otro el documento, se puede obtener el valor de la similitud entre ellos.

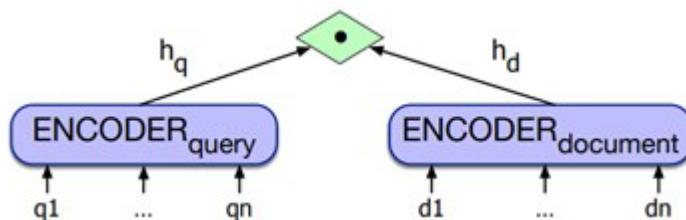


Figura 25. IR mediante el uso de embeddings. Fuente: Jurafsky, 2021.

IR para factoid QA

Los sistemas de IR se pueden usar para no solo encontrar documentos relevantes para una consulta, sino también para **poder responder a consultas** que consisten en preguntas específicas usando la información disponible en la colección de documentos. Estos sistemas de QA basados en IR (denominados IR-based QA) se pueden diseñar con una aproximación de recuperación y lectura, tal y como indica la siguiente imagen.

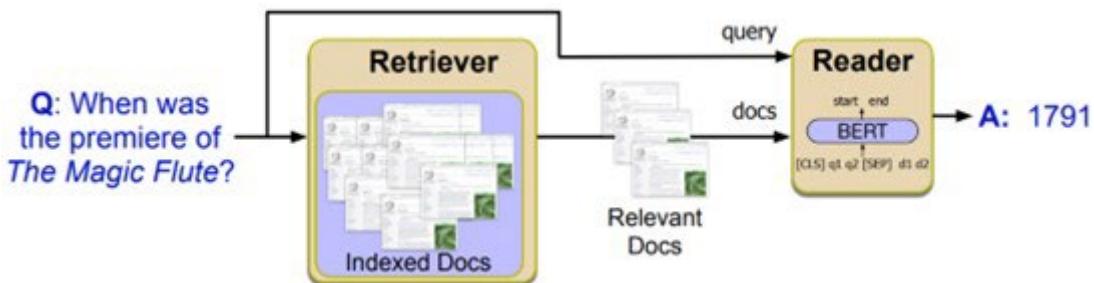


Figura 26. Sistema de IR para *factoid questions*. Fuente: Jurafsky, 2021.

Esta **aproximación** tiene **dos fases**. En la primera, **se recuperan documentos de la colección** que son relevantes en relación con la consulta de entrada. En la **segunda**, se lleva a cabo un **proceso de comprensión del texto mediante algoritmos** (basados, por ejemplo, en redes neuronales) con los que se trata de encontrar el fragmento de texto concreto que respondería a la consulta del usuario. Así, esta fase de comprensión mediante la detección de fragmentos (*spans*) de texto que respondan a la pregunta de entrada pasa por identificar una subcadena de caracteres que sirva como respuesta.

Formalmente, la tarea sería: dada una consulta q de n tokens q_1, q_2, \dots, q_n , y dado un documento p de m tokens p_1, p_2, \dots, p_m , calcular la probabilidad de que la cadena de texto a sea la respuesta, $P(a | q, p)$. Este cálculo de probabilidad se puede aproximar considerando sólo las posiciones iniciales y finales de la cadena a , tal que:

$$P(a|q,p) = P_{start}(a_s|q,p)P_{end}(a_e|q,p)$$

De manera que para cada token p_i del documento p , se obtengan dos probabilidades, $p_{start}(i)$ y $p_{end}(i)$ que representen la probabilidad de que ese token sea el principio o fin de la respuesta, respectivamente. La imagen siguiente muestra la arquitectura de un modelo para esta tarea.

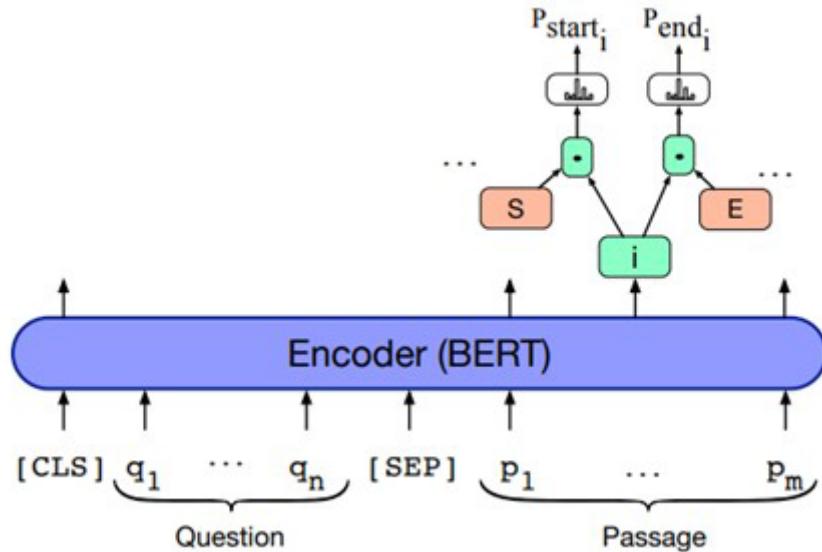


Figura 27. Arquitectura de un modelo para QA basado en fragmentos de documentos. Fuente: Jurafsky, 2021.

Partiendo de los tokens de la consulta de entrada y de un documento en cuestión, se concatenan usando un token genérico [SEP], sirviendo como entrada de un modelo de *encoding*, como por ejemplo uno basado en la arquitectura de BERT. Con ello, se obtiene primero un vector de salida p'_i para cada etapa i correspondiente a un token de entrada p_i del documento. Este vector de salida se usa en un producto escalar con un vector de *embedding* S para obtener la probabilidad de que ese *token* p_i del documento corresponda al inicio de la subsecuencia de respuesta. Análogamente, se multiplicaría con un vector E para obtener la probabilidad de que corresponda al fin de la secuencia. Los valores de los vectores de *embeddings* S y E se obtendrían durante el entrenamiento (*fine tuning*) del modelo sobre el conjunto de datos usado en el entrenamiento. De esta manera, normalizando con una función *softmax*, el cálculo de estas probabilidades sería:

$$P_{\text{start}_i} = \frac{\exp(S \cdot p'_i)}{\sum_j \exp(S \cdot p'_j)}$$

$$P_{\text{end}_i} = \frac{\exp(E \cdot p'_i)}{\sum_j \exp(E \cdot p'_j)}$$

El entrenamiento del modelo se haría usando conjuntos de datos con pares de preguntas-documentos donde está indicado el inicio y el fin de la respuesta. Con ello, la subsecuencia candidata como respuesta a la consulta sería la que va del token k al token j , con $j \geq k$, P'_{start_k} la probabilidad máxima de inicio (que corresponde al token k), y P'_{end_j} la probabilidad máxima de fin (que corresponde al token j). Comparando los tokens de inicio-fin de la secuencia propuesta como respuesta con los tokens de inicio y fin reales del conjunto de datos de entrenamiento, se obtendría **la métrica de error**.

Además de esto, el modelo tiene que ser capaz de predecir también los casos en los que la respuesta a la pregunta no se pueda responder con ningún documento. Una manera de hacer esto es tratando esos casos como aquellos en los que la respuesta correspondiese al token [CLS] (que sería tanto el inicio como el fin de la secuencia).

Finalmente, una arquitectura como la de BERT tiene un **número limitado de tokens de entrada** (512 como máximo para la arquitectura estándar). El problema es que los tokens de una pregunta-documento pueden superar este valor. Para tratar estas casuísticas, se recorre el documento con ventanas de tokens que correspondan al máximo permitido (descontando el número de tokens de la consulta) y se predice si la respuesta pudiera estar en esa ventana o no. Posteriormente, si hay varias respuestas posibles correspondientes a distintas ventanas con una probabilidad superior al caso por defecto de que la respuesta sea [CLS] (es decir, que no hay respuesta), se elige de entre ellas la de mayor probabilidad.

Evaluación de las respuestas

De cara a evaluar si el sistema funciona correctamente y es capaz de responder a una pregunta con la respuesta adecuada, existen **métricas como mean reciprocal rank (MRR)**. Con MRR, se compara la lista de respuestas posibles ordenadas por probabilidad con respecto a la pregunta de referencia de los datos de evaluación. Si, por ejemplo, la respuesta correcta aparece en la tercera posición, el valor RR

(reciprocal rank) sería 1/3. Con ello, para un conjunto de evaluación de Q preguntas, la métrica final sería la media de los distintos RR, como expresa la siguiente ecuación:

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

Puedes consultar otros sistemas de QA basados tanto en aprendizaje neuronal como en técnicas clásicas en el Capítulo 23 del libro: Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall.

<https://web.stanford.edu/~jurafsky/slp3/23.pdf>

9.6. Reconocimiento automático del habla y text-to-speech

Dentro de las aplicaciones industriales de PLN aparece el **Reconocimiento Automático del Habla** (Automatic Speech Recognition, ASR), con el que se busca transformar a texto el audio generado por el habla de un usuario. Esto es importante para muchas aplicaciones basadas en PLN, como por ejemplo los asistentes virtuales, donde un usuario puede comunicarse libremente con el sistema para realizar preguntas o pedirle a este que realice determinadas tareas.

Así, con el ASR se busca poder tener una interfaz de comunicación con un usuario, de manera que este se pueda expresar con el habla, y esa información se convierta en texto para poder ser procesada computacionalmente.

Ahora bien, esta tarea es compleja, y aparecen varios aspectos a considerar.

Por un lado, es importante el **aspecto relacionado con el tamaño del vocabulario usado por los usuarios**. La complejidad de la tarea de ASR no es la misma si, por ejemplo, se espera que el usuario conteste con un «Sí» o «No», frente a un caso en el que el usuario puede comunicarse con el sistema de manera abierta.

En segundo lugar, la **complejidad dependerá de hacia quién se esté comunicando el usuario o la manera en la que este lo está haciendo**. Por ejemplo, no es lo mismo la claridad de la comunicación cuando el usuario se dirige a un sistema (donde las órdenes o cuestiones que formula son relativamente claras o directas), frente a la claridad que puede tener la comunicación de un usuario cuando este se encuentra conversando con otra persona. La tarea de ASR será menos compleja en ese primer caso que en el segundo.

En tercer lugar, **influye el canal de comunicación y el ruido**. La complejidad del tratamiento de la información se verá afectada en función de aspectos como el lugar donde se esté comunicando el usuario con el sistema, siendo, por ejemplo, más compleja cuando se está en un lugar ruidoso (como una calle) frente a otros lugares más tranquilos (como una sala cerrada sin más usuarios).

Finalmente, **influyen** también aspectos **como el acento u otras características concretas de cada usuario particular**. No serán igual de precisos los resultados del ASR cuando se reciban mensajes en inglés con un determinado tipo de acento si el sistema sólo se ha entrenado sobre datos de un acento en concreto.

Además de las tareas de ASR, existe también el **proceso inverso**: partiendo de un **texto escrito, poder generar automáticamente un mensaje de audio asociado**. Esta tarea se conoce como **text-to-speech (TTS)**.

Así, la tarea de TTS es el proceso inverso al ASR, como se indica en la siguiente imagen. En ella se indica como la representación del mensaje es o bien en modo texto, o bien en modo audio, caracterizada por una onda acústica.

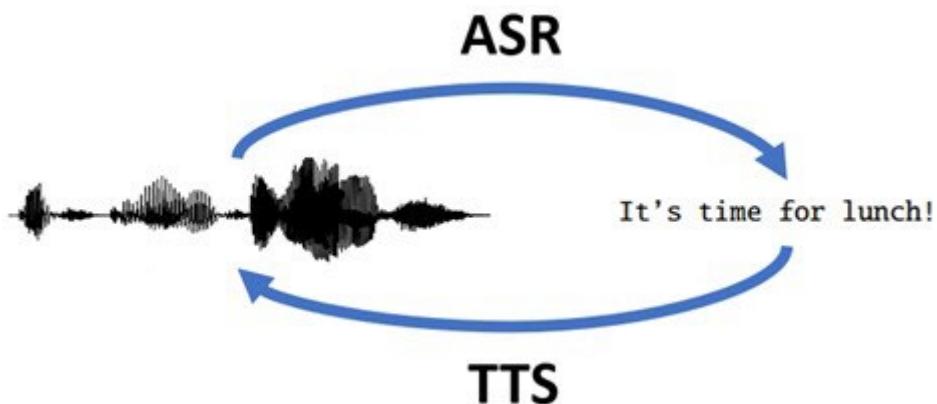


Figura 31. Tareas de reconocimiento del habla: ASR para pasar de voz a texto, y TTS para pasar de texto a voz.

Fuente:

Extracción de rasgos para ASR

Para poder pasar de una onda acústica al texto correspondiente mediante el uso de técnicas de ASR, el **primer paso es identificar una serie de variables** o rasgos que representen la información fundamental de dicha onda, algo análogo al tratamiento de otras fuentes de datos cuando van a ser utilizadas en modelos de aprendizaje automático.

A nivel intuitivo, la **idea es, partiendo de la onda, tener un vector de variables que pueda servir de entrada de un modelo** de aprendizaje automático. Esto se lleva a cabo mediante distintas fases relacionadas con el **procesamiento de señales**: muestrear la señal para tener distintas muestras discretas (*sampling*), recoger la información de la amplitud de onda con el proceso de cuantificación (*quantify*), recorrer estos datos mediante ventanas suficientemente pequeñas para que el comportamiento de la señal sea estacionario dentro de ella, **y aplicar distintas transformaciones para convertir estos datos en un vector de variables**.

Puedes consultar más información sobre aspectos relacionados con el tratamiento de la señal y sobre la fonética en los Capítulos 25 y 26 del libro: *Jurafsky, D. y Martin, J. H. (2021). Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall. <https://web.stanford.edu/~jurafsky/slp3>

Arquitectura de un sistema de ASR

Una de las arquitecturas más habituales para ASR usando redes neuronales se basa en el uso de *encoders-decoders*, de modo similar a cómo ocurría con los modelos de MT. Partiendo de la señal acústica, se extraen los rasgos que sirven de entrada al modelo. La salida del modelo será o bien una predicción a nivel de palabras (qué palabras aparecen en función de la secuencia de entrada) o a nivel de caracteres o letras. El siguiente ejemplo muestra una arquitectura en la que la salida sirve para predecir el texto asociado a la señal de entrada a nivel de letras.

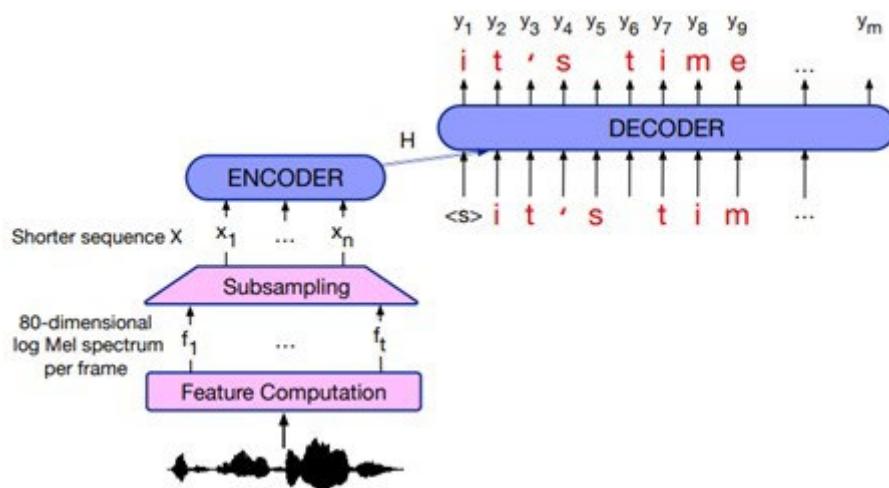


Figura 32. Ejemplo de arquitectura *encoder-decoder* para un sistema de ASR. Fuente: Jurafsky, 2021.

Si se trata de predecir a nivel de letras, se considera entonces un problema de clasificación en el que los posibles valores de salida corresponden a las distintas letras o caracteres de un alfabeto. Por ejemplo, los valores posibles de y serían (incluyendo letras, números y caracteres especiales):

$$y_i \in \{a, b, c, \dots, z, 0, \dots, 9, \langle \text{space} \rangle, \langle \text{comma} \rangle, \langle \text{period} \rangle, \langle \text{apostrophe} \rangle, \langle \text{unk} \rangle\}$$

Una consideración para tener en cuenta es que, aunque los *modelos encoding-decoding* son útiles cuando el número de datos de entrada no coincide con el de salida, el caso de ASR es especialmente acentuado en este aspecto, ya que una salida de caracteres asociados a una palabra (que podrían ser, por ejemplo, cinco para una palabra como perro), está asociada a una gran cantidad de variables de entrada. El motivo es que la señal acústica que asociada a esa palabra sería del orden de segundos (por ejemplo, 2 seg.), y las variables que se generan desde esa secuencia vendrían de hacer una partición en múltiples segmentos de mucha menor duración.

Por este motivo, se añade un paso adicional antes de llegar al modelo de *encoding* con el que comprime la información de entrada para que el vector de variables sea más reducido. Hecho esto, el resto del modelo funciona como un modelo estándar de *econding-decoding* como los ya vistos, donde se usa habitualmente la aproximación de *teacher forcing* en el entrenamiento.

Una consideración particular es que el problema de ASR es a nivel general un problema de modelado causal, donde solo está disponible la información de la secuencia previa para hacer las predicciones.

Por este motivo, una aproximación considerada es usar también la información de modelos de lenguaje entrenados para predecir la posible secuencia que continuaría a la ya detectada, y combinar ambos modelos en uno, usando una única función de coste.

Evaluación de un sistema de ASR

Una métrica de evaluación habitual para los ASR es la tasa de error por palabra (*word error rate*). Con esta métrica se analiza cuanto difiere la secuencia predicha de la secuencia real. En concreto, se contabiliza cuántas palabras adicionales hay en la

secuencia predicha que no están en la real (*insertions*), cuantas palabras hay en la secuencia real que no aparecen en la predicha (*deletions*), y cuántas palabras hay en la secuencia predicha que no coinciden con las de la original (*substitutions*). La fórmula que contabiliza todos estos aspectos sería:

$$\text{Word Error Rate} = 100 \times \frac{\text{Insertions} + \text{Substitutions} + \text{Deletions}}{\text{Total Words in Correct Transcript}}$$

Arquitectura de un sistema de TTS

Con un sistema de TTS **se parte de una cadena de texto, y se busca generar la señal acústica**. Esto es importante en distintas aplicaciones, como los agentes conversacionales, donde el sistema interactúa con un usuario y le habla con voz.

Como ocurre con otras aplicaciones de PLN, los sistemas de TTS también se pueden construir con un esquema *encoder-decoder*, donde la entrada es una cadena de texto y la salida la señal acústica.

Ahora bien, **hay una diferencia en la aproximación de un TTS frente a un ASR más allá de que un proceso sea el inverso del otro**. En el **caso del ASR, el sistema debería ser independiente del hablante**. Es decir, que se debe entrenar con audios de distintos hablantes para ser agnóstico al usuario concreto que se comunique con el sistema. **En el caso del TTS no es relevante que el sistema sea entrenado con datos de distintos hablantes, ya que lo que se busca es que se genere un tipo de voz fija**.

Por este motivo los TTS son dependientes del hablante, y se necesitan menos datos para entrenarlos que en el caso de los ASR.

Un sistema de TTS tiene **dos etapas de predicción**. En primer lugar, se parte de una **cadena de texto** para predecir el **espectrograma asociado a ella** (en concreto, una aproximación común es mapear letras al espectrograma de Mel). **En segundo lugar,**

los elementos del espectrograma se mapean a la señal acústica asociada con un proceso denominado **vocoding**.

En la arquitectura de un TTS hay un paso adicional a considerar que sirve para normalizar los caracteres especiales que pueda haber en el texto de entrada. Cuando aparecen números, caracteres como \$, signos de puntuación... se tienen que convertir a palabras para poder generar el audio correspondiente (ej., \$ a «dólares»). Ahora bien, en algunos casos esto dependerá del contexto de la frase. Por ejemplo, un mismo número, como por ejemplo 1910, se podría leer distinto dependiendo de si hace referencia a una contraseña (donde el texto sería «uno nueve uno cero»), o a una fecha (donde sería «mil novecientos diez»). Así, **aparecen distintos tipos de verbalizaciones posibles según el contexto**. Según los casos, habrá distintas verbalizaciones preferidas, identificadas como clases semióticas. La siguiente imagen muestra algunos de estos ejemplos.

semiotic class	examples	verbalization
abbreviations	gov't, N.Y., mph	government
acronyms read as letters	GPU, D.C., PC, UN, IBM	G P U
cardinal numbers	12, 45, 1/2, 0.6	twelve
ordinal numbers	May 7, 3rd, Bill Gates III	seventh
numbers read as digits	Room 101	one oh one
times	3.20, 11:45	eleven forty five
dates	28/02 (or in US, 2/28)	February twenty eighth
years	1999, 80s, 1900s, 2045	nineteen ninety nine
money	\$3.45, €250, \$200K	three dollars forty five
money in tr/m/billions	\$3.45 billion	three point four five billion dollars
percentage	75% 3.4%	seventy five percent

Figura 35. Distintos ejemplos de normalización en función de la verbalización buscada. Fuente: Jurafsky, 2021.

Esta normalización para generar el texto adecuado para la verbalización **se puede hacer con reglas predefinidas**. Con estas reglas se identifican estos caracteres y, en función de la clase semiótica detectada, se normalicen de una manera u otra para generar el texto final adecuado para la verbalización. Esta tarea de normalización también se puede llevar a cabo con *encoders-decoders* cuando se dispone de un corpus de entrenamiento con los pares de caracteres y verbalización correspondiente según el contexto.

Con todo ello, un ejemplo de arquitectura TTS es la **Tacotron2** (Shen et al., 2018), que aparece ilustrada en la siguiente imagen.

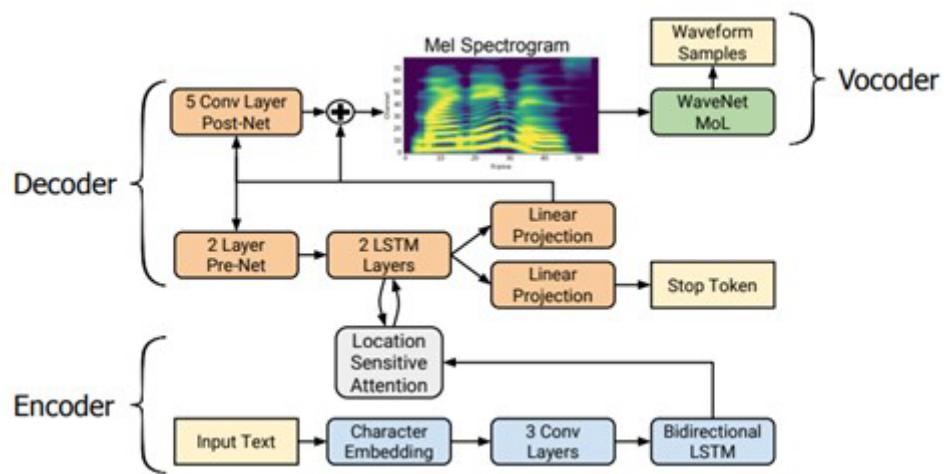


Figura 36. Arquitectura Tacotron2 para TTS. Fuente: Shen et al., 2018.

Como se puede apreciar, en la fase de *encoding* se utilizan distintas capas de redes neuronales para generar la representación de los textos de entrada, que se tratan a nivel de caracteres. Posteriormente, la fase de *decoding* genera el espectrograma que se usará de entrada en vocoder para generar la señal acústica. Un elemento adicional que tiene esta arquitectura es que, en cada fase, se lleva a cabo una predicción de **stop token** de cara a que el sistema decida si tiene que seguir produciendo secuencias de salida o no.

Puedes consultar más información sobre cómo es el bloque de vocoding y otros aspectos del sistema TTS, puedes consultar el capítulo 26 del libro: Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall. <https://web.stanford.edu/~jurafsky/slp3/26.pdf>

9.7. Referencias bibliográficas

Jurafsky, D. y Martin, J. H. (2021). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall.

Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang Y., Wang, Y., Skerry-Ryan, R., Saurous, R. A., Agiomyrgiannakis Y., and Wu, Y. (2018). *Natural TTS synthesis by conditioning WaveNet on mel spectrogram predictions*. ICASSP.

A fondo

Natural Language Processing with Python

Jurafsky, D. y Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall. <https://web.stanford.edu/~jurafsky/slp3/>

Una parte importante de las referencias del curso en general, y de este tema en particular, provienen de este libro. En concreto, para entender en más detalles las distintas aplicaciones de PLN, se puede consultar este libro. También, sirve de base para tener una referencia sobre las publicaciones científicas más relevantes asociadas a estas aplicaciones.

Deep Learning

Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep Learning*. MIT Press. <https://www.deeplearningbook.org/>

Como se ha visto en este tema, así como en temas previos, gran parte de las arquitecturas modernas de sistemas de PLN se basan en modelos de aprendizaje profundo. Si se quiere conocer más sobre estos modelos, cómo funcionan, el detalle de cómo se entrena se puede consultar esta referencia.

Test

1. Indica las afirmaciones verdaderas en el contexto de las tareas de MT:

 - A. Un buen sistema de MT puede centrarse sólo en la traducción individual de las palabras, sin considerar el contexto general de la frase.
 - B. Las traducciones entre dos lenguas son directas ya que las diferencias que pueda haber entre ellas no son relevantes para hacer una traducción.
 - C. Los aspectos de divergencia léxica hacen referencia a diferencias entre dos lenguas en relación con la variación en el significado que puede tener una misma palabra entre ellas.
 - D. Ninguna de las anteriores.
2. ¿Cuál es el objetivo de usar *teacher forcing* dentro de un modelo *encoder-decoder*?

 - A. Conseguir que la ejecución de la fase del *encoding* sea más rápida
 - B. Calcular la función de coste en el entrenamiento del modelo de manera que durante cada etapa t se calcule con respecto a la palabra real de la secuencia en $t-1$, y no la palabra que se predijo en $t-1$. Con ello, se busca evitar que se degraden la calidad de las predicciones a medida que se avanza en el *decoder*.
 - C. Reducir el tamaño del conjunto de entrenamiento para facilitar la fase de aprendizaje del modelo.
 - D. Mejorar la calidad de las predicciones del modelo *encoding-decoding* mediante el uso de LM causales tanto en la parte del *encoder* como en la parte del *decoder*.

3. ¿Qué diferencias existen entre una capa cross-attention y otra causal self-attention dentro de un bloque del decoder para un modelo de MT?

- A. La capa causal *self-attention* se encarga de recibir el contexto generado en el *encoder*, mientras que la capa *cross-attention* es una capa estándar *self-attention* que recibe como entrada la secuencia recorrida en el *decoder* hasta ese momento.
- B. La capa *cross-attention* se encarga de recibir el contexto generado en el *encoder*, mientras que la capa causal *self-attention* es una capa estándar que recibe como entrada la secuencia recorrida en el *decoder* hasta ese momento.
- C. La capa *cross-attention* es siempre la primera capa de los bloques del *decoder*, mientras que la capa causal *self-attention* es siempre la última.
- D. Ninguna de las anteriores.

4. ¿Cuál de estas afirmaciones es correcta con respecto a las diferencias entre la arquitectura de un modelo basado en *transformers* para autocompletado de textos frente a otro para la generación automática de resúmenes?

- A. En el caso del autocompletado de textos no se dispone desde el principio de la secuencia completa, por lo que el modelo de *transformers* seguirá un esquema como el de un LM causal. En cambio, en la generación de resúmenes sí que se dispone de todo el texto, con lo que el *transformer* puede ser bidireccional.
- B. En el caso de la generación automática de resúmenes no se dispone desde el principio de la secuencia completa, por lo que el modelo de *transformers* seguirá un esquema como el de un LM causal. En cambio, en el autocompletado de textos sí que se dispone de todo el texto, con lo que el *transformer* puede ser bidireccional.
- C. En el caso del autocompletado de textos se puede usar el concepto de *teacher forcing* en el entrenamiento, mientras que en la generación automática de resúmenes no es posible.
- D. Ninguna de las anteriores.

5. ¿Cómo se llevaría a cabo un análisis de sentimientos no supervisado basado la información de sentimientos a nivel de palabras disponible en un lexicón?

- A. Se generarían rasgos que caractericen el texto (ej., BoW), y con eso se entrenaría un modelo, como por ejemplo un Naive-Bayes, con el que predecir las clases (sentimientos) asociadas al texto.
- B. Se caracterizarían los sentimientos de las palabras que componen cada texto y, por ejemplo, se aplicarían reglas de clasificación que asignen un sentimiento a cada texto en función del sentimiento asociado a sus palabras individuales.
- C. Se generarían rasgos que caractericen el texto (ej., BoW), y con eso se entrenaría un modelo, como por ejemplo una Red Neuronal Recurrente, con el que predecir las clases (sentimientos) asociadas al texto.
- D. Ninguna de las anteriores.

6. ¿Cuál de estas preguntas sería un ejemplo de *factoid question*?

- A. ¿Cuál es la composición de la atmósfera de Marte?
- B. ¿Cuál es la mejor forma de viajar a Roma?
- C. ¿Qué me recomiendas para aprender PLN?
- D. Todas las anteriores.

7. ¿Para qué propósito se usa el *inverted index* dentro de un sistema de IR para *ad hoc retrieval*?
- De cara a mejorar el *scoring* de similitud entre el vector de la consulta y el vector de variables que representa el documento.
 - De cara a realizar una búsqueda más eficiente al usar un diccionario intermedio que indica los documentos que tienen ciertas palabras, calculando con ello la similitud entre consulta-documento solo si ese documento tiene palabras que aparezcan en la consulta.
 - De cara a poder utilizar técnicas de *embeddings* dentro del IR.
 - Ninguna de las anteriores.
8. Indica cuál de las siguientes afirmaciones es correcta en relación con un IR para *factoid QA*:
- Se desarrollan sistemas que llevan a cabo dos etapas: 1) Para una consulta de entrada, elegir el conjunto de documentos de la colección que podrían servir para responderla. 2) Llevar a cabo una tarea de comprensión donde se encuentre el fragmento de texto dentro de un documento que responda a la consulta.
 - Dentro de que se usan documentos de una colección para responder a preguntas de un usuario, el sistema tiene que poder identificar cuándo ninguno de los documentos tiene secuencias de texto que sirvan como respuesta a una consulta particular. Para ello, se usa como token por defecto el [CLS], de manera que si se predice ese token es que no se ha encontrado respuesta.
 - Debido a que la secuencia de entrada tiene un tamaño limitado, se tiene que recorrer cada documento con un tamaño de ventana acotado al buscar la secuencia de texto que pueda responder a la consulta de entrada.
 - Todas son correctas.

9. Indica cuál de estas afirmaciones es incorrecta sobre un sistema de ASR:

- A. La entrada corresponde a una señal de audio, de la que se obtienen distintos rasgos tras aplicar técnicas de muestreo, que sirven directamente de entrada a un modelo (ej., un *encoder*).
- B. Los rasgos asociados a la señal acústica de entrada dan lugar a un vector de variables de mucha mayor dimensionalidad que el vector de salida. Por este motivo se suele comprimir la información de entrada antes de que entre al modelo con el que se predice el texto.
- C. El problema de ASR es un problema causal, con lo que se pueden mejorar las predicciones del modelo combinándolo con un LM.
- D. Todas son incorrectas.

10. Indica cuál de estas afirmaciones es incorrecta sobre un sistema de TTS:

- A. En general, se puede considerar que un sistema de TTS es dependiente del hablante, es decir, no es necesario que sea entrenado en múltiples voces distintas.
- B. En la arquitectura de un modelo de TTS es necesario incluir un bloque denominado Vocoder para obtener la señal acústica asociada al espectrograma predicho por el modelo.
- C. Se suelen usar modelos *encoding-decoding* donde la salida de la última capa de la red neuronal del *decoder* corresponde con la predicción de la señal acústica asociada al texto de entrada.
- D. Todas son incorrectas.

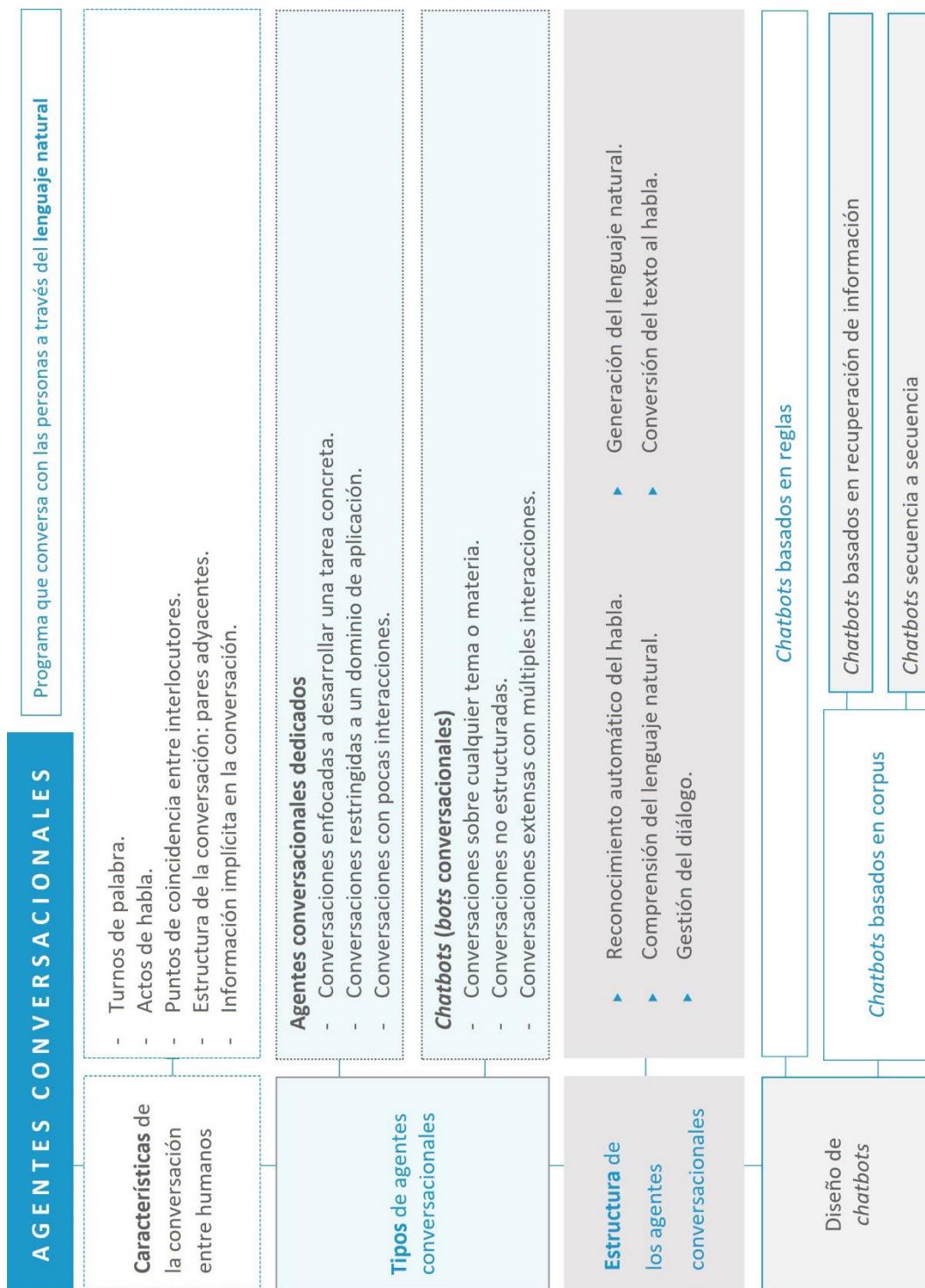
Procesamiento del Lenguaje Natural

Agentes conversacionales

Índice

Esquema	3
Ideas clave	4
10.1. Introducción y objetivos	4
10.2. Definición de agente conversacional	5
10.3. Características de las conversaciones entre humanos	6
10.4. Tipos de agentes conversacionales	14
10.5. Estructura de los agentes conversacionales	17
10.6. Diseño de <i>chatbots</i>	35
10.7. Referencias bibliográficas	42
A fondo	46
Test	48

Esquema



10.1. Introducción y objetivos

A continuación, se definirá lo que es un **agente conversacional**. Analizamos las características más relevantes de las conversaciones entre humanos que son las que determinan las características de los agentes conversacionales. Además, se describen los **agentes conversacionales dedicados** y los **chatbots**, dos tipos de agentes conversacionales que se diferencian por el tipo de conversación que son capaces de mantener y la funcionalidad que pretenden implementar. También, presentamos la estructura básica de un agente conversacional y se describe la utilidad de cada uno de los módulos que la componen. Por último, veremos los **diferentes enfoques de cómo diseñar un chatbot**.

Objetivos

- ▶ Conocer lo que es un agente conversacional.
- ▶ Conocer las características más relevantes de las conversaciones entre humanos que son las que determinan las características de los agentes conversacionales.
- ▶ Identificar los diferentes tipos de agentes conversacionales dependiendo del tipo de conversación que son capaces de mantener y la funcionalidad que pretenden implementar.
- ▶ Representar gráficamente la estructura básica de los agentes conversacionales y describir la utilidad de cada uno de los módulos que la componen.
- ▶ Conocer los diferentes enfoques utilizados para diseñar un *chatbot*.

10.2. Definición de agente conversacional

La Real Academia Española (RAE, s. f.) en su diccionario de la lengua española define la palabra diálogo:

«Del lat. dialōgus, y este del gr. διάλογος diálogos. 1. m. Plática entre dos o más personas, que alternativamente manifiestan sus ideas o afectos». Este mismo diccionario define la palabra plática: «De práctica. 1. f. conversación (|| acción de hablar)».

Los agentes conversacionales, también llamados sistemas de diálogo, son programas que conversan con las personas a través del lenguaje natural.

Los agentes conversacionales pueden interactuar con el humano, ya sea a través de la voz (hablando con el usuario), de texto, en el caso de que la conversación se lleve a cabo a través de un chat de texto, o utilizando ambas modalidades a la vez. **Los agentes conversacionales multimodales integran el habla con otras modalidades como los gráficos o las imágenes.**

Algunos agentes conversacionales modernos están empezando a incorporar las emociones, por ejemplo: sorpresa, enfado, alegría, tristeza o estrés, en la conversión de texto en habla para dar más realismo a la respuesta del agente conversacional. De hecho, las emociones también pueden formar parte del proceso de comprensión del lenguaje y la posterior generación del diálogo más adecuado a las emociones detectadas.

Las emociones se pueden detectar del propio contenido del mensaje, pero también se podrían extraer del análisis de los gestos y expresiones faciales de la persona que interactúa con el agente conversacional.

En el caso más sencillo de agente conversacional, este es capaz de contestar a las preguntas del usuario o de realizar una acción de control a partir de un comando de

voz. Este es el caso de los asistentes virtuales que tienen una única interacción con la persona. Sin embargo, también existen agentes conversacionales más complejos capaces de continuar con la interacción con el usuario y, por ejemplo, contestar a una segunda pregunta relacionada con la primera. Sería el caso de agentes conversacionales que ya incluyen diálogo y que se han aplicado en asistentes para planificar y gestionar viajes, como en el caso de GUS (Bobrow et al., 1977).

Los agentes conversacionales incluso pueden usarse en dominios mucho más complejos como los agentes conversacionales pedagógicos que actúan como **tutor del estudiante**, como sería el caso de ITSPOKE (Forbes-Riley y Litman, 2011). Por último, los agentes conversacionales se pueden usar simplemente como **divertimento para charlar con ellos**, pero sin que la idea sea realizar una tarea concreta, como en Cleverbot.

10.3. Características de las conversaciones entre humanos

Las principales características de los agentes conversacionales se deben a las características intrínsecas que tiene una conversación entre humanos. Una conversación es una acción que se realiza de forma compartida entre varias personas y que implica un alto grado de complejidad, ante esto, las conversaciones entre una persona y una máquina intentan **emular los comportamientos que se dan en las conversaciones entre humanos**.

Turnos de palabra

Una conversación se compone de **turnos de palabra** que se dan de **forma consecutiva** y que alternan las opiniones de los interlocutores. En el caso de que dos

personas mantengan una conversación, el interlocutor A declara algo, el interlocutor B le responde, luego el interlocutor A vuelve a expresarse, y así sucesivamente.

Las personas somos capaces de identificar fácilmente quién es la siguiente persona a la que le toca expresarse y el momento exacto en el que debe intervenir. Por lo tanto, durante una conversación hablada, los humanos somos capaces de esperar a que nuestro interlocutor acabe de hablar antes de responder, identificar que nuestro interlocutor ya ha acabado su frase y saber que somos nosotros los que debemos responder, determinar el momento preciso en el que debemos responder (por ejemplo, un instante después de que acabe de hablar nuestro interlocutor), o interpretar que un silencio muy prolongado de nuestro interlocutor puede significar que le incomoda la pregunta que le hemos realizado y que no la va a contestar.

En las conversaciones entre humanos, los **turnos de palabra están claros** y se asignan de forma intuitiva, sin embargo, para los agentes conversacionales llegar a manejar correctamente los turnos de palabra y gestionar eficientemente los tiempos es una tarea complicada. La gestión de los turnos de palabra en los agentes conversacionales se estudia en el campo del **análisis conversacional** (Levinson, 1983) y se basa en la idea de que el intercambio de turnos se puede modelar con una serie de reglas. Estas reglas, que indican cómo manejar los turnos de palabra, se evalúan en los momentos en que la estructura del lenguaje permite intercambiar el interlocutor, momentos llamados en inglés *transition-relevance places (TRP)* (Sacks, Schegloff y Jefferson, 1974).

Actos de habla

Se ha comentado anteriormente que una conversación se compone de **una secuencia de turnos de palabra**. Además, se puede añadir que cada uno de los turnos incluye una o varias expresiones o declaraciones. La teoría de los actos de habla sugiere que **cada expresión en una conversación es un tipo de acción que realiza el interlocutor** (Austin, 1962). Esta teoría fue desarrollada por el filósofo británico John L. Austin

antes de morir en 1960 y presentada en su obra póstuma, en la cual se acuñó el término acto de habla.

Un acto de habla es cada uno de los tipos de acciones que involucran el uso del lenguaje natural.

En una expresión tal como:

- ▶ A mi nuevo gato lo he llamado Felix.

La acción de llamar a mi gato Felix tiene la consecuencia de que el gato tenga el nombre de Felix. En este caso queda claro que la acción que realiza el interlocutor y que se recoge en la expresión ha permitido cambiar el estado del mundo haciendo que el gato que no tenía nombre tenga ahora uno. Sin embargo, el trabajo de Austin sugiere que todas las expresiones tienen asociado algún acto de habla, aunque este no sea tan evidente como el descrito anteriormente.

Los actos de habla definidos por Austin se han ido extendiendo con el paso de los años y hoy en día podemos hablar de actos que cambian el estado del mundo, sugieren al interlocutor que haga algo, le dan directrices para que haga algo, prometen o planifican que el interlocutor va a hacer algo en el futuro o expresan el estado psicológico o la actitud del interlocutor sobre una acción.

En una conversación los humanos somos capaces de hacer preguntas, dar órdenes o informar sobre algo sin darnos cuenta de que nuestras expresiones contienen diferentes tipos de los actos de habla definidos por Austin. Sin embargo, para los agentes conversacionales no es trivial saber interpretar en cada momento en qué acto de habla se encuentran y tomar las decisiones correctas para responder de la forma más adecuada.

No se puede afirmar que todos los agentes conversacionales sean capaces de tratar con múltiples de actos de habla. De hecho, solo algunos agentes

conversacionales modernos implementan funcionalidades más avanzadas y son capaces de lanzar una pregunta, dar una orden, hacer una propuesta, rechazar una sugerencia o estar de acuerdo con una expresión presentada por su interlocutor.

Puntos de coincidencia entre interlocutores

Como hemos visto, cada turno de palabra y, más en concreto, cada expresión del turno se puede ver como una acción de un interlocutor. Sin embargo, un diálogo no se compone de una serie de actos independientes sin relación alguna. Sino que, a diferencia de en un monólogo, **una conversación es un acto colectivo** realizado entre ambos interlocutores. Debido a esta acción conjunta que es una conversación, **se deben establecer constantemente unos puntos de coincidencia entre los interlocutores** (Stalnaker, 1978).

Los puntos de coincidencia son el conjunto de fundamentos en los que creen los interlocutores que participan en una conversación.

Es necesario asegurar que se han establecido unos puntos de coincidencia entre interlocutores para que la conversación pueda desarrollarse con éxito. Esto significa que el receptor del mensaje debe dejar claro que ha entendido el significado y la intención de la expresión lanzada por el emisor, y también el caso contrario, indicar que no ha entendido al emisor del mensaje (Clark, 1996).

En las conversaciones entre humanos **se utilizan diferentes técnicas para garantizar que se están estableciendo los puntos de coincidencia entre interlocutores** (Clark y Schaefer, 1989). En una conversación, el receptor del mensaje puede asentir o utilizar palabras tipo «vale» o «genial» u onomatopeyas tipo «ajá» para mostrar que está entendiendo y, además, está de acuerdo con su interlocutor.

Otras opciones que garantizan los puntos de coincidencia serían que el receptor del mensaje **reproduzca textualmente, reformule o parafrasee lo que le ha transmitido**

su interlocutor o que le ayude a acabar de construir su argumentación. Además, el receptor del mensaje puede mostrar interés por la conversación y entonces se interpreta directamente que está conforme con lo que le comenta su interlocutor. Por último, el receptor puede simplemente pasar a proponer su siguiente contribución al diálogo, por lo que se supone que está de acuerdo con lo que se ha comentado hasta ese momento.

Es imprescindible que los agentes conversacionales imiten todos los comportamientos que se acaban de describir para poder asegurar que se establecen puntos de coincidencia entre el agente y su interlocutor humano. De hecho, se ha demostrado que, si el agente no garantiza que ha entendido al humano, va a crear confusión a la persona y esta no se va a sentir cómoda utilizando el agente conversacional (Yankelovich, Levow y Marx, 1995).

Ejemplo ilustrativo 1

Posibles conversaciones de un asistente virtual que ayuda a un cliente de una operadora de telefonía a configurar su perfil de usuario, consultar su consumo, etc.

Conversación A:

Asistente: ¿Desea revisar algo más de su perfil de usuario?

Cliente: No.

Asistente: ¿Qué desea hacer ahora?

Conversación B:

Asistente: ¿Desea revisar algo más de su perfil de usuario?

Cliente: No.

Asistente: Entonces ¿qué desea hacer ahora?

En el ejemplo de la conversación A, el cliente no puede asegurar que el asistente virtual haya entendido que le ha dicho que no porque este agente conversacional no le ha corroborado la recepción de la información, ya sea de forma explícita o implícita. Entonces, se puede afirmar que en la conversación A no se ha establecido un punto de coincidencia entre la persona y el agente conversacional.

Sin embargo, en la conversación B, por el mero hecho de que el asistente virtual haya utilizado el adverbio «entonces», se sobreentiende que el agente conversacional ha entendido que el cliente ha respondido que no y le está lanzando una nueva pregunta. En este caso, sí se ha establecido el punto de coincidencia entre la persona y el agente conversacional, tan necesario para el correcto funcionamiento del agente.

Estructura de la conversación

Se ha comentado en los apartados anteriores que la conversación se compone de una serie de turnos de palabra relacionados entre sí y en los que los interlocutores exponen diferentes expresiones. También, se ha explicado que la conversación es un acto colectivo en el que requiere que ambos interlocutores lleguen a establecer una serie de puntos de coincidencia. Para asegurar los fundamentos de la conversación y garantizar que el receptor del mensaje ha entendido al emisor, se han presentado algunas técnicas en las que se veía una clara correlación entre los actos de un turno de palabra y los del siguiente turno. De hecho, aunque las conversaciones tienen una estructura libre, se pueden identificar algunos patrones bien definidos en algunas partes de los diálogos entre humanos.

En numerosas ocasiones se va a dar el caso de que un turno de palabra esté relacionado con el siguiente. Por ejemplo, cuando se lanza una pregunta en un turno, se condiciona claramente a que en el siguiente turno de palabra se dé una respuesta. Es por esta razón que se puede hablar de que en las conversaciones se da un patrón de pares adyacentes (Schegloff, 1968).

Los pares adyacentes son estructuras que se dan en una conversación del tipo pregunta-respuesta (pregunta seguida de una respuesta), propuesta-aceptación o propuesta-rechazo, petición-concesión, saludo-saludo.

Según el patrón de los pares adyacentes, en una conversación en la que se lance una propuesta (primera parte del par), acto seguido va a darse una aceptación o un rechazo (segunda parte del par). Sin embargo, se puede dar el caso de que, en un

diálogo, la segunda parte del par no aparezca directamente justo después de la primera parte, sino que haya algunas expresiones intermedias. Esta situación se podría dar si el interlocutor necesita hacer una pregunta aclaratoria para recabar información antes de poder aceptar o rechazar la propuesta que le había lanzado el otro interlocutor. En este caso se daría un **subdiálogo** entre la propuesta (primera parte del par) y la aceptación o rechazo de la misma (segunda parte del par).

Aparte de la estructura de la conversación marcada por los pares adyacentes, se pueden encontrar otros patrones en **cómo se organiza una conversación**. Por ejemplo, en una llamada de teléfono siempre habla primero la persona que responde al teléfono, luego lo hace la persona que llama indicándole quién es; en los pasos siguientes, muy posiblemente, se intercambien saludos y luego, la persona que llama indica el motivo de la llamada. Este patrón da comienzo a una llamada telefónica y es solo un ejemplo concreto, pero podríamos tener también un patrón para el final de la llamada. Además, en otros tipos de conversaciones podríamos encontrar otros patrones que describan la estructura de la conversación.

Los agentes conversacionales se van a beneficiar de los patrones en la estructura de la conversación para facilitar la generación de diálogos de forma automática. Por ejemplo, saber que la persona ha lanzado una pregunta y que, por lo tanto, el agente debe proporcionar una respuesta será una información muy útil a lo hora de crear un diálogo. Además, si se sabe que se está desarrollando un agente que responde a llamadas telefónicas, se podrá prefijar el patrón que dé comienzo o que finalice la conversación.

Información implícita en la conversación

En los apartados anteriores se han tratado las características de la conversación relacionadas con lo que se podría llamar la «infraestructura» de la conversación:

- ▶ La **conversación** es una **actividad que realizan de forma conjunta** varios interlocutores.

- ▶ Estos generan **turnos de palabra** en los que se exponen una serie de contribuciones que requieren la aceptación por parte del interlocutor y que cumplen con unos patrones predeterminados.

Sin embargo, hasta este punto no se ha comentado nada acerca de la información que se comunica entre los interlocutores durante el diálogo.

En una conversación, el significado de una contribución no se limita al significado de las palabras entendidas de forma individual. Es más, este significado tampoco se limita al significado que pueda tener una expresión concreta.

Esto se debe a que la interpretación de una expresión se basa en algo más que el significado literal de las oraciones que la componen.

En la interpretación de una expresión intervienen el conocimiento que comparten los interlocutores, el contexto en el que se desarrolla la conversación o la intención del emisor del mensaje. Es por eso por lo que se puede decir que en una conversación **hay información** que tiene un **significado literal o explícito**, pero también hay otra información que el receptor del mensaje es capaz de inferir, por lo tanto, **información implícita**.

Las implicaturas o informaciones implícitas son los significados que no aparecen de forma explícita en la conversación, pero que el receptor del mensaje infiere.

La inferencia juega un papel crucial en una conversación, el emisor de un mensaje puede hacer llegar al receptor más información de la que le transmite de forma explícita en las expresiones que componen su mensaje. Por ejemplo, si para hacer una reserva de un vuelo, un agente de viajes le pregunta a su cliente que qué día quiere viajar y este le responde que tiene que llegar para una reunión el 15 de mayo, entonces, el agente de viajes va a interpretar que tiene que reservar el vuelo cuya llegada se dé el día 14 de mayo. En este caso se ve cómo claramente el cliente no ha

respondido al agente de viajes con la información que este le había preguntado, sino que le ha proporcionado otra información que de forma implícita responde a su pregunta. De hecho, en este caso el cliente ha decidido de forma intencionada proporcionarle al agente una información explícita que le permita a través de un proceso de inferencia obtener la información implícita requerida.

Los agentes conversacionales también deben ser capaces de extraer la información implícita que les hace llegar un humano cuando mantienen un diálogo.

Para ello, los agentes implementan mecanismos de inferencia que les permiten obtener los significados adicionales de un mensaje a partir del significado literal de las expresiones que lo componen.

Además, los agentes conversacionales para poder funcionar correctamente utilizando las informaciones implícitas deben ser capaces de determinar la intención de su interlocutor. El agente conversacional debe interpretar que el humano no se ha equivocado porque no haya respondido explícitamente la pregunta que le había formulado, sino que lo ha hecho a propósito porque la intención de la persona era transmitir al agente una información implícita.

La inferencia de información implícita en la conversación es una de las tareas más complicadas a las que se enfrenta un agente conversacional a la hora de comportarse como una persona y de reproducir las características que tiene una conversación entre humanos.

10.4. Tipos de agentes conversacionales

Existen dos tipos de agentes conversacionales: los **agentes conversacionales dedicados a una tarea concreta** y los **chatbots**. Esta clasificación se hace con base en

el tipo de conversación que son capaces de mantener los agentes conversacionales y la funcionalidad u objetivo final que pretenden implementar.

Agentes conversacionales dedicados

Los agentes conversacionales dedicados a una tarea concreta están **diseñados para desarrollar una tarea en particular** y están restringidos a un dominio de aplicación.

Esta única tarea que puede realizar este tipo de agentes sería, por ejemplo, obtener algún tipo de información del usuario o ayudarle a completar una acción sencilla.

Los agentes conversacionales dedicados están configurados para mantener conversaciones cortas con la persona, en general, conversaciones que involucren desde una interacción a unas pocas de interacciones (en el rango de media docena).

Además, cabe resaltar que las conversaciones mantenidas por estos agentes son **sencillas** y no se realizan de una manera natural como cuando conversan dos humanos.

Los asistentes personales que encontramos en los teléfonos móviles o en algunos dispositivos inteligentes para el control domótico (Google Assistant, Siri de Apple, Cortana de Windows o Alexa de Amazon) son agentes conversacionales que forman parte de la categoría de agentes conversacionales dedicados a una tarea concreta.

Estos pueden realizar tareas simples como, por ejemplo, dar instrucciones sobre un viaje, controlar electrodomésticos, encontrar restaurantes, ayudar a hacer llamadas telefónicas o enviar mensajes de texto.

Muchas empresas implementan agentes conversacionales dedicados en sus sitios web para ayudar a los clientes a responder a sus preguntas y dudas para resolver los problemas que pudieran tener con los productos o servicios contratados. Además, los agentes conversacionales juegan un papel importante como **interfaz hombre-máquina** en algunos robots.

Chatbots

Los *chatbots*, llamados en español *bots* conversacionales, son aquellos agentes conversacionales que permiten entablar conversaciones extensas, es decir, con múltiples interacciones entre la persona y el agente conversacional.

Permiten mantener conversaciones no estructuradas, una característica fundamental de las conversaciones entre personas, y sobre cualquier tema o materia a diferencia de agentes conversacionales dedicados a una tarea concreta.

Por tanto, los *chatbots* ofrecen una conversación realista entre una persona y el agente conversacional utilizando el lenguaje natural.

Ejemplo ilustrativo 2

Mitsuku: el *chatbot* ganador del Premio Loebner 2017

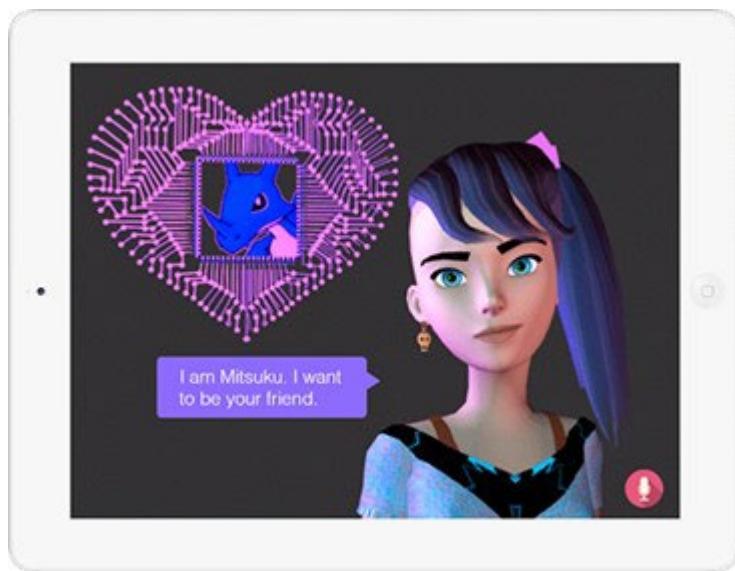


Figura 1. Avatar del chatbot Mitsuku. Fuente: Pandorabots, 2017.

Mitsuku está considerado como el mejor *chatbot* en la actualidad, ya que ha sido el ganador del Premio Loebner en las ediciones 2017, 2016 y 2013.

El Premio Loebner es una competición anual que premia a la aplicación de inteligencia artificial más inteligente del mundo. Para determinar el ganador de

la competición lo que se hace es aplicar el test de Turing. A través de este test, un juez humano tiene que decidir si detrás de la pantalla a la que realiza preguntas está un programa de ordenador o un ser humano. Mitsuku consiguió superar el test de Turing gracias a que respondió de forma meditada y a un ritmo razonable y, además, contestó las preguntas de forma convincente.

Puedes observar el funcionamiento de Mitsuku, en el siguiente enlace:
<https://www.pandorabots.com/mitsuku/>

Aunque en la prensa y la industria se utilice la palabra *chatbot* para nombrar cualquier agente conversacional, en este curso solo se va a utilizar este término para designar al grupo de agentes conversacionales que pueden mantener una conversación prolongada y realista sobre cualquier tema, tal como se hace en la comunidad del procesamiento del lenguaje natural. Por lo tanto, Mitsuku es considerado un *chatbot*, pero no Google Assistant, aunque sea capaz de desarrollar llamadas de forma fluida según las últimas funcionalidades presentadas en el congreso de desarrolladores Google I/O que tuvo lugar en mayo de 2018.

10.5. Estructura de los agentes conversacionales

Los agentes conversacionales, aunque puedan utilizar diferentes modalidades para interactuar con el usuario e implementen **diferentes métodos para gestionar el diálogo**, siguen una **estructura básica común**.

La Figura 2 muestra la arquitectura de un agente conversacional basado en voz:

- ▶ Un **módulo de entrada** para el reconocimiento automático de la voz.
- ▶ Un **módulo de comprensión** del lenguaje natural.
- ▶ Un **módulo de gestión** del diálogo.
- ▶ Un **módulo de generación** del lenguaje natural.
- ▶ Un **módulo de una salida** de conversión de texto al habla.

La arquitectura para un agente conversacional basado en texto sería equivalente, aunque el módulo de entrada implementaría el reconocimiento automático de texto y el módulo de salida renderizaría el texto a un formato de presentación gráfica adecuado.

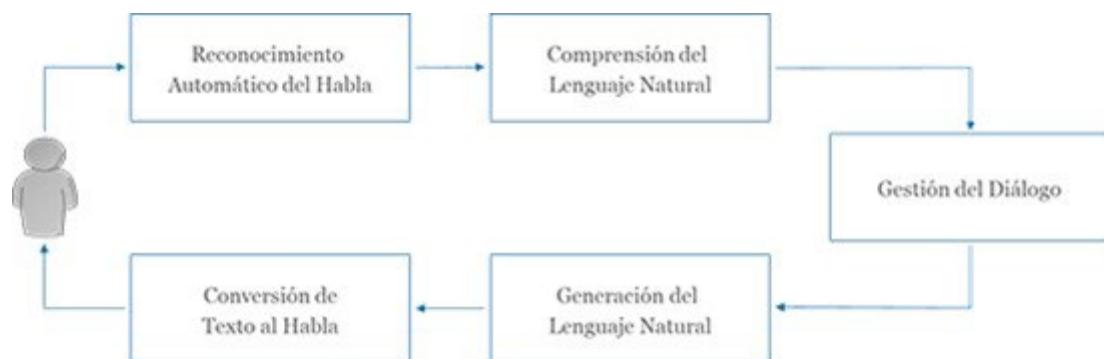


Figura 2. Arquitectura de un agente conversacional. Fuente: ¿

Reconocimiento automático del habla

El módulo de entrada tiene como función el **reconocimiento automático del habla**, es decir, **extraer las palabras a partir de la señal de audio** que se ha grabado con un **micrófono y que recoge la voz del usuario**. Aquí se utilizan tecnologías estándar para el modelado acústico-fonético, sin embargo, otros aspectos se pueden optimizar para hacer que su implementación sea más adecuada a los agentes conversacionales. Los métodos utilizados para el reconocimiento del habla no son objeto de estudio en este curso, aunque en los siguientes párrafos se describen brevemente algunas **estrategias** que permiten optimizar estos métodos.

Si el agente conversacional está dedicado a una tarea concreta y, por tanto, no se espera que vaya a responder a cualquier pregunta, solo aquellas que tienen que ver con la aplicación concreta, se puede hacer que solo sea capaz de transcribir frases en ese ámbito. Con este fin, los agentes conversacionales pueden utilizar modelos basados en gramáticas de estados finitos que especifican todas las respuestas posibles que el sistema entiende.

Otra opción para optimizar los modelos de lenguaje es que los agentes conversacionales utilicen un modelo de lenguaje N-gram en el cual las probabilidades en el diálogo de estados se adaptan para incorporar las restricciones concretas relacionadas con el contexto del diálogo.

Comprendión del lenguaje natural

El módulo de comprensión del lenguaje natural realiza las tareas básicas de procesamiento del lenguaje natural para extraer la semántica de las frases que recibe del módulo de reconocimiento automático del habla. Para representar la semántica se pueden utilizar diferentes alternativas y en el proceso de extracción se usan algunas de las técnicas de procesamiento del lenguaje natural que se han explicado en los capítulos anteriores.

En el caso de los agentes conversacionales dedicados, lo más común es utilizar *frames* y pares atributo-valor para representar la semántica de las frases. Entonces, estos agentes conversacionales basados en *frames* utilizan normalmente una ontología para la representación formal de la semántica de la conversación. Esta ontología define los diferentes *frames*, es decir, las posibles plantillas para una frase, una colección de los huecos (*slots*) en cada plantilla y los posibles valores para rellanar estos huecos.

Los pares atributo-valor restringen la tipología semántica de los valores que pueden ir asociados a cada hueco en la plantilla.

El módulo de comprensión del lenguaje natural de un agente conversacional basado en *frames* extrae la semántica de la petición lanzada por el usuario identificando diferentes elementos. De hecho, si el agente conversacional puede realizar tareas en múltiples dominios, **primero identifica el dominio al que hace referencia el usuario.**

Por ejemplo, detecta si el usuario está hablando de reservar vuelos, programando la alarma de un reloj o gestionando las citas de su calendario. Evidentemente, en los

agentes dedicados que trabajan en un único dominio no sería necesario realizar este paso. Sin embargo, la mayoría de los agentes conversacionales modernos tipo Siri de Apple, Alexa de Amazon o Google Assistant realizan varías tareas en algunos dominios predefinidos y necesitan implementar esta funcionalidad para clasificar la petición del usuario de acuerdo con el dominio al que hace referencia.

Una vez se ha identificado el dominio en el que se clasifica la petición lanzada por el usuario, lo siguiente es determinar la **intención del usuario**. Es decir, se tiene que identificar cuál es el objetivo del usuario y qué tarea pretende que realice el agente conversacional. Por ejemplo, en el dominio de la búsqueda de vuelos, la intención del usuario podría ser mostrar todos los vuelos entre dos ciudades para una fecha en concreto. En el otro ejemplo, el agente conversacional que gestiona la alarma de un reloj, las tareas que podría realizar serían activar la alarma a una hora concreta o desactivarla dependiendo de la intención del usuario.

Finalmente, cuando el agente conversacional ya ha detectado el dominio y la intención del usuario, a partir de la petición lanzada por el usuario también debe identificar los huecos (slots) en la plantilla que define la tarea que tiene que realizar el agente conversacional y obtener los valores necesarios para llenar esos diferentes huecos de la plantilla.

El análisis semántico realizado por el módulo de comprensión del lenguaje natural requiere no solo extraer y representar el significado de la oración, incluyendo el dominio, intención y valores de los huecos del *frame*, sino también identificar otros aspectos propios de un diálogo como son **los puntos de coincidencia**. Entonces, en el proceso de extracción de la semántica de la petición lanzada por el usuario se deben identificar también los **turnos de confirmación, negación o cortesía**. Por ejemplo, si el módulo de comprensión del lenguaje natural recibe la frase:

- ▶ Sí, me gustaría saber el precio del vuelo de Madrid a Logroño.

Este debe identificar que la palabra «Sí» al inicio de la frase se refiere a una afirmación y es independiente de la tarea que debe realizar el agente conversacional, que es la consulta del precio de un vuelo concreto.

Ejemplo ilustrativo 3

Representación semántica utilizando *frames* de una interacción con un agente conversacional dedicado a la reserva de vuelos.

El agente conversacional basado en *frames* define con una ontología los pares atributo-valor, es decir los huecos (*slots*) en la plantilla y los posibles valores para llenar estos huecos. La Tabla 1 muestra algunos ejemplos de pares atributo-valor definidos para el agente conversacional que reserva vuelos:

Hueco (slot)	Valor (type)
ORIGIN_CITY	city
DESTINATION_CITY	city
DEPARTURE_DATE	date
ARRIVAL_DATE	date
DEPARTURE_TIME	time
ARRIVAL_TIME	time

Tabla 1. Pares atributo-valor definidos para el agente conversacional que reserva vuelos. Fuente: elaboración propia.

Se observa en la Tabla 1 que el hueco llamado ORIGIN_CITY y el llamado DESTINATION_CITY deben tener un valor que sea una ciudad (*city*). Por tanto, si aparecen esos huecos en la plantilla, solo se pueden llenar con nombres de ciudades.

Además, la ontología que define los pares atributo-valor para los huecos y sus valores puede proporcionar una estructura jerárquica. Así, el valor *date* para los huecos DEPARTURE_DATE y ARRIVAL_DATE de la Tabla 1 puede ser a su vez el siguiente *frame* con valores del tipo entero (*INTEGER*), nominal (*NAME*) o miembros de un conjunto de nombres (*MEMBER*):

```
DATE
MONTH NAME
DAY (BOUNDED-INTEGER 1 31)
YEAR INTEGER
WEEKDAY (MEMBER (SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY
SATURDAY))
```

Cuando en un turno de palabra el módulo de comprensión del lenguaje natural recibe la petición en inglés «*Show me flights from Boston to San Francisco on Tuesday morning*», lo primero que hace es identificar el dominio al que hace referencia el usuario: el de los viajes en avión (*AIR-TRAVEL*). Además, este módulo identifica que la intención del usuario es ver los vuelos disponibles y que, por lo tanto, la tarea que debe realizar es mostrar los posibles vuelos (*SHOW-FLIGHTS*).

El módulo de comprensión del lenguaje natural es también capaz de identificar varios huecos y obtener sus valores a partir de la petición lanzada por el usuario. Este módulo extrae la información que el origen del vuelo es Boston y el destino es San Francisco, y determina que la fecha del vuelo es el martes (*tuesday*) por la mañana (*morning*).

Entonces, el módulo de comprensión del lenguaje natural basado en *frames* genera la siguiente representación semántica de la petición en inglés «*Show me flights from Boston to San Francisco on Tuesday morning*»:

```
DOMAIN: AIR-TRAVEL
INTENT: SHOW-FLIGHTS
ORIGIN_CITY: Boston
DEPARTURE_DATE: Tuesday
DEPARTURE_TIME: morning
DESTINATION_CITY: San Francisco
```

El módulo de comprensión del lenguaje natural de un agente conversacional basado en *frames* extrae la semántica de la petición lanzada por el usuario aplicando algunas de las técnicas de procesamiento del lenguaje natural que se han explicado en los capítulos anteriores. Por ejemplo, algunos agentes conversacionales como el Core Language Engine utilizan gramáticas de unificación con la semántica adjunta para identificar el significado de la frase y, a partir de esta, se extraen las posibles palabras que pueden completar los huecos en el *frame*.

Muchos de los agentes conversacionales comerciales en la actualidad y también el clásico agente conversacional para la reserva de viajes GUS (Bobrow et al., 1977) utilizan analizadores semánticos basados en **gramáticas semánticas**. Estas pertenecen al grupo de las gramáticas libres de contexto y tienen como característica que en la parte izquierda de las reglas que componen la gramática aparecen los nombres de los posibles huecos en las plantillas (*frames*).

Ventajas

La ventaja de las soluciones basadas en gramáticas semánticas es que el análisis semántico se puede implementar utilizando cualquier algoritmo estándar que permita realizar el análisis basado en una gramática libre de contexto o basado en programación dinámica como, por ejemplo, CKY.

El resultado del analizador es la cadena de caracteres de entrada etiquetada jerárquicamente con nodos semánticos. Muchas soluciones para la comprensión del lenguaje natural utilizan gramáticas semánticas porque algunos nodos obtenidos del análisis se corresponden directamente con los huecos en un *frame*.

Ejemplo ilustrativo 4

Gramática semántica utilizada por el módulo de comprensión del lenguaje natural para la identificación de la intención y la obtención de los huecos y sus valores a partir de una petición lanzada por el usuario.

Para obtener la representación semántica de la petición en inglés «*Show me flights from Boston to San Francisco on Tuesday morning*» presentada en el ejemplo ilustrativo 3, es necesario que el módulo de comprensión del lenguaje natural basado en *frames* trabaje con una gramática semántica.

A continuación, se presenta un fragmento de una gramática semántica que permite realizar el análisis semántico de la petición del usuario que quiere ver distintos vuelos:

```

SHOW → show me | i want | can i see | ...
DEPART_TIME_RANGE → (after | around | before) HOUR |
morning | afternoon | evening
HOUR → one | two | three | four ... | twelve (AMPM)
FLIGHTS → (a) flight | flights
AMPM → am | pm
ORIGIN → from CITY
DESTINATION → to CITY
CITY → Boston | San Francisco | Denver | Washington

```

Entonces, utilizando cualquier analizador para gramáticas libres de contexto y que permita generar un árbol a partir de la gramática anterior, se obtendría la estructura jerárquica de las etiquetas semánticas resultantes de analizar la petición del usuario. Considerando la gramática presentada justo arriba y la petición «*Show me flights from Boston to San Francisco on Tuesday morning*», la figura 3 presenta el resultado del análisis semántico utilizando la gramática semántica.

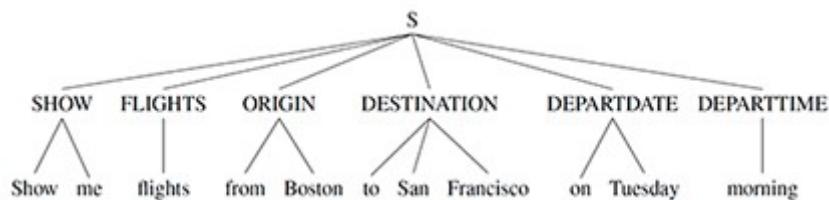


Figura 3. Resultado del análisis semántico utilizando la gramática semántica que permite crear un árbol donde los nodos son los posibles huecos de los frames. Fuente: Jurafsky y Martin, 2009.

Por ejemplo, en la Figura 3 se observa que del nodo ORIGIN en el árbol se puede obtener el valor a asignar al hueco ORIGIN_CITY en el *frame* que representa la petición del Ejemplo ilustrativo 3. Además, la intención de la petición se puede obtener de la combinación de los nodos SHOW y FLIGHTS.

Problemas

Aunque muchas soluciones para la comprensión del lenguaje natural utilizan gramáticas semánticas debido a sus ventajas y su alta precisión para un dominio restringido, las gramáticas semánticas tienen varios problemas.

- ▶ El primero es que no funcionan cuando se dan ambigüedades. Para solucionarlo, se pueden añadir probabilidades en la gramática semántica y utilizar un modelo basado en las gramáticas libres de contexto probabilísticos.
- ▶ Otro problema de las gramáticas semánticas es que estas se tienen que generar de forma manual, lo que es lento y costoso. Como alternativa existe la probabilidad de utilizar aprendizaje automático supervisado para aprender las reglas, suponiendo que se disponga de un conjunto de entrenamiento donde cada oración está etiquetada con la semántica correcta, se puede entrenar un clasificador para que asigne a cada oración su dominio y sus intenciones, y generar un modelo que mapee de la frase a los valores de los diferentes huecos en el *frame*.

Una alternativa a las gramáticas semánticas que se podría implementar en el módulo de comprensión del lenguaje natural sería un modelo **HMM semántico** (Pieraccini y Levin, 1991); este método probabilístico evita que se tengan que codificar las gramáticas de forma manual.

En este modelo HMM semántico los estados ocultos son las etiquetas de los huecos, mientras que las palabras observadas serían los valores con las que se podrían llenar los huecos.

Se acaba de describir en detalle el funcionamiento del módulo de comprensión del lenguaje natural para agentes conversacionales basados en *frames*. Los agentes conversacionales comerciales actuales utilizan esa arquitectura, sin embargo, en el ámbito de la investigación se están desarrollando agentes más modernos que implementan una arquitectura basada en el diálogo o, mejor dicho, en las características del diálogo.

Este nuevo tipo de agentes conversacionales basados en el diálogo permiten, además de hacer preguntas, realizar otro tipo de acciones como informar de un hecho, sugerir algo o dar órdenes. Por lo tanto, estos agentes necesitan identificar los diferentes actos de habla que se dan en la conversación con el usuario. De hecho,

la identificación de los actos de habla permite saber si el usuario simplemente está proporcionando una información al agente conversacional, si está afirmando o confirmado algo, haciendo una pregunta, etc.

El módulo de comprensión del lenguaje natural en un agente conversacional basado en el diálogo debe identificar los actos de habla, además de obtener los valores de los huecos del *frame* para representar el significado de la interacción.

Primero, se identifican los actos de habla aplicando técnicas de aprendizaje supervisado. Se entrena un clasificador a partir de un corpus etiquetado donde cada expresión se ha anotado a mano con su acto de habla. Para entrenar, se utiliza una gran variedad de características: características léxicas (bigramas; palabras como «no», que indican negación; o «muéstrame», que indica una petición; la longitud de la expresión o la puntuación) o características semánticas (contexto del diálogo o el acto de habla de la interacción previa).

Luego se obtienen los valores de los huecos, tal como se hacía en el caso de los agentes conversacionales basados en *frames*. En estos módulos se tienen en cuenta otros aspectos propios del diálogo como son los puntos de coincidencia.

Entonces, en el proceso de extracción de la semántica de una interacción con el agente conversacional se identifican también los turnos de confirmación, negación o cortesía.

Como hemos ejemplificado antes, si el módulo de comprensión del lenguaje natural recibe la frase «Sí, me gustaría saber el precio del vuelo de Madrid a Logroño», este debe identificar que la palabra «Sí» al inicio de la frase se refiere a una afirmación y que es independiente de la petición de la consulta del precio del vuelo.

Gestión del diálogo

El módulo de gestión del diálogo, también llamado **controlador del dialogo**, es el elemento principal de los agentes conversacionales y se encarga de **identificar la acción** que debe realizar el agente conversacional en el **siguiente turno de palabra y cómo se debe continuar la conversación**. Para ello, este módulo analiza la representación semántica de la petición que ha efectuado el usuario del agente conversacional o de la frase que ha pronunciado la persona.

Así, analiza la representación obtenida por el módulo de comprensión del lenguaje natural y proporciona a su salida la acción a realizar o el concepto que se quiere transmitir al usuario del sistema conversacional en el siguiente turno de palabra y como respuesta a la petición realizada por el usuario.

La gestión del diálogo requiere **mantener el estado y el flujo de la conversación** y en algunos casos pedir más información al usuario para poder completar la información necesaria y dar una respuesta. Por ejemplo, si el usuario ha realizado la petición

- ▶ «Me gustaría saber el precio del vuelo de Madrid a Logroño.

El agente conversacional solo sabrá a qué vuelo se refiere el usuario si mantiene el estado de la conversación y en alguna interacción anterior este le ha dado más detalles sobre el vuelo en cuestión, por ejemplo, el día en el que quiere viajar. Si ese no fuera el caso, el agente conversacional no puede responder directamente al usuario con el precio del vuelo y necesitaría realizarle una serie de preguntas para obtener la información que le falta sobre la fecha concreta del vuelo antes de poder ejecutar la búsqueda de los precios y contestar a la petición del usuario mostrándole los precios de los vuelos.

De esta forma, cualquier agente conversacional que necesite asegurar que puede responder a las peticiones del usuario, aunque estas sean incompletas, y que no solamente conteste a las peticiones para las cuales el usuario proporcione toda la

información en una única petición, **debe implementar un módulo de gestión del diálogo que mantenga el estado de la conversación y que permita múltiples interacciones con el usuario para obtener la información no disponible**. De hecho, si el agente conversacional solo respondiera a las peticiones completas y no mantuviera ninguna interacción con el usuario, se hablaría de un simple sistema de preguntas y respuestas y no de un agente conversacional.

Existen diferentes estrategias para la gestión del diálogo. Por ejemplo, la mayoría de los agentes conversacionales dedicados que utilizan *frames* tienen un módulo de diálogo basado en una **máquina de estados finitos** que define las acciones necesarias para llevar a cabo una tarea concreta:

- ▶ La máquina de estados finitos que rige un agente conversacional dedicado basado en *frames* gestiona el dialogo y modela cómo este debe desarrollarse.
- ▶ Los estados de la máquina de estados finitos se corresponden con las preguntas a realizar al usuario y que son necesarias para obtener los valores de los diferentes huecos en la plantilla.
- ▶ Los arcos de la máquina de estados finitos se corresponden a las acciones que deben realizarse en función de lo que el usuario responda.

Entonces, la máquina de estados finitos controla completamente la conversación con el usuario. El agente conversacional le hace una serie de preguntas al usuario, ignorando (o malinterpretando) cualquier contestación que no sea una respuesta directa a la pregunta y luego pasa a la siguiente pregunta. De hecho, la arquitectura de control para un agente conversacional basado en *frames* también utiliza plantillas para generar el contenido de las nuevas oraciones con las que se debe continuar la conversación. Por lo que va a ser necesario llenar los huecos en las plantillas con los valores relevantes que se han obtenido de las interacciones previas con el usuario.

Ejemplo ilustrativo 5

Gestión del diálogo en un agente conversacional dedicado basado en *frames* utilizando una máquina de estados finitos.

La máquina de estados finitos que rige un agente conversacional dedicado a la reserva de vuelos (similar al que se ha presentado en el ejemplo ilustrativo 3 y 4) se muestra en la Figura 4. Este agente conversacional basado en *frames* gestiona el dialogo y modela cómo se debe desarrollar con base en esta máquina de estados finitos. De hecho, se observan las diferentes preguntas que se van a realizar para obtener la información necesaria para llenar los diferentes huecos del *frame* antes de ejecutar la tarea de reservar el vuelo.

Por ejemplo, en uno de los turnos de palabra el agente conversacional pregunta:

- ▶ «Do you want to go from <FROM> to <TO> on <DATE>? ».

Se observa que esta pregunta es una plantilla y que tiene tres huecos *FROM*, *TO* y *DATE* que deben ser rellenados a partir de las respuestas obtenidas de las preguntas realizadas en turnos anteriores.

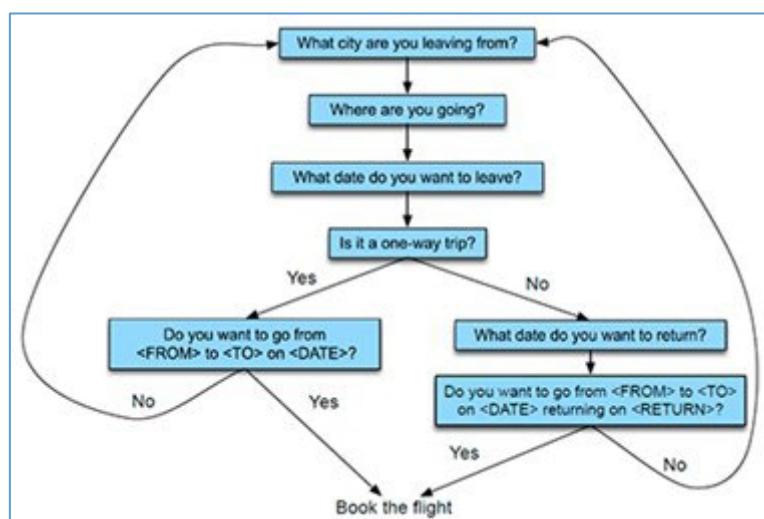


Figura 4. Máquina de estados finitos para modelar la gestión del diálogo de un agente conversacional basado en *frames* para reservar vuelos. Fuente: Jurafsky y Martin, 2009.

La Tabla 2 muestra la asociación entre los huecos y las preguntas a partir de las cuales se va a obtener el valor para llenar esos huecos.

Hueco (slot)	Pregunta
FROM	What city are you leaving from?
TO	Where are you going?
DATE	What date do you want to leave?
RETURN	What date do you want to return?

Tabla 2. Asociación entre los huecos del *frame* y las preguntas que necesita realizar el agente conversacional para determinar el valor de esos huecos. Fuente: elaboración propia.

Una estrategia de gestión del diálogo basada en una máquina de estados finitos en la cual el agente conversacional toma totalmente la iniciativa tiene la ventaja de que el sistema siempre sabe a qué pregunta responde el usuario. Esto significa que el sistema puede preparar el módulo de reconocimiento automático del habla con un modelo de lenguaje adaptado a las respuestas para esta pregunta. Esto también facilita el funcionamiento del módulo de comprensión del lenguaje natural que puede obtener la semántica de una forma más fácil. No obstante, una arquitectura basada en una máquina de estados finitos tan simplista solo se puede aplicar a tareas muy sencillas y no es suficiente para los agentes conversacionales que necesitan algo más de flexibilidad en el diálogo.

Google Assistant, Siri de Apple o Alexa de Amazon utilizan una estrategia de gestión del diálogo más compleja y que está basada en la arquitectura del agente conversacional GUS (Bobrow et al., 1977). Esta estrategia permite al usuario tomar cierta iniciativa y ayuda al agente conversacional a tratar los *frames* de una forma mucho más flexible. El agente conversacional hace preguntas al usuario, rellenando cualquier hueco en la plantilla al que haga referencia la información que el usuario le transmite. Entonces, en cada interacción, el agente conversacional puede llenar varios huecos a la vez, incluso si la respuesta del usuario no se corresponde con la pregunta realizada. El sistema simplemente salta las preguntas asociadas con los huecos que ya han sido llenados previamente.

Los diferentes huecos del *frame* se llenan sin seguir una secuencia concreta y en función de la información proporcionada por el usuario.

Una vez que el agente conversacional tiene la suficiente información, ya puede realizar la acción, por ejemplo, **consultar la base de datos de vuelos**, y finalmente devolver el resultado al usuario.

Los agentes conversacionales basados en *frames* que siguen una estrategia de control flexible como, por ejemplo, muchos de los agentes conversacionales actuales, necesitan cambiar el control del diálogo y saltar entre *frames* dependiendo de la información proporcionada por el usuario. **Debido a esta necesidad de cambiar dinámicamente el control, la arquitectura GUS para la gestión del diálogo se implementa como un sistema de reglas de producción y la información proporcionada por el usuario puede activar varias reglas de producción**, el resultado de lanzarlas permite llenar múltiples huecos en diferentes *frames*.

A diferencia de los agentes conversacionales dedicados basados en *frames*, los agentes conversacionales basados en el diálogo **aplican estrategias** mucho más **avanzadas para la gestión del diálogo**. Por ejemplo, para la gestión del diálogo, algunos de estos agentes conversacionales modernos, que todavía están en el ámbito de la investigación, aplican una estrategia basada en el contexto local y otros una estrategia basada en procesos de decisión de Markov y aprendizaje por refuerzo.

Los agentes conversacionales que se basan en el diálogo e implementan en el módulo de gestión del diálogo una estrategia basada en el contexto local lo que hacen es predecir, para un turno de palabra, cuál es la acción para realizar según el estado del dialogo, es decir, con base en las acciones que han tomado el agente y el usuario en los turnos de palabra anteriores.

Para simplificar, se puede considerar que el estado del diálogo principalmente solo va a depender de la última acción que ha realizado el agente conversacional, la última acción que ha realizado el usuario y del estado actual

del *frame*, es decir, del conjunto de los huecos rellenos y los valores que estos toman.

A partir de un corpus de conversaciones lo suficientemente grande se puede estimar la probabilidad de que el agente conversacional tome una acción dadas las acciones que han tomado el agente conversacional y el usuario en el turno de palabra anterior y el estado actual del *frame*.

Entonces, el módulo de gestión del diálogo del agente conversacional va a seleccionar la acción que maximice la probabilidad para ese turno de palabra.

Esta estrategia de gestión del diálogo que decide qué acción debe realizar el agente conversacional tiene un problema: solo se fundamenta en el pasado del diálogo, ignorando completamente si la acción que toma el agente conversacional es probable que conduzca al resultado esperado o no (por ejemplo, a completar correctamente la reserva de un vuelo). De hecho, en el momento de planificar la acción aún falta mucho para poder saber si el resultado será exitoso. Es por eso por lo que hoy en día se están estudiando **nuevas estrategias para la gestión del diálogo** asentadas en procesos de decisión de Markov y aprendizaje por refuerzo.

Generación del lenguaje natural

El módulo de generación del lenguaje natural tiene como **objetivo elegir los conceptos que se quieren expresar al usuario y, además, planificar cómo expresarlos en palabras**. Entonces, la función de este módulo se puede separar en dos etapas: **qué decir y cómo decirlo**.

Primera etapa

En la primera **etapa** del módulo de generación del lenguaje se realiza una tarea de **planificación del contenido**, es decir, se decide **qué contenido se debe expresar al usuario en cada turno de palabra**. Por ejemplo, si el usuario ha hecho una pregunta,

se presenta la respuesta a esa pregunta o, si ha hecho una propuesta, el agente conversacional puede aceptarla o rechazarla.

Así, la tarea de planificación del contenido está basada en el patrón de los pares adyacentes que define la estructura de las conversaciones y tiene en consideración los actos de habla asociados a cada una de las expresiones en la conversación.

Muchas veces la funcionalidad de planificación del contenido se integra con el módulo de gestión de diálogo y no aparece como un componente separado en el módulo de generación del lenguaje. Este sería el caso de los agentes conversacionales basados en *frames* que se han presentado en los ejemplos ilustrativos 3, 4 y 5. Estos agentes gestionan el diálogo utilizando una máquina de estados finitos de forma que, para cada estado, se obtiene directamente la expresión que el sistema le transmite al usuario en ese turno de palabra. De forma similar, los agentes conversacionales basados en el diálogo que aplican una estrategia según el contexto local también planifican la acción a realizar para un turno de palabra en el módulo de gestión de diálogo.

Segunda etapa

En la segunda etapa del módulo de generación del lenguaje se escogen las estructuras sintácticas y las palabras que se necesitan para expresar el concepto que se quiere transmitir al usuario. Existen diferentes formas para realizar esta tarea dependiendo de la estrategia utilizada para la gestión del diálogo en el módulo de gestión de diálogo o la implementación concreta de primera etapa del módulo de generación del lenguaje.

En el caso de un agente conversacional dedicado basado en *frames*, el módulo de gestión de diálogo ya ha determinado la pregunta que el agente conversacional debe efectuar al usuario en el siguiente turno de palabra. De hecho, esa pregunta que se transmitirá al usuario puede estar completa o tener algunos huecos que se deben

rellenar con los valores obtenidos por el agente conversacional en las interacciones anteriores. Por lo tanto, el módulo de generación del lenguaje natural no tiene que realizar ninguna acción complicada, solo identificar los huecos en la frase y rellenarlos con los valores correctos para ese turno de palabra.

En el caso de un agente conversacional que se basa en el diálogo e implementa una **estrategia de gestión basada en el contexto local**, el módulo de gestión de diálogo ha determinado el acto de habla del siguiente turno de palabra, es decir, la acción que el agente conversacional debe realizar en el siguiente turno de palabra. Por lo tanto, el módulo de generación del lenguaje natural conoce el acto de habla del siguiente turno de palabra y el estado actual del *frame*, es decir, el conjunto de los huecos y los valores que estos toman. Entonces, el módulo de generación del lenguaje natural debe generar el mensaje que el agente conversacional va a transmitir al usuario en el siguiente turno de palabra.

Para generar el mensaje asociado a un acto de habla se utiliza un proceso con dos pasos:

- ▶ Primero, se genera una frase deslexicalizada (una frase con huecos).
- ▶ Luego, se rellenan los huecos con los valores adecuados que se obtienen del estado actual del *frame*.

Se sigue esta estrategia de dos pasos porque **generar una frase deslexicalizada es más fácil que generar la frase final**. Para obtener la frase deslexicalizada lo más normal es **utilizar un N-grama entrenado a partir de las frases de un corpus que están etiquetadas con el acto de habla que se quiere expresar en el siguiente turno de palabra**. Sin embargo, algún trabajo reciente, en lugar de utilizar N-gramas, utiliza modelos neuronales que aprenden cómo obtener la frase resultante a partir del siguiente turno de palabra y del estado actual del *frame* (Wen et al., 2015).

Conversión del texto al habla

El módulo de salida tiene como función la conversión del texto al habla sintetizando en formato de voz el mensaje a transmitir al usuario en el siguiente turno de palabra: Este módulo de conversión del texto al habla obtiene del módulo de generación del lenguaje natural el mensaje en formato de texto y sintetiza la onda de sonido para este mensaje. Los métodos utilizados para la conversión del texto al habla no son objeto de estudio en este curso, aunque describiremos brevemente algunas ideas de cómo se realiza la síntesis de la voz.

El módulo de conversión del texto al habla asigna anotaciones sobre el acento y la entonación de cada una de las palabras que componen el mensaje (Reiter y Dale, 2000) y luego las utiliza para sintetizar una onda de sonido. Los métodos de síntesis que más se han utilizado hasta la fecha son los basados en concatenación. En estos métodos se graban unas muestras del habla y se guardan en una base de datos, después se combinan para crear nuevas palabras y frases. Sin embargo, hay otros métodos más modernos fundamentados en hacer las transiciones más suaves o en la articulación.

10.6. Diseño de *chatbots*

Los *chatbots* son aquellos agentes conversacionales que permiten reproducir conversaciones extensas y no estructuradas que se han utilizado en el ámbito del entretenimiento o para propósitos prácticos como probar teorías de la práctica psicológica, se dividen en dos clases en función de su implementación: sistemas basados en reglas y sistemas basados en corpus.

Chatbots basados en reglas

Entre estos se incluye uno de los primeros agentes conversacionales, **ELIZA** (Weizenbaum, 1966). ELIZA fue diseñado para simular a un psiquiatra cuyo método involucra hacer reflexionar al paciente devolviéndole sus propias declaraciones. Por ejemplo, si un paciente le dice al psiquiatra: «Fui a dar un paseo en bote», el psiquiatra le va a responder: «Hábleme de los botes». En este caso no se asumirá que el psiquiatra no sabe qué es un bote, se asume que la respuesta tiene un objetivo puramente conversacional.

Entonces, un *chatbot* basado en reglas como ELIZA implementa este tipo de conversación poco común en la cual el *chatbot* no conoce casi nada del mundo real y solo se dedica a reformular las palabras del usuario.

Un *chatbot* basado en reglas como ELIZA **utiliza reglas patrón→transformación**. Cada regla describe la transformación que se debe aplicar a la declaración hecha por el usuario para obtener la respuesta que le va a proporcionar el *chatbot*, siempre y cuando la declaración del usuario cumpla un cierto patrón. Cada regla está asociada a una **palabra clave** que puede ocurrir en la frase, además, a las palabras clave se les asigna un rango, las palabras clave más específicas tienen un rango más alto y las más generales tienen un rango más bajo.

En cada turno de palabra, el *chatbot* responde con la transformación del patrón asociado a la palabra clave que tiene un rango más alto de todas las palabras contenidas en la declaración hecha por el usuario. En el caso de que no hubiera ninguna palabra en la declaración del usuario que coincidiera con las palabras claves asociadas a las reglas, el *chatbot* daría una respuesta evasiva como «esto es muy interesante» o «por favor, sigue». También puede utilizar algún truco para intentar evitar estas respuestas evasivas, por ejemplo, cuando le llega una frase en la que la palabra clave más probable es el determinante «mi», se aplica una transformación con un patrón similar a «Antes has dicho que tu ...» y el resultado se guarda en

memoria para utilizarlo en el próximo turno de palabra en el que no coincide ninguna palabra clave. La Figura 5 muestra la lógica del algoritmo implementado por ELIZA.

```
function ELIZA GENERATOR(user sentence) returns response
    Find the word w in sentence that has the highest keyword rank
        if w exists
            Choose the highest ranked rule r for w that matches sentence
            response  $\leftarrow$  Apply the transform in r to sentence
            if w = 'my'
                future  $\leftarrow$  Apply a transformation from the 'memory' rule list to sentence
                Push future onto memory stack
            else (no keyword applies)
                either
                    response  $\leftarrow$  Apply the transform for the NONE keyword to sentence
                or
                    response  $\leftarrow$  Pop the top response from the memory stack
            return(response)
```

Figura 5. Lógica del algoritmo implementado por ELIZA, *chatbot* basado en reglas. Fuente: Jurafsky y Martin, 2009.

El modelo de reglas patrón→transformación de ELIZA aún se utiliza hoy en día. Por ejemplo, un *chatbot* moderno como ALICE se basa en una versión actualizada del *chatbot* basado en reglas ELIZA.

Ejemplo ilustrativo 5

ELIZA, uno de los primeros agentes conversacionales, un *chatbot* basado en reglas que simula a un psiquiatra cuyo método involucra hacer reflexionar al paciente contestándole con sus propias declaraciones.

A continuación, se presenta un fragmento de la conversación más famosa de ELIZA recogida en el artículo de Weizenbaum (1966). Las frases en minúscula son las que pronunció el paciente y las frases en mayúscula las contestaciones generadas por parte del *chatbot* que simulaba el psiquiatra:

Men are all alike.

IN WHAT WAY?

They're always bugging us about something or other.

CAN YOU THINK OF A SPECIFIC EXAMPLE?

Well my boyfriend made me come here.

YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.

I AM SORRY TO HEAR YOU ARE DEPRESSED

...

WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU?

My father

YOUR FATHER

You are like my father in some ways.

WHAT RESEMBLANCE DO YOU SEE?

You are not very aggressive but I think you don't want me to notice that.

WHAT MAKES YOU THINK I AM NOT AGGRESSIVE?

You don't argue with me.

WHY DO YOU THINK I DON'T ARGUE WITH YOU?

You are afraid of me.

DOES IT PLEASE YOU TO BELIEVE I'M AFRAID OF YOU?

My father is afraid of everybody.

WHAT ELSE COMES TO MIND WHEN YOU THINK OF YOUR FATHER?

Bullies.

*DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR BOYFRIEND
MADE YOU COME?*

Algunas de las reglas patrón→transformación que implementa ELIZA son las siguientes:

(0 YOU 0 ME) -> (WHAT MAKES YOU THINK I 3 YOU)

(I *) -> (You say you 2)

En la primera regla, cuando se da el patrón 0 YOU 0 ME, es decir, algo seguido de la palabra YOU, otro algo y luego la palabra ME, entonces se transforma en la frase en WHAT MAKES YOU THINK I 3 YOU, donde el número tres debe substituirse por el valor del segundo cero en el patrón. Por lo tanto, a partir de la primera regla, la oración pronunciada por el usuario «You hate me» iría seguida de la oración obtenida en la transformación «WHAT MAKES YOU THINK I HATE YOU».

Si el usuario pronuncia la siguiente frase: «I know everybody laughed at me», ELIZA le contestaría «YOU SAY YOU KNOW EVERYBODY LAUGHED AT YOU» después de aplicar la transformación recogida en la segunda regla.

Chatbots basados en corpus

Los *chatbots* basados en corpus **aplican técnicas de minería de datos** a grandes conjuntos de conversaciones entre humanos **con el fin de extraer las posibles respuestas del *chatbot* al usuario** y para no tener que generar reglas a mano.

Los principales corpus que permiten aprender de las conversaciones están basados en conversaciones en plataformas de chat, en Twitter o en diálogos de películas (Serban et al., 2017). De hecho, a veces estos *chatbots* también pueden minar conversaciones entre un humano y una máquina e incluso extraer oraciones de un texto que no sea un diálogo.

Al igual que los basados en reglas, **la mayoría de *chatbots* basados en corpus tienden a utilizar poca información del contexto de la conversación** para generar una respuesta y suelen generar la respuesta basándose solamente en el turno de palabra inmediatamente anterior.

Entonces, **los *chatbots* basados en corpus tienen cierta similitud con los sistemas de búsqueda de respuestas (Question Answering) que permiten la recuperación de información basada en el lenguaje natural.**

Hay **dos tipos de *chatbots* basados en corpus:**

- ▶ **Basados en recuperación de información.**
- ▶ **Los *chatbots* secuencia a secuencia.**

Por ejemplo, Cleverbot, que permite que un usuario pueda hablar con él por puro divertimento, pertenece al grupo de los *chatbots* basados en recuperación de información.

Chatbots basados en recuperación de información

Este tipo de *chatbots* funcionan utilizando un método que les permite obtener una respuesta proporcionada por un humano en una conversación previa y utilizarla para responder al turno de palabra actual.

Así, usan una respuesta que aparece en el corpus y cuyo turno anterior es lo más parecido posible a la interacción que justo acaba de efectuar el usuario con el *chatbot* para el turno de respuesta de esta interacción. La idea es que se debe buscar un turno que se parezca lo más posible al turno del usuario y devolver la respuesta humana a ese turno (Jafarpour, Burges y Ritter, 2009; Leuski y Traum, 2011).

Otra opción sería que el *chatbot* basado en recuperación de información, en lugar de devolver la respuesta al turno más similar, devolviera directamente el turno más similar. La idea en este caso es buscar las coincidencias entre la interacción del usuario y los turnos en el corpus, ya que una buena respuesta a menudo compartirá palabras o semántica con el turno anterior.

Aunque devolver la respuesta al turno más similar parece ser un algoritmo más intuitivo, en la práctica parece que devolver el turno más similar funciona mejor (Ritter, Cherry y Dolan, 2011; Wang et al. 2013).

Para medir la similitud entre turnos se puede utilizar cualquier medida de similitud, por ejemplo, la similitud del coseno computada sobre las palabras (usando una medida de *term frequency-inverse document frequency*). De hecho, se puede extender el algoritmo para que incluya más características, además de las palabras en el turno actual, por ejemplo, las palabras en turnos anteriores o información sobre el usuario.

Debido a la importancia que tiene el corpus para el correcto funcionamiento de los *chatbots* basados en recuperación de información, las respuestas que va generando el usuario durante su interacción con el *chatbot* también se pueden usar para seguir entrenando este.

Chatbots secuencia a secuencia

Estos *chatbots* están basados en el paradigma de la traducción automática y aplican técnicas de aprendizaje automático supervisado. Una forma alternativa de utilizar un corpus para generar un diálogo es pensar en la generación de respuestas como la tarea de transducción desde el turno del usuario hasta el siguiente turno del sistema, es decir, un *chatbot* que implemente esta idea sería básicamente una versión de ELIZA, pero basado en aprendizaje automático.

En la primera implementación de esta idea se utilizaron técnicas de traducción automática basada en frases (Ritter et al., 2011) para traducir un turno de usuario en una respuesta del sistema. Sin embargo, rápidamente quedó claro que la tarea de generación de respuestas era demasiado diferente de la traducción automática. En esta, las palabras o frases en las oraciones de origen y de destino se alinean entre sí. No obstante, en una conversación, una declaración del usuario puede no compartir palabras o frases con una respuesta coherente. Entonces, aproximadamente de forma simultánea se empezaron a modelar los transductores para la generación de respuestas utilizando modelos secuencia a secuencia (seq2seq) (Shang, Lu y Li, 2015; Vinyals y Le, 2015; Sordoni et al. 2015).

El modelo seq2seq tiene una serie de problemas, por ejemplo, es incapaz de modelar el contexto anterior a la conversación y genera respuestas que no tienen por qué ser coherentes entre los diferentes turnos de palabra. Entonces, se han realizado algunas modificaciones al modelo seq2seq básico para que pueda funcionar en *chatbots* secuencia a secuencia. Por ejemplo, se ha introducido información sobre los turnos de palabra previos utilizando un modelo jerárquico (Lowe et al., 2017) y se han implementado métodos de aprendizaje por refuerzo para aprender a escoger las respuestas que hacen que la conversación sea más natural (Li et al., 2017).

10.7. Referencias bibliográficas

Austin, J. L. (1962). *How to Do Things with Words*. Harvard University Press.

Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H. y Winograd, T. (1977). GUS, A frame driven dialog system. *Artificial Intelligence*, 8, 155–173.
Clark, H. H. (1996). Cambridge University Press.

Clark, H. H. y Schaefer, E. F. (1989). Contributing to discourse. *Cognitive Science*, 13, 259-294.

Forbes-Riley, K. y Litman, D. J. (2011). Benefits and challenges of real-time uncertainty detection and adaptation in a spoken dialogue computer tutor. *Speech Communication*, 53(9), 1115–1136.

Jafarpour, S., Burges, C. y Ritter, A. (2009). Filter, rank, and transfer the knowledge: Learning to chat. <https://aritter.github.io/chat.pdf>

Jurafsky, D. y Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition and Computational Linguistics*. Prentice-Hall.

Leuski, A. y Traum, D. (2011). NPCEditor: Creating virtual human dialogue using information retrieval techniques. *AI Magazine*, 32(2), 42-56.

Levinson, S. C. (1983). Conversational Analysis. En *Pragmatics*. Cambridge University Press.

Li, J., Monroe, W., Shi, T., Ritter, A. y Jurafsky, D. (2017). Adversarial learning for neural dialogue generation. En *EMNLP* (pp. 2157-2169). Association for Computational Linguistics.

Lowe, R. T., Pow, N., Serban, I. V., Charlin, L., Liu, C. W. y Pineau, J. (2017). Training end-to-end dialogue systems with the ubuntu dialogue corpus. *Dialogue & Discourse*, 8(1), 31–65.

Pieraccini, R. y Levin, E. (1992). Stochastic representation of semantic structure for speech understanding. *Speech Communication*, 11(2-3), 283-288.

RAE. (s. f.). Diálogo. En Diccionario de la lengua española (actualización de la 23^a ed.).
<https://dle.rae.es/?id=DetWqMJ>

Reiter, E. y Dale, R. (2000). *Building Natural Language Generation Systems*. Cambridge University Press.

Ritter, A., Cherry, C. y Dolan, B. (2011). Data-driven response generation in social media. In EMNLP-11, pp. 583-593.

Sacks, H., Schegloff, E. A. y Jefferson, G. (1974). A simplest systematics for the organization of turn-taking for conversation. *Language*, 50(4), 696–735.

Schegloff, E. A. (1968). *Sequencing in conversational openings*. *American Anthropologist*, 70, 1075–1095.

Serban, I. V., Lowe, R. T., Charlin, L. y Pineau, J. (2017). A survey of available corpora for building data-driven dialogue systems. arXiv:1512.05742.

Shang, L., Lu, Z. y Li, H. (2015). Neural responding machine for short-text conversation. En *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (pp. 1577-1586). Beijing: Association for Computational Linguistic.

Sordoni, A., Galley, M., Auli, M., Brockett, C., Ji, Y., Mitchell, M., Nie, J. Y., Gao, J. y Dolan, B. (2015). A neural network approach to context-sensitive generation of conversational responses.

<http://rali.iro.umontreal.ca/rali/sites/default/files/publis/1506.06714.pdf>

Stalnaker, R. C. (1978). Assertion. En P. Cole (Ed.), *Pragmatics: Syntax and Semantics* (vol. 9, pp. 315-332). Academic Press.

Vinyals, O. y Le, Q. (2015). A neural conversational model. En *Proceedings of the International Conference on Machine Learning*.
<https://arxiv.org/pdf/1506.05869.pdf>

Wang, H., Lu, Z., Li, H. y Chen, E. (2013). A dataset for research on short-text conversations. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (pp. 935-945). Association for Computational Linguistics.

Weizenbaum, J. (1966). ELIZA—A computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1), 36-45.

Wen, T. H., Gasic, M., Kim, D., Mrksic, N., Su, P. H., Vandyke, D. y Young, S. J. (2015). Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. En *SIGDIAL 2015 Meeting on Discourse and Dialogue* (pp. 275-284). Association for Computational Linguistics.

Yankelovich, N., Levow, G. A. y Marx, M. (1995). Designing SpeechActs: issues in speech user interfaces. En *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '95) (pp. 369-376). ACM Press/Addison-Wesley Publishing Co.

A fondo

Selección de las transcripciones de los finalistas del Premio Lobner 2017

Martin, A. O. (2017). AISB Loebner Prize 2017 Finalist Selection Transcripts.
http://www.aomartin.co.uk/uploads/loebner_2017_finalist_selection_transcripts.pdf

El documento presenta una selección de las transcripciones de las conversaciones llevadas a cabo por el juez y los diferentes *chatbots* durante los test de Turing que permitieron decidir el ganador del Premio Lobner 2017.

Como superar el test de Turing (ganar el Premio Loebner)

Moloney, C. (24 de septiembre de 2017). How to win a Turing Test (the Loebner prize).
<https://chatbotsmagazine.com/how-to-win-a-turing-test-the-loebner-prize-3ac2752250f1>

El artículo recoge la experiencia de uno de los jueces que formó parte del jurado que falló el Premio Loebner 2017. Además, en el artículo se expone una idea muy interesante de este juez: el hecho de que se deberían programar los *chatbots* para que cometieran errores como pasa en el caso de los humanos y así engañar a los jueces.

Google Assistant realizando una llamada durante el Google I/O '18

Presentación de apertura. (8 de mayo de 2018). Youtube. <https://youtu.be/ogfYd705cRs>

El vídeo (desde el minuto 35:00 al minuto 40:24) muestra la presentación de la nueva funcionalidad del Google Assistant que permite realizar llamadas de forma automática. La demostración de esta funcionalidad la realizó el CEO de Google, Sundar Pichai, durante el congreso de desarrolladores Google I/O que tuvo lugar en mayo de 2018.

Discurso y diálogo

Sanchís, E. (2016). Discurso y diálogo. En Á. L. Gonzalo (Coords.), *Tecnologías del lenguaje en España: comunicación inteligente entre personas y máquinas*. Barcelona: Fundación Telefónica-Ariel. https://www.fundaciontelefonica.com/arte_cultura/publicaciones-listado/pagina-item-publicaciones/itempubli/565/

Este capítulo, que forma parte del estudio impulsado por la Fundación Telefónica y el Instituto para la Promoción de las Tecnologías de la Lengua, introduce las complejidades lingüísticas, psicológicas y cognitivas de los procesos de discurso y diálogo, y concluye que:

«Uno de los retos más importantes actualmente, además de aumentar la robustez en reconocimiento y comprensión, es la posibilidad de diseñar sistemas que aprendan de las interacciones con usuarios reales y se adapten automáticamente a nuevas situaciones. Debe tenerse en cuenta que la usabilidad de estos sistemas sólo se popularizará cuando sus prestaciones sean muy altas, ya que de otra forma el posible usuario se frustra y escoge otro tipo de interacción».

Test

1. Indica las afirmaciones correctas sobre los agentes conversacionales:

 - A. Son programas que conversan con las personas a través del lenguaje natural.
 - B. Interactúan con el humano o a través de la voz o a través de texto, pero nunca a través de los dos.
 - C. Son capaces de continuar la interacción con el usuario contestando a preguntas dependientes de las anteriores.
 - D. También se llaman sistemas de diálogo.
2. Indica las afirmaciones correctas sobre las características de las conversaciones entre humanos que influyen en los agentes conversacionales:

 - A. Todos los agentes conversacionales son capaces de tratar con múltiples de actos de habla, es decir, que son capaces de lanzar una pregunta, dar una orden, hacer una propuesta, rechazar una sugerencia o estar de acuerdo con una expresión presentada por su interlocutor.
 - B. La gestión de los turnos de palabra en agentes conversacionales se puede modelar con una serie de reglas que se evalúan en los momentos en que la estructura del lenguaje permite intercambiar el interlocutor.
 - C. Para establecer puntos de coincidencia con el interlocutor, el agente conversacional debe dejar claro explícitamente que ha entendido el significado y la intención de la expresión lanzada por el humano.
 - D. En las conversaciones se da un patrón de pares adyacentes, por ejemplo, si en un turno de palabra se lanza una pregunta (primera parte del par), en el siguiente turno se espera que se dé una respuesta (segunda parte del par). Sin embargo, puede darse el caso que la segunda parte del par no aparezca directamente justo después de la primera.

3. Indica las afirmaciones correctas sobre los diferentes tipos agentes conversacionales:

- A. Los agentes conversacionales dedicados permiten entablar conversaciones extensas con múltiples interacciones.
- B. Los *chatbots* son aquellos agentes conversacionales que permiten entablar conversaciones sobre cualquier tema o materia.
- C. Los agentes conversacionales disponibles en las páginas web que proporcionan ayuda a los clientes para resolver dudas son agentes conversacionales dedicados.
- D. Cortana de Windows es un *chatbot*.

4. Indica las afirmaciones correctas sobre la estructura de los agentes conversacionales:

- A. Los agentes conversacionales, independientemente de las modalidades que utilicen para interactuar con el usuario, siguen una estructura básica común.
- B. La mayoría de los agentes conversacionales implementan el mismo método para gestionar el diálogo.
- C. Los agentes conversacionales más sencillos pueden integrar las funcionalidades de comprensión del lenguaje natural, de gestión del diálogo y de generación del lenguaje natural en un único módulo.
- D. Un agente conversacional basado en voz se compone de un módulo de entrada para el reconocimiento automático de la voz, un módulo de comprensión del lenguaje natural, un módulo de gestión del diálogo, un módulo de generación del lenguaje natural y un módulo de una salida de conversión de texto al habla.

5. Indica las afirmaciones correctas sobre el módulo de comprensión del lenguaje natural en un agente conversacional:

- A. Es el módulo de entrada de la arquitectura del agente conversacional.
- B. Para representar la semántica de las frases es muy común que utilice *frames* y pares atributo-valor.
- C. Aparte de determinar los huecos y sus posibles valores expresados en la frase, tiene que identificar los aspectos propios de un diálogo como, por ejemplo, los puntos de coincidencia o los actos del habla.
- D. Tiene como objetivo extraer la semántica de las frases en un turno de palabra.

6. Indica las afirmaciones correctas sobre el módulo de gestión del diálogo en un agente conversacional:

- A. Es el elemento principal del agente conversacional.
- B. Analiza la representación semántica de la frase extraída por el módulo de comprensión del lenguaje natural.
- C. Es imprescindible mantener el estado del diálogo y el flujo de la conversación para la gestión del diálogo.
- D. Se puede implementar como una máquina de estados finitos o como un proceso de decisión de Márkov.

- 7.** Indica las afirmaciones correctas sobre el módulo de generación del lenguaje natural en un agente conversacional:
- A. Tiene como objetivo elegir los conceptos que se quieren expresar al usuario y además planificar cómo expresar estos conceptos en palabras.
 - B. Realiza una tarea de planificación del contenido totalmente independiente de la tarea realizada por el módulo de gestión de diálogo.
 - C. Escoge las estructuras sintácticas y las palabras que se necesitan para expresar el concepto.
 - D. Es el módulo de salida de la arquitectura del agente conversacional.
- 8.** Indica las afirmaciones correctas sobre los *chatbots* basados en reglas:
- A. Las reglas conforman una gramática libre de contexto y están anotadas semánticamente.
 - B. Cada regla está asociada a una palabra clave en la frase.
 - C. Implementan un tipo de conversación poco común en la cual el *chatbot* no conoce casi nada del mundo real.
 - D. ELIZA, uno de los primeros agentes conversacionales, implementa este tipo de estrategia.
- 9.** Indica las afirmaciones correctas sobre los *chatbots* basados en corpus:
- A. Aplican técnicas de minería de datos a un conjunto de conversaciones entre humanos para extraer las posibles respuestas del *chatbot* al usuario.
 - B. Las respuestas del *chatbot* se pueden extraer de un texto que no sea un diálogo.
 - C. Utilizan poca información del contexto de la conversación para generar una respuesta.
 - D. Suelen generar la respuesta basándose solamente en el turno de palabra inmediatamente anterior.

- 10.** Indica las afirmaciones correctas sobre los tipos de *chatbot* basados en corpus:
- A. Los *chatbot* basados en la recuperación de información usan como turno de respuesta a una consulta una respuesta que aparezca en el corpus y cuyo turno anterior sea lo más parecido posible a la consulta.
 - B. Los *chatbot* basados en el aprendizaje supervisado de la transducción de secuencias se implementan utilizando transductores de autómatas finitos.
 - C. Los *chatbot* secuencia a secuencia se basan en la idea de que generar una respuesta es una tarea de transducción del turno de usuario al turno del sistema, es decir, traducir el turno de usuario en una respuesta del sistema.
 - D. Los *chatbot* secuencia a secuencia utilizan métodos de aprendizaje por refuerzo para aprender a escoger las respuestas que hacen que la conversación sea más natural.