Tema 3

Procesamiento de lenguaje natural y sus aplicaciones. Potencial y limitaciones.

En esta unidad veremos:

3.1 Procesamiento de lenguaje natural.
3.2 Potencial de las técnicas existentes de procesamiento de lenguaje. Limitaciones.
3.3 Formación del investigador en PLN.
3.4 Elaboración de un sistema de procesamiento de lenguaje orientado a una tarea específica.

Índice

3.1 Procesamiento de lenguaje natural

El procesamiento del lenguaje natural es un campo muy amplio, que abarca diversas áreas, y relaciona varios campos de la técnica y la lingüística. Siguiendo la línea del test de Turing, en 1954 se llevó a cabo el **test de Georgetown** que supuso la traducción automática de unas 60 oraciones del ruso al inglés de manera exitosa.
(Figura 3.1).

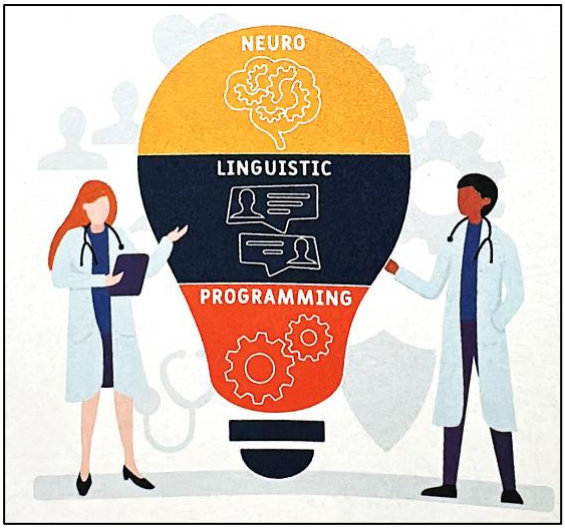


Figura 3.1 Esquema del test de Georgetown.

El **procesamiento del lenguaje natural** estudia las relaciones del lenguaje entre los seres humanos y las máquinas.

El Test de Georgetown es una técnica utilizada en lingüística computacional y procesamiento del lenguaje natural para evaluar la capacidad de traducción automática de una máquina. Fue desarrollado en 1954 durante un experimento conjunto entre la Universidad de Georgetown y la IBM Corporation. Este test marcó un hito al demostrar que las máquinas podrían realizar traducciones entre idiomas, específicamente del ruso al inglés.

Características del Test de Georgetown:

Traducción limitada: Utilizaba un vocabulario restringido de unas 250 palabras y un conjunto básico de 6 reglas gramaticales.

Objetivo experimental: El propósito no era una aplicación práctica inmediata, sino demostrar que la traducción automática era viable.

Impacto: El éxito del test generó un gran interés en el desarrollo de tecnologías de traducción automática durante la década de 1950 y 1960.

Aunque el test fue un éxito en su contexto, la traducción automática ha evolucionado enormemente desde entonces gracias a avances como las redes neuronales y el aprendizaje profundo, permitiendo sistemas como Google Translate o DeepL.

Tras ello hubo un parón en el desarrollo de esta disciplina, debido a limitaciones del hardware, que perduraría hasta la década de 1980.

La **ley de Moore** indica que cada 2 años, desde 1970, se duplica el número de transistores en un microprocesador (Figura 3.2), lo que directamente expresa un aumento en términos de potencia de computación. Ese incremento de potencia, al igual que en otras disciplinas de la inteligencia artificial, como la visión artificial, ha sido lo que ha posibilitado el estado alcanzado actualmente por la técnica.

El procesamiento del lenguaje natural implica necesariamente de la unión de dos áreas: el área asociada a la máquina y el área relacionada con el ser humano. Esta última comporta la forma actual de comunicación entre seres humanos, ya que, si el objetivo es tratar de que una máquina se comunique con un ser humano, y viceversa, es necesario el estudio formal de la manera en la que se comunica una persona, y es precisamente en este punto donde entra en juego la lingüística como disciplina de estudio.

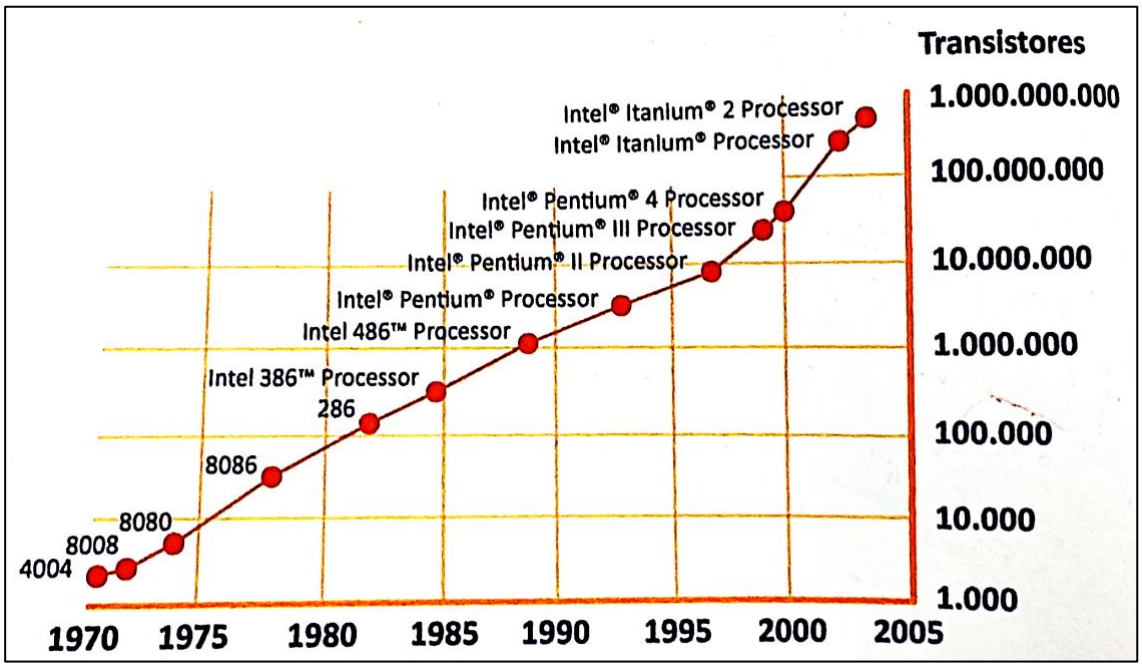


Figura 3.2 Ley de Moore.

Para recordar…

A largo de la historia de la humanidad ha habido distintos tipos de teorías lingüísticas comunes a varios idiomas, siendo muy común la de Noam Chomsky (por ejemplo, la **gramática de constituyentes/transformacional**).

Noam Chomsky es un lingüista, filósofo y activista político conocido principalmente por sus contribuciones revolucionarias a la lingüística teórica. Su teoría transformó la forma en que entendemos el lenguaje humano, destacando la estructura innata de la capacidad lingüística y la gramática universal. A continuación, se resumen los conceptos clave de su teoría lingüística:

Chomsky propuso que los humanos tienen una capacidad innata para adquirir lenguaje, que está gobernada por una **gramática universal**. Según Chomsky, todos los idiomas comparten una estructura subyacente común, conocida como **gramática universal**.

La **gramática generativa** busca describir explícitamente las reglas y estructuras que permiten a una persona generar y entender oraciones gramaticales en su lengua. Introdujo las **transformaciones**, procesos que convierten estructuras básicas (como las oraciones simples) en estructuras más complejas.

Esta teoría sugiere que los humanos nacen con un conjunto de principios gramaticales innatos, lo que explica por qué los niños pueden aprender cualquier idioma al que estén expuestos. Una característica esencial del lenguaje humano, según Chomsky, es su creatividad: la capacidad de producir y comprender oraciones completamente nuevas que nunca antes hemos escuchado.

Esta teoría ha influido en campos como la inteligencia artificial, psicología cognitiva, filosofía del lenguaje y la educación y aunque ha sido criticada y revisada, sigue siendo una base fundamental para el estudio moderno del lenguaje.

Sin embargo, la teoría de Chomsky, **la gramática de constituyentes/transformacional** describe cómo las oraciones están estructuradas en componentes jerárquicos (constituyentes) y cómo pueden ser transformadas para generar diferentes estructuras superficiales manteniendo el mismo significado subyacen.

Esta gramática de **constituyentes/transformacional** en sus fundamentos teóricos no emplea predominantemente **corpus**, que es la clave en la que se basa el aprendizaje máquina para el procesamiento del lenguaje.

Por tanto, la unión de una falta de potencia de cálculo y del tipo de lingüística predominante hasta 1980 marcó el lento desarrollo del procesamiento del lenguaje natural.

Un **corpus** es un conjunto de palabras (un texto) de una lengua y se emplea para formar un diccionario, pero no en el sentido de documento donde se explica o definen palabras, sino como **concepto de conjunto de palabras de una lengua**. En el ámbito de la inteligencia artificial (IA), un **corpus** es un conjunto estructurado de datos o textos que se utiliza para entrenar, evaluar o probar modelos de aprendizaje automático, especialmente en tareas relacionadas con el **procesamiento del lenguaje natural (PLN)**.

Ejemplo 1

Si una persona española tuviera que pensar qué corpus emplear para el entrenamiento de un procesador de lenguaje natural en español, es fácil que centrara en las grandes obras literarias de nuestra lengua, como por ejemplo El Quijote, o un texto difundido en nuestra lengua, de amplia proyección, como puede ser La Biblia.

Lo mismo ocurre en otros idiomas, y en inglés en concreto, existe una base de datos que contiene un conjunto de corpus en esta lengua, que fue realizado en 2014 y recibe el nombre de Gutemberg.

Para saber más…

En el siguiente link se puede ampliar información acerca de la base de datos Gutemberg: https://github.com/pgcorpus/gutenberg

En el link que aparece a continuación figura uno de los muchos corpus que pueden encontrarse para español: https://github.com/roquegv/spanishNLPModelCorpus

Por tanto, a la hora de llevar a cabo avances en el campo del procesamiento del lenguaje natural, **es necesario disponer de un set de datos (o corpus) específico para cada lengua** y, por consiguiente, existirán **modelos específicos para cada idioma**; todo ello determina el papel del lingüista en un proyecto de inteligencia artificial.

Tal como ya se ha estudiado, para que una máquina y un ser humano se comuniquen es preciso entender y estudiar la parte de la máquina y la parte del ser humano. Se iniciará este estudio comenzando por esta última, **definiendo la lingüística y sus áreas principales.**

Atendiendo a lo expuesto por D. Jurafsky en su texto *«Speech and languaje processing»*, en el estudio de la lingüística de una lengua no solo es necesario llevar a cabo el estudio de las palabras y expresiones regulares del mismo, sino que **es preciso estudiar las relaciones generadas por estos cuatro campos: morfología, sintaxis, semántica y pragmática.**

Un lingüista en un proyecto de inteligencia artificial aporta el conocimiento y el enfoque para llevar a cabo exitosamente la programación de las complejas estructuras (variables en virtud de los detalles de origen y evolución de cada lengua) entre los diferentes caracteres para formar una palabra, y de las diversas palabras para crear oraciones. **Una clave de esta labor es el etiquetado**, por el cual a una palabra, grafía, sintagma o estructura determinada se le asigna una etiqueta que lo clasifica.

Ejemplo 2

Perro es la combinación de las grafías «p»• «e» • «r» -«r»• «o», que a su vez desde un punto semántico representa el concepto de un animal, y cuya vocal «-» final denota el género de la palabra, que hace referencia al sexo del animal.

Dejando de lado conceptos técnicos más específicos, como el uso de N-gramas, expresiones regulares para la búsqueda de cadenas, transductores de estado finito, o modelos como el de cadenas ocultas de Markov, para el experto en inteligencia artificial (IA) es preciso tener un mínimo entendimiento de la parte realizada por el lingüista y, por tanto, conocer la esencia de estos campos (Figura 3.3):

Gramática: que incluye la morfología, la sintaxis y, para algunos autores, también la fonología.

Sintaxis: acorde al texto anteriormente citado de Jurafsky, estudia las reglas y principios que gobiernan la combinación de los constituyentes sintácticos. Analiza la formación de los sintagmas y las oraciones gramaticales. Mediante la sintaxis, se conocen las formas en que se combinan las palabras, así como las relaciones sintagmáticas y paradigmáticas existentes entre ellas. Como los sintagmas pueden unirse, para formar un grupo sintagmático, la sintaxis también establece la manera en la que llevar a cabo el proceso de unificación. En el etiquetado sintáctico se adjudica un POS (Part of Speech); por ejemplo, en español a la palabra mostrar se le asignaría la etiqueta POS de verbo, mientras que en la oración «el niño llora» se etiquetaría lo siguiente:

- El → Determinante.
- Niño → Nombre.
- El niño → Grupo sintagmático nominal.
- Llora → Verbo (y grupo sintagmático predicativo).

Por supuesto, dentro de una etiqueta pueden existir subcategorías, por ejemplo, «niño» es un sustantivo (nombre) de tipo «común» y «el» es un determinante de tipo «artículo».

Morfología: según Jurafsky (un investigador destacado en el campo de la lingüística computacional y el procesamiento del lenguaje natural), estudia las reglas que rigen la flexión, la composición y la derivación de las palabras. **Durante el etiquetado morfológico se determina la forma, clase o categoría gramatical de una palabra. El género de las palabras (masculino y femenino), el número de los nombres (singular o plural), o la persona de las formas conjugadas (primera, segunda o tercera) son ejemplos de este campo.**

Semántica: siguiendo el texto, estudia **significado** de las expresiones lingüísticas, es decir, las realidades que representan las grafías.

Pragmática: se centra en **el análisis de la relación del lenguaje con los usuarios y las circunstancias** de la comunicación o contexto. El contexto debe entenderse como situación, ya que puede incluir cualquier aspecto extra-lingüístico: la situación comunicativa, un conocimiento popular compartido por los hablantes, relaciones personales, y otros muchos.

Para saber más…
Existen herramientas web, como la que figura en el siguiente link, que llevan a cabo el etiquetado y la unificación: https://sintaxis.org/analizador/solucion/

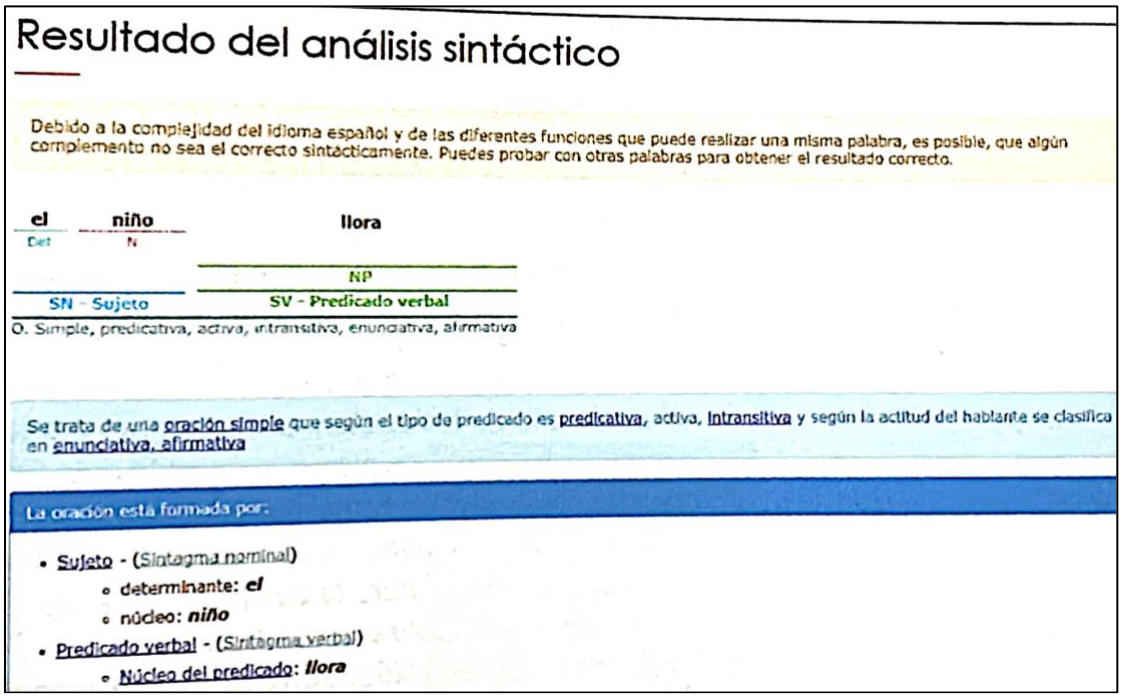


Figura 3.3 Sintaxis: etiquetado y unificación.

Para saber más…

El siguiente enlace del Grupo de Estructuras de Datos y Lingüística Computacional del Departamento de Informática y Sistemas de la Universidad de Las Palmas de Gran Canaria muestra un etiquetador para la lengua española, así como herramientas relacionadas, como un lematizador o un flexionador:

http://www.gedlc.ulpgc.es/investigacion/scogeme02/lematiza.htm

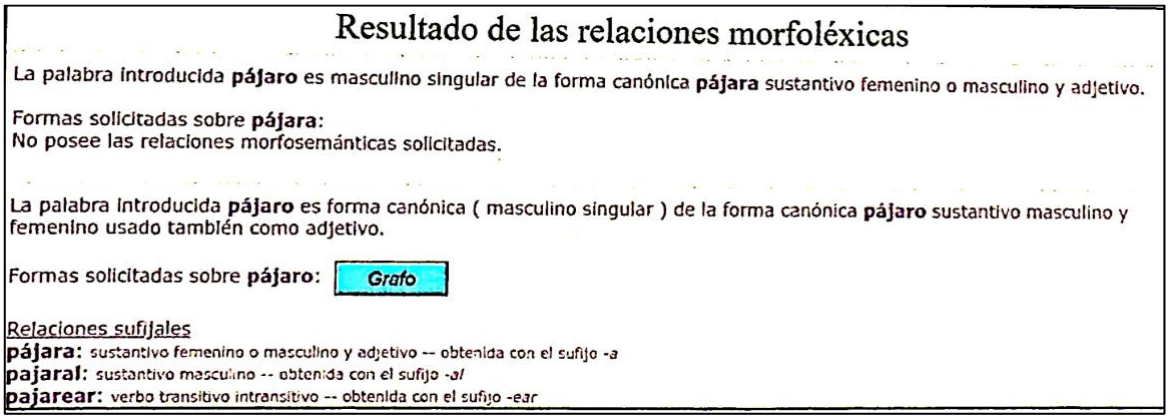


Figura 3.4 Resultado de las relaciones morfoléxicas.

Para recordar…

El conocimiento inmersivo y exhaustivo de estos términos sus relaciones, y el etiquetado es labor del lingüista, y se lo debe proporcionar al experto en lA para una correcta modelización de un sistema de procesamiento del lenguaje natural, en un idioma concreto.

3.2 Potencial de las técnicas existentes de procesamiento de lenguaje. Limitaciones.

3.2.1 Potencial del procesamiento del lenguaje natural

Las aplicaciones del procesamiento del lenguaje natural son muchas: chatbots.

- Para el desarrollo de los asistentes de compra o sistemas conversacionales.
- El análisis del sentimiento y de la opinión que permite detectar por las palabras empleadas sesgos y opiniones sobre un tema, especialmente en publicaciones de redes sociales, o en encuestas determinadas.
- Clasificación automática de documentos.
- Búsqueda de similitudes en textos para la búsqueda de plagios, detección de entidades (NER) que permiten contextualizar y clasificar textos
- Búsqueda automática de información en múltiples idiomas, traducción automática, reconocimiento del habla, y otras muchas.

Los aspectos fundamentales de las aplicaciones arriba mencionadas son los que se explican a continuación.

3.2.1.1 **Reconocimiento del habla (ASR, Automatic Speech Recognition).**

Disciplina encargada de convertir los fonemas emitidos por un ser humano en espectros de ondas de audio captados mediante un dispositivo de entrada de sonido de una máquina que, tras ser procesados dentro del contexto de un modelo lingüístico, da lugar a las grafías escritas del mismo; es decir, los sistemas de las máquinas encargados de convertir la voz captada por medio de un sensor (normalmente un micrófono) en la forma escrita de una lengua.

Dada su complejidad, no es posible abarcar una ejemplificación del despliegue de un servidor de inferencia de procesamiento del lenguaje como nVidia JARVIS - RIVA-TRITON.

¿Y qué es un servidor de inferencia de procesamiento del lenguaje natural (PLN)? Pues es una plataforma que ejecuta modelos de inteligencia artificial especializados en el análisis, comprensión y generación de lenguaje natural. Su función principal es recibir textos de entrada y devolver resultados procesados, como traducciones, respuestas automáticas, resúmenes, clasificación de sentimientos, entre otros.

Se presenta a continuación el esquema básico de nVidia NeMo (https://github.com/NVIDIA/NeMo): NeMo - a toolkit for conversational Al - GitHub.

La solución para el ecosistema Jetson es Jetson-voice en lugar de NeMo. La instalación de NeMo puede llevarse a cabo desde dos perspectivas diferentes, bien como un contenedor de Docker mediante el contenedor:

```
sudo docker pull nvcr.io/nvidia/nemo:1.5.1
sudo docker run --runtime=nvidia -it -v /mnt/d/docker/nemo:/
nemo/notebooks/host -p 8888:8888 -p 6006:6006 --shm-size=16g
--ulimit memlock=-1 --ulimit stack=67108864 nvcr.io/nvidia/
nemo:1.5.1
```

O bien para ordenadores portátiles o de escritorio basados en sistemas x 64 por medio de las siguientes instrucciones:

```
apt-get update && apt-get install -y libsndfile1 ffmpeg
pip install Cython
pip install nemo_toolkit['all']
```

Es preciso tener en cuenta la gran diferencia entre los sistemas online (en los que el texto captado en un dispositivo [como en un JetBot] es enviado a un servidor de inferencia [por ejemplo, JARVIS] donde es procesado, y el texto devuelto al robot original), y los sistemas offline, donde toda la operación se realiza en el robot. El primer sistema corresponde al de los asistentes como Alexa o Google.

Para saber más…

En la siguiente carpeta podrá localizar información específica con ejemplos tutorizados sobre el reconocimiento de voz.
NeMo/examples/asr at main • NVIDIA/NeMo • GitHub
NeMo/tutorials/asr at main • NVIDIA/NeMo • GitHub

Figura 3.5 Pantalla de smartphone con aplicación de búsqueda por voz activada.

Desde un punto de vista más general, y sin emplear los núcleos sombreadores de la tarjeta gráfica para acelerar el proceso, existen muchas soluciones académicas y generales que se pueden utilizar para llevar a cabo proyectos de inteligencia artificial.

* Nota: Los **núcleos sombreadores** (*shader cores*) de una **tarjeta gráfica (GPU)** son unidades de procesamiento especializadas en renderizar gráficos y ejecutar cálculos paralelos. Su función principal es procesar y transformar píxeles, vértices y otros elementos gráficos en una imagen final que se muestra en la pantalla.

El siguiente enlace https://pypi.org/project/SpeechRecognition/ corresponde a una librería sobre Python capaz de interactuar con diversas API para llevar a cabo reconocimiento de voz, fácilmente instalable mediante pip en Python3. La librería Pyaudio se empleará para gestionar el micrófono. CMU Sphinx se puede utilizar para trabajar sin conexión a Internet (offline), mientras que otras como la de Google requerirá uso de Internet para llegar a los servidores de procesamiento.

Nota: normalmente las compañías no otorgan licencia para uso comercial gratuito de sus sistemas, consulta las condiciones particulares de cada una.

Para saber más…

Para instalar en Python las librerías se ejecuta desde la terminal los siguientes comandos:

- pip install SpeechRecognition
- pip install pyaudio

3.2.1.2 Síntesis de texto a voz

También conocido como TTS (Text to speech), estos sistemas hacen la labor contraria a un ASR, es decir, a partir de un texto escrito son capaces de reproducirlo mediante el dispositivo de salida de audio (altavoces).

Lo mismo que se señaló para AST en relación con NeMo es de aplicación aquí; los links son los siguientes:
https://github.com/NVIDIA/NeMo/tree/main/examples/tts
https://github.com/NVIDIA/NeMo/tree/main/tutorials/tts

De cara a las unidades JetBot, debido a la baja carga de computación que require un TTS, en lugar de ejecutar NeMo es posible instalar una librería genérica de TTS como espeak TTS (offline) o gtts (online a través de Google). Para reproducir lo se puede emplear cualquier función de salida de audio que se haya instalado, como playsound o por medio de la librería os, emplear una sentencia del estilo os. system ("mpg321 fichero.mp3").

eSpeak TTS (*Text-to-Speech*) es un **sintetizador de voz** de código abierto que convierte texto en habla. Funciona en **modo offline**, lo que significa que no necesita conexión a Internet para generar audio.

gTTS (Google Text-to-Speech) es una librería de Python que permite convertir texto en voz utilizando la API de **Google Translate TTS**. A diferencia de **eSpeak TTS**, **gTTS requiere conexión a Internet**, ya que envía el texto a los servidores de Google y devuelve un archivo de audio con la síntesis de voz.

```
pip install python-espeak
pip install pyttsx3
- - - - - - - - - - - - - - - - - - (tras instalarlo)
import espeak
import pyttsx3
engine = pyttsx3.init()
engine.setProperty('rate', 120)
engine.setProperty('voice', 'spanish')
engine.setProperty('volume', 1)
engine.say("Hola Mundo. Prueba de texto a voz en español")
engine.runAndWait()
```

```
pip install gTTS
- - - - - - - - - - - - - - - - - - (tras instalarlo)
from gtts import gTTS
import os
from playsound import playsound
tts = gTTS('Hola mundo. Prueba de texto a voz.',
 lang='es-es')
tts.save("fichero.mp3")
playsound("fichero.mp3")
```

Más información disponible en la web del fabricante:
https://pypi.org/project/gTTS/

Figura 3.6 Chatbots y asistentes virtuales.

Bien de manera escrita, o bien mediante el auxiliar de las aplicaciones de reconocimiento de la voz (ASR), y de síntesis de texto a voz (TTS), los agentes conversacionales se emplean para multitud de tareas, desde ser asistentes al usuario que lo guíen a través de caminos preconcebidos como asistentes de ventas, o como resolutores de problemas postventas, ligándose enormemente a las métricas y los procedimientos de los departamentos de control de calidad y de satisfacción al cliente.

Para saber más…

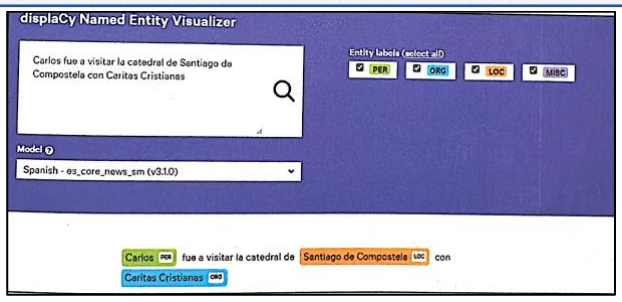
Una aplicación de atención al cliente de un ISP podría emplear un chatbot de tal manera que, en función de los conceptos claves detectados mediante un NER, fuera haciendo preguntas al usuario y sugiriendo posibles caminos de resolución.

3.2.1.3 Detección de entidades nombradas (NER, named entity recognition).

Consiste en trocear el texto (tokenizar) de tal manera que se detecten las palabras clave. Se trata pues de una labor de extracción de información que localiza y clasifica en categorías (normalmente personas, organizaciones, lugares, y cantidades) las entidades encontradas en un texto.

Ejemplo 3

Se trabaja con la herramienta online https://explosion.ai/demos/ que emplea como base Spacy y sus diccionarios.



```
!pip install spacy
!python -m spacy download es_core_news_sm
!python -m spacy download en_core_web_sm
# Tras instalar SPacy y los modelos en español e inglés#
import spacy
from spacy import displacy
from collections import Counter
import es_core_news_sm
nlp = es_core_news_sm.load()
doc = nlp('Carlos enseña a los alumnos a programar con NVIDIA')
print([(X.text, X.label_) for X in doc.ents])
```

Nota: *mediante una orden gráfica y un entorno gráfico de iPython como el de Jupyter, es posible graficar los NER y la tokenización mediante:*

displacy. render (doc, style='dep, Jupyter = True, options = ['distance': 120}).

La salida no gráfica es la siguiente:

[('Carlos', 'PER'), ('NVIDIA', 'ORG')]

Para saber más...

En el siguiente recurso WEB :

Curso de Spacy: https://course.spacy.io/es/chapter1

Comenzando con Spacy: https://medium.com/datos-y-ciencia/comenzando-con-spacy-para-procesamiento-de-lenguaje-natural-e8cf24a18a5a

NLTK : https://adictosaltrabajo.com/2023/07/27/nltk-python/

3.2.1.4 Traducción automática

En el siguiente link se estudia todo lo relativo a la traducción automática.

https://docs.nvidia.com/deeplearning/nemo/archives/nemo-100rc1/user-guide/docs/nlp/machine_translation.html

```
sudo docker pull nvcr.io/nvidia/nemo:1.5.1
sudo docker run --runtime=nvidia -it -v /mnt/d/docker/
 nemo:/nemo/notebooks/host -p 8888:8888 -p 6006:6006 --shm-
 size=16g --ulimit memlock=-1 --ulimit stack=67108864 nvcr.
 io/nvidia/nemo:1.5.1


from nemo.collections.nlp.models import MTEncDecModel
# Se obtiene el listado de modelos preentrenados disponibles
MTEncDecModel.list_available_models()
# Se descarga el modelo de inglés a español
model = MTEncDecModel.from_pretrained("nmt_en_es_transfor-
 mer12x2")
# Se ejecutra sobre la muy conocida frase "Hola Mundo"
translations = model.translate(["Hello World!"],
 source_lang="en", target_lang="es")
print(translations)
```

```
>> translations = model.translate(["Hello World!"], source_lang="en", target_lang="es")
>> print(translations)
['¡Hola mundo!']
>>
```

Una prueba análoga sobre el acto I de Hamlet arroja el siguiente resultado:

```
>>> translations = model.translate(["Bernardo climbed the stairs to the castle's ramparts. It was a bitter
ly cold night. He made his way carefully through the freezing fog to relieve Francisco of his guard duty.
He saw a dim figure and challenged him. 'Who's there?' 'No, you answer me!' It was Francisco's voice. 'Sto
p and identify yourself!' Bernardo stopped. 'Long live the King!'"], source_lang="en", target_lang="es")
(translations)>>> print(translations)
["Bernardo subió las escaleras hasta las murallas del castillo. Era una noche amargamente fría. Se abrió p
aso cuidadosamente a través de la helada niebla para liberar a Francisco de su deber de guardia. Vio una f
igura tenue y le desafió. '¿Quién está ahí?' '¡No, me respondes!' Era la voz de Francisco. '¡Deténgase e i
dentifíquese!' Bernardo se detuvo. '¡Viva el rey!'"]
>>>
```

3.2.1.5 Similitud de textos

Algunos modelos (normalmente de tamaño considerable) han sido entrenados de tal forma que las palabras son etiquetadas de manera que aludan a conceptos semánticos (Figura 3.7).

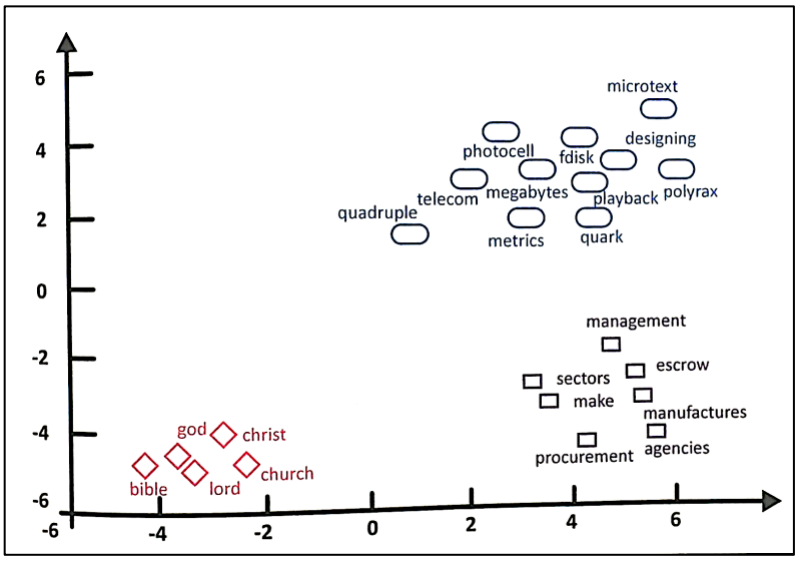


Figura 3.7
Similitud de textos.

Por ejemplo, un perro es un mamífero y a su vez es un animal. Un etiquetado correcto situará en lugares más próximos entre sí a un perro y a un gato (dos mamíferos) que a un perro y a un salmón. De la misma manera una manzana es una fruta, que a su vez es comida. En buena lógica es más «parecido» semánticamente a una manzana una naranja que una zanca de pollo, aunque todo sea comida.

Mediante técnicas de vectorización y de cálculo del coseno es posible analizar dos textos y decir el «parecido» que tengan entre sí, aunque lógicamente esta apreciación de parecido es subjetiva y depende del set de datos que haya sido empleado para llevar a cabo el entrenamiento.

Ejemplo 4

Este ejemplo se realiza con Spacy, si bien es preciso puntualizar que se emplea el diccionario español de mayor tamaño, que ha de ser previamente descargado (es_core_news_lg).

```
import spacy
from spacy import displacy
from collections import Counter


import es_core_news_lg
nlp = es_core_news_lg.load()
doc1 = nlp('Carlos se come una manzana')
doc2 = nlp('María se come una ensalada')
doc3 = nlp('María y carlos se comen un plato de pasta ‹)
doc4 = nlp('María y carlos ven una película')
print(doc1, "<->", doc2, doc1.similarity(doc2))
print(doc1, "<->", doc3, doc1.similarity(doc3))
print(doc1, "<->", doc4, doc1.similarity(doc4))
print(doc3, "<->", doc4, doc3.similarity(doc4))
```

```
PROBLEMAS   SALIDA   TERMINAL   CONSOLA DE DEPURACIÓN

PS C:\Users\Usuario\Documents\Master UNED\Marcombo> & C:/Python38/python.exe "c:/Users/Usuario/Documents
Carlos se come una manzana <-> María se come una ensalada 0.9545383806411006
Carlos se come una manzana <-> María y carlos se comen un plato de pasta  0.6870097213414148
Carlos se come una manzana <-> María y carlos ven una pelicula 0.3394106710397441
María y carlos se comen un plato de pasta  <-> María y carlos ven una pelicula 0.4397361911649576
PS C:\Users\Usuario\Documents\Master UNED\Marcombo>
```

Como puede verse las dos primeras frases son muy parecidas (95%), dado que Carlos es una persona, María es otra persona, y manzana y ensalada aluden a comidas relativamente parecidas. Sin embargo, ver una película y comer un plato de pasta son asuntos distintos, aunque sean los dos realizados por María y Carlos, lo cual reduce la similitud de la frase 3 con la 4 a un 43,9%, y entre las frases 1 y 4 cae a un 33,9% ya que no sólo el predicado es distinto, sino que el sujeto también cambia.

3.2.1.6 Análisis del sentimiento

El procesamiento del lenguaje natural puede emplear también el etiquetado característico del aprendizaje automático supervisado para entrenar un modelo, de tal manera que sea capaz de captar la positividad o negatividad de un texto en relación con un tema particular.

Esta potente herramienta es especialmente útil para llevar a cabo sondeos «invisibles» de la opinión de grandes grupos muestrales alrededor de un tema. Las redes sociales, y especialmente Twitter, conforman un campo de datos especialmente bueno para llevar a cabo este tipo de aplicaciones, ya que permiten analizar la opinión de un gran grupo de personas (por ejemplo, todo un país) alrededor de un asunto, o de una persona, lo que indirectamente se puede considerar un excelente sondeo de la opinión sobre la valoración de un político, o de un evento. También puede permitir a un asistente o chatbot, en una conversación larga, averiguar el estado de ánimo del usuario, y actuar en consecuencia.

Figura 3.8 Análisis del sentimiento.

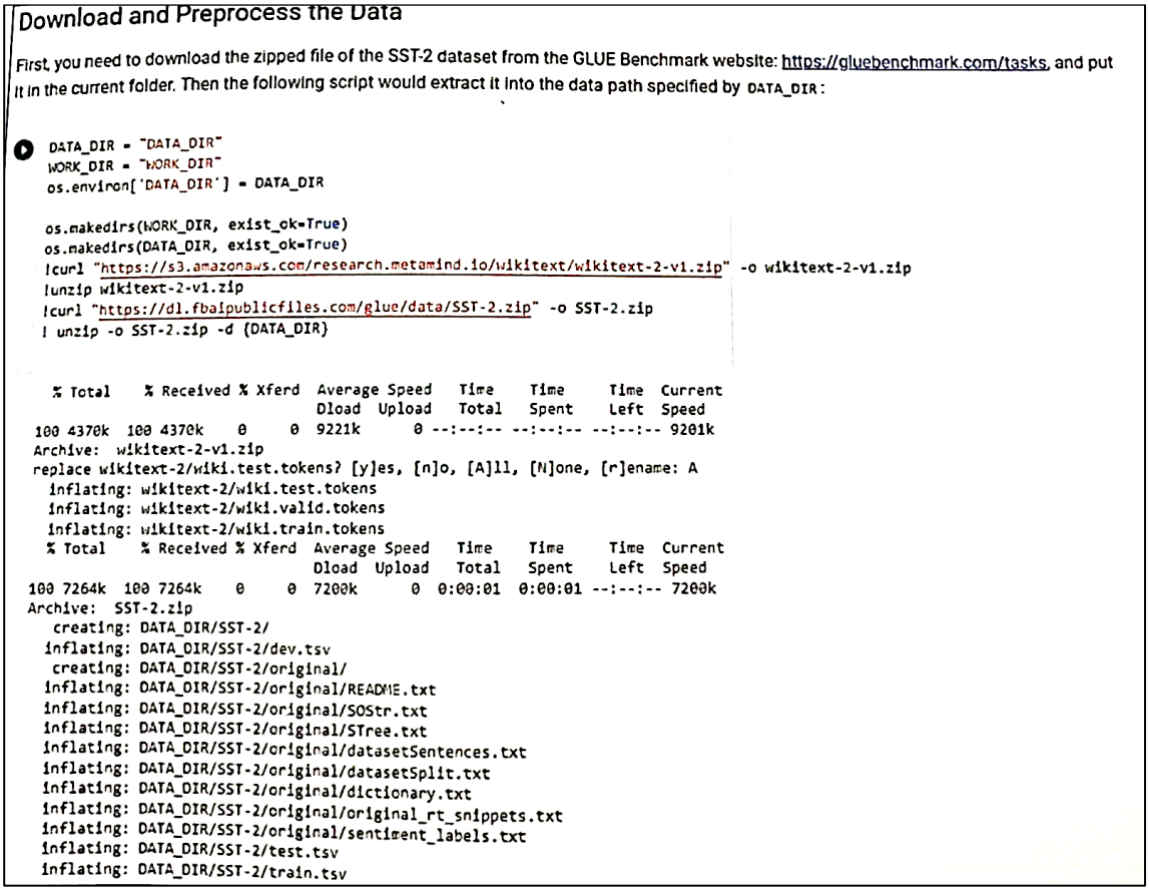
Un excelente ejemplo para ver cómo llevar a cabo un análisis del sentimiento puede encontrarse en el repositorio de GitHub de NeMo y ser ejecutado sobre Google Colab, o de manera local:

https://colab.research.google.com/github/NVIDIA/NeMo/blob/stable/tutorials/nip/Text_Classification_Sentiment_Analysis.ipynb

No obstante, el código tal cual está planteado fallaría por la falta de descarga del set de datos. Como indica el tutorial, sería necesaria la descarga del set de datos SST-2 de la página web indicada, aunque también es posible ejecutar cur/ para su descarga (si bien queda de la mano del lector comprobar que el sitio de descarga cumple los términos de licencia del set de datos, y que no ha sido alterado con ningún tipo de código malicioso. La siguiente captura de pantalla es tan sólo un ejemplo del uso de curl, sobre una fuente no comprobada, propuesta por terceros en:

https://gist.github.com/W4ngatang/60c2bdb54d156a41194446737ce03e2e.

Es importante llevar a cabo la descarga desde la página oficial.



Download and Preprocess the Data

First, you need to download the zipped file of the SST-2 dataset from the GLUE Benchmark website: https://gluebenchmark.com/tasks, and put it in the current folder. Then the following script would extract it into the data path specified by `DATA_DIR`:

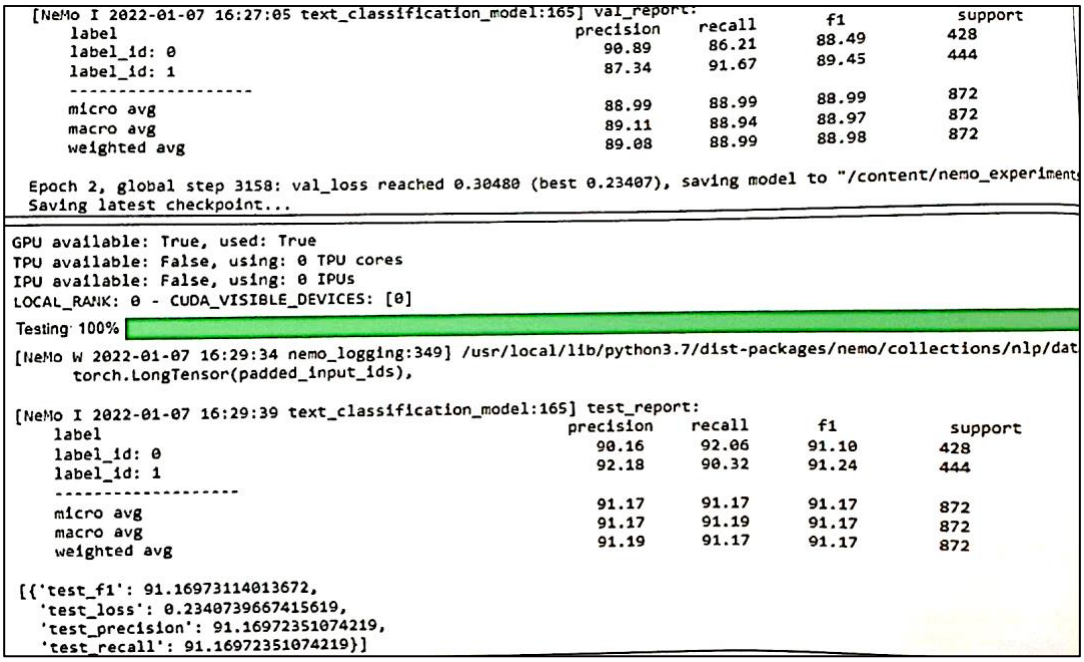
```
DATA_DIR = "DATA_DIR"
WORK_DIR = "WORK_DIR"
os.environ['DATA_DIR'] = DATA_DIR

os.makedirs(WORK_DIR, exist_ok=True)
os.makedirs(DATA_DIR, exist_ok=True)
!curl "https://s3.amazonaws.com/research.metamind.io/wikitext/wikitext-2-v1.zip" -o wikitext-2-v1.zip
!unzip wikitext-2-v1.zip
!curl "https://dl.fbaipublicfiles.com/glue/data/SST-2.zip" -o SST-2.zip
! unzip -o SST-2.zip -d (DATA_DIR)
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 4370k  100 4370k    0     0  9221k      0 --:--:-- --:--:-- --:--:-- 9201k
Archive:  wikitext-2-v1.zip
replace wikitext-2/wiki.test.tokens? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
  inflating: wikitext-2/wiki.test.tokens
  inflating: wikitext-2/wiki.valid.tokens
  inflating: wikitext-2/wiki.train.tokens
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 7264k  100 7264k    0     0  7200k      0  0:00:01  0:00:01 --:--:-- 7200k
Archive:  SST-2.zip
   creating: DATA_DIR/SST-2/
  inflating: DATA_DIR/SST-2/dev.tsv
   creating: DATA_DIR/SST-2/original/
  inflating: DATA_DIR/SST-2/original/README.txt
  inflating: DATA_DIR/SST-2/original/SOStr.txt
  inflating: DATA_DIR/SST-2/original/STree.txt
  inflating: DATA_DIR/SST-2/original/datasetSentences.txt
  inflating: DATA_DIR/SST-2/original/datasetSplit.txt
  inflating: DATA_DIR/SST-2/original/dictionary.txt
  inflating: DATA_DIR/SST-2/original/original_rt_snippets.txt
  inflating: DATA_DIR/SST-2/original/sentiment_labels.txt
  inflating: DATA_DIR/SST-2/test.tsv
  inflating: DATA_DIR/SST-2/train.tsv
```

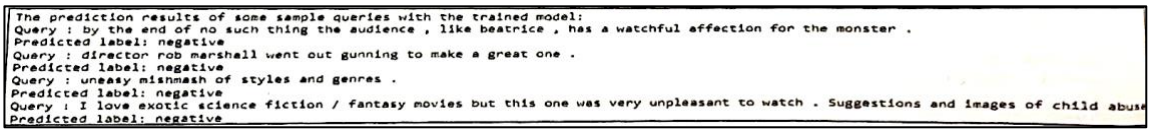
Hay que tener en cuenta que las diferentes versiones de NeMo requieren distintas dependencias, y es posible que en el primer paso del tutorial sea necesario añadir mediante ejecución de la CLI a través del magic command Ipip un determinado paquete en una determinada versión. Por ejemplo:

```
!pip install folium==0.8.3
!pip install imgaug==0.2.5
```

Nota: se recomienda verificar el número de etapas de entrenamiento:
config. trainer .max_epochs = 3

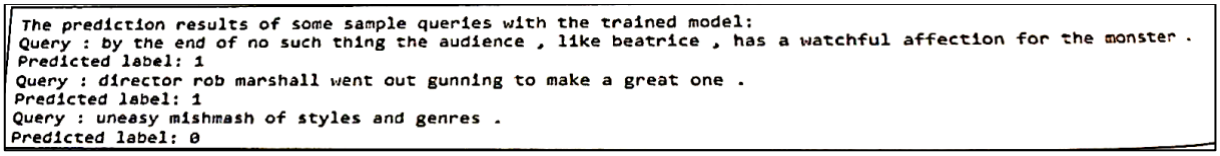


Los resultados obtenidos, como puede observarse, son extremadamente buenos, determinando que críticas negativas son realmente negativas.



Mientras que las críticas positivas son detectadas como positivas (etiquetadas como 1, frente a las negativas que se etiquetan como 0) en virtud de las etiquetas y textos definidos en:

processed_results = postprocessing (preds, {"0": "negative", "1": "positive"})



3.2.2 Modelos y técnicas existentes

Para una correcta explicación de los modelos empleados en procesamiento de lenguaje natural (en especial BERT) es necesario entender el proceso de **atención** (en inglés, attention) y el modelo de **transformadores.**

Para lo primero a su vez sería preciso comprender el funcionamiento de redes neuronales LSTM (Long Short-Term Memory).

Una red neuronal modelada con un proceso de atención que actúa de una manera similar a como lo hace la atención del ser humano. Un ser humano atiende (o deja de

atender) a la información que está recibiendo en función a la relevancia e interés que le suscita. Una red neuronal que lleva a cabo un proceso de atención es capaz «de hacer caso» a algunos valores de sus variables de estado en tiempos pasados, mientras que ignora otras. Este hecho es muy importante cuando por ejemplo se hace la traducción de una lengua a otra, dado que el orden de los sintagmas de las frases no siempre sigue la misma colocación, produciéndose un efecto de inversión, algo muy característico por ejemplo al traducir del alemán al español. Así pues, un traductor necesita almacenar no sólo la palabra previa a la que está actualmente procesando dentro de la frase, sino toda ella, dado que tal vez la primera palabra en una frase en un idioma, al ser traducida aparezca al final en otro idioma. Por tanto, una red que emplea este mecanismo utilizará no solo los pesos de la última iteración en el proceso de propagación inversa, sino que empleará también pesos de iteraciones anteriores para algunas variables. Es importante resaltar que únicamente en algunas variables, y que, durante el entrenamiento de la red, parte de la complejidad para ella es determinar a qué valores previos les presta atención y cuál los descarta.

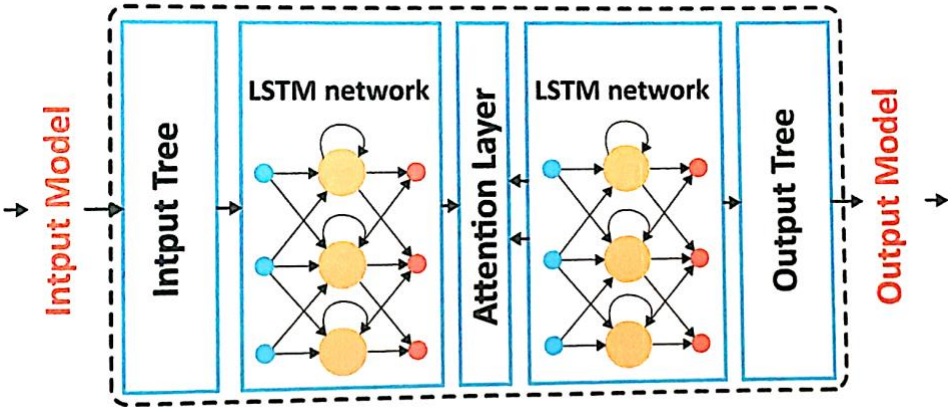


Figura 3.9 Red neuronal.

Los transformadores usan un proceso de primero codificación y posteriormente decodificación, con etapas de atención y autoatención para de forma semi-supervisada entrenar la red. Información específica sobre el funcionamiento del modelo, de alto poder didáctico, puede ser obtenida en el siguiente enlace, http://jalammar.github.io/illustrated-transformer/ aunque la mejor explicación se encuentra en su artículo original «Attention Is All You Need» https://arxiv.org/abs/1706.03762 . De la misma manera el lector puede afianzar prácticas en este campo a través de los recursos de NVidia y su curso Building Transformer-Based Natural Language Processing Applications, que explican y practican con alto detalle el uso de la GPU en transformadores y mecanismos de atención con modelos BERT y MEGATRON.

```
docker run --runtime=nvidia -d -v $PWD: /dli/task --shm-size=16g -p 8888:8888 -P
6006:6006 --ulimit memlock=-1
--ulimit stack=67108864 nvcr.io/nvidia/dli/dli-nlp-nemo:
```

BERT (Bidirectional Encoder Representations from Transformers)

Como su nombre indica, incluye transformadores, que por tanto hacen uso del proceso de atención. Fue desarrollado en 2018 por el equipo de Google (Jacob Devlin et al.), como evolución de las redes neuronales recurrentes (con o sin me-moria).

BERT tiene dos etapas: una primera con aprendizaje no supervisado, y una posterior con aprendizaje supervisado dedicada exclusivamente al afinado del modelo.

Una vez que el modelo queda preentrenado puede ser compartido, lo cual reduce las necesidades de tamaño de los sets de entrenamiento de manera similar al funcionamiento del transfer learning en visión artificial.

BERT tiene su origen en modelos de aprendizaje de secuencia semisupervisado como ELMO (Microsoft) y ULMFit, pero BERT es bidireccional (de cara al mecanismo de atención y la recurrencia de las redes neuronales se analizan las palabras ubicadas a la izquierda y a la derecha de cada uno de los términos actuales) y sin supervisión (no requiere un corpus anotado) que busca mediante los autotrans-formadores recrear una salida a partir de una entrada, auto-validándose.

Otra característica de BERT es que a diferencia de modelos como word2vec o Glo-Ve que generan una representación de una única palabra para cada palabra en el vocabulario, BERT tiene en cuenta el contexto para cada aparición de una palabra concreta, ayudando a la desambiguación. Por ejemplo, mientras que el vector para palabras polisémicas como «bolsa» tendrá la misma representación vectorial de word2vec en las oraciones «Se ha caído la bolsa de Nueva York» y «La bolsa de la compra está llena», BERT proporciona una representación diferente para cada oración. Debido a que el modelo de BERT tiene entre 12 y 24 capas con varios mecanismos de atención, para hacer esto aplicable en un entorno de operaciones nVidia desarrolló una optimización de TensoRT para BERT.

Otros modelos

- **RoBERTa** (A Robustly Optimized BERT Pretraining Approach): es una evolución sobre BERT introducida por el equipo de Facebook, que difiere del modelo de BERT especialmente en la manera de llevar el enmascaramiento en el proceso de atención (la forma en la que se decide cuando hacer caso o no hacer caso a un valor de la red recurrente). También incluye cambios en el proceso de afinado de los hiperparámetros, en el tamaño de lote, y en el valor de la tasa de aprendizaje. Es de entrenamiento más lento, emplea un set de datos diez veces mayor, y globalmente consigue una mejora de entre el 2 y el 20% dependiendo de la fuente consultada.

- **ALBERT** (A Little BERT): es una modificación del equipo de Google que incorpora al modelo dos técnicas de reducción de parámetros: una parametrización factorizada embebida y una capa cruzada de compartición de parámetros que supuso una mejora del 70% en coste computacional medido en tiempo (x1.7 a la velocidad) con una pérdida de precisión apenas apreciable.

- **DistilBERT o Distilled BERT:** lleva a cambio modificaciones en el algoritmo para que a cambio de sacrificar precisión se mejoren los tiempos y disminuyan los requisitos de costo computacional (reducción de hasta 4 veces menos que el tiempo de BERT a cambio de perder un 3% de precisión).

- **XLNET:** contrario a lo logrado en DistilBERT, a cambio de Incrementar el tiempo de entrenamiento (x5) obtiene una mejora de entre el 2 y el 15% en la precisión. Al igual que en RoBERTa el cambio principal se produce en el enmascarado.

- **MEGATRON:** desarrollado por nVidia, es un modelo con 8.3 millardos de parámetros entrenado en 512 GPU nVidia TESLA V100, alcanzando el tamaño de 24 veces BERT y 5.6 veces GPT-2. Es el mayor modelo conocido por este equipo a fecha de edición de este texto.

- **GPT-3:** es una evolución del GPT original de OpenAl. Lo primero que ha de decirse es que no es un modelo open source como el de Google, sino que es propietario, y por tanto de uso restringido. La diferencia fundamental respecto de BERT es tu tamaño (es colosal frente a BERT) y que no es bidireccional, lo cual hace que las labores de afinado sean tan costosas que es casi impensable. No obstante, el modelo es autorregresivo, y su gran tamaño (175 millardos de parámetros frente a 340 millones por parte de BERT), tanto en el modelo como en los sets de entrenamiento (499 millardos de tokens de varios corpus frente a 2.5 millardos de tokens procedentes de la Wikipedia para BERT), hacen que sobrepase a BERT en muchos campos, sin requerir afinamiento.

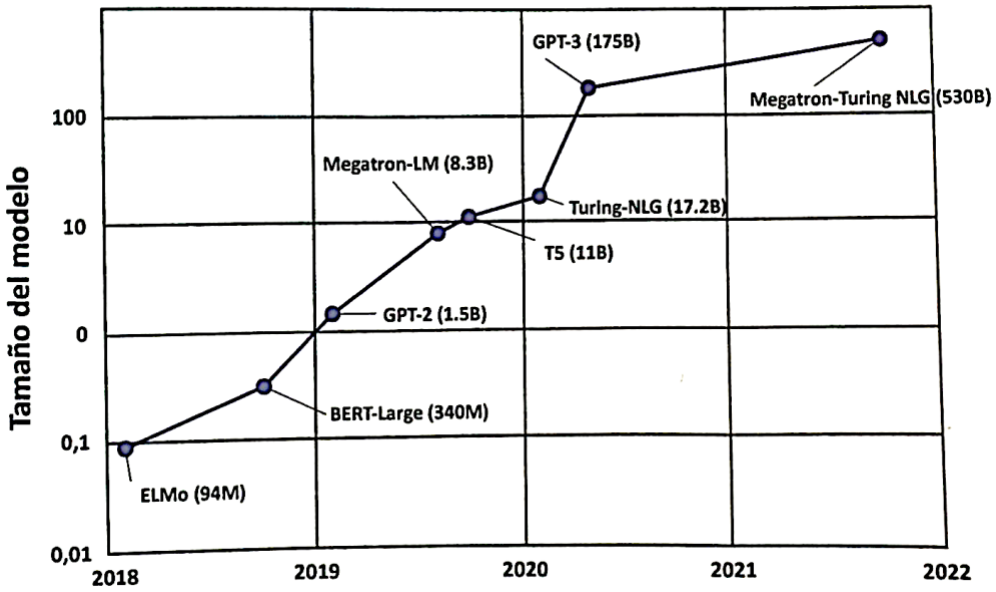


Figura 3.10 Modelos y técnicas.

3.2.3 Limitaciones. La ambigüedad

Una de las limitaciones mayores alrededor del procesamiento de lenguaje natural es la ambigüedad lingüística. Múltiples interpretaciones de una misma palabra pueden arruinar la capacidad de un sistema automático para responder correctamente y desarrollar su función. El etiquetado y el análisis visto en el apartado anterior ayuda a evitar este tipo de errores. Un ejemplo detallado para el español puede verse en el artículo de C. Zapata et al., de la Universidad de Colombia (Zapata, 2007).

Los problemas de ambigüedad más comunes en los textos son los siguientes:

- **Ambigüedad sintáctica:** se presenta cuando una oración tiene asociada más de una representación sintáctica, es decir, si hay más de una regla gramatical representa dicha oración.

  Un ejemplo de ambigüedad sintáctica es la siguiente oración:
  "Vi a la mujer con prismáticos."

Esta oración puede tener dos interpretaciones:
1. Yo usé los prismáticos para ver a la mujer.
2. La mujer tenía prismáticos cuando la vi.

La ambigüedad surge porque la frase "con prismáticos" puede modificar tanto al verbo ("vi") como al sustantivo ("la mujer"), generando distintas interpretaciones.

- **Ambigüedad léxica/morfológica (gramatical):** la primera ocurre cuando la duda surge respecto a un término aislado, que admite diversas interpretaciones. La segunda tiene lugar si una palabra que se encuentra en una oración representa más de un rol sintáctico o categoría gramatical dentro de la misma.

  Un ejemplo de ambigüedad léxica es la palabra "banco", que puede significar:
  Institución financiera: **"Voy al banco a sacar dinero."**
  Asiento largo para sentarse: **"Me senté en el banco del parque."**
  Aquí, la ambigüedad se debe a que la misma palabra tiene múltiples significados.

- **Ambigüedad semántica:** se presenta cuando afecta a un elemento de la frase que puede ser interpretado de diversos modos.

  Un ejemplo de **ambigüedad semántica** es la oración:

  "Juan vio a su hermano borracho."

  Esta frase puede interpretarse de dos maneras:

  1. **Juan estaba borracho cuando vio a su hermano.**
  2. **El hermano de Juan estaba borracho cuando Juan lo vio.**

  La ambigüedad surge porque no queda claro a quién se refiere el adjetivo "borracho".

- **Ambigüedad pragmática:** aparece si la realidad depende del contexto del lenguaje y del hablante, en un momento dado.

  Un ejemplo de **ambigüedad pragmática** es la siguiente conversación:

  Persona 1: *"¿Puedes pasarme la sal?"*
  Persona 2: *"Sí."* (pero no la pasa).

  Aquí, la ambigüedad radica en la interpretación del significado dentro del contexto. Aunque la pregunta gramaticalmente pide información (sobre la capacidad de pasar la sal), en la práctica se espera que sea una petición para que la persona efectivamente la pase.

- **Ambigüedad fonológica:** se presenta cuando una cadena de sonidos de resultar confusa.

  La **ambigüedad fonológica** ocurre cuando dos palabras o frases suenan igual pero tienen significados diferentes.

  Un ejemplo clásico en español es:

  "Vino viejo"

  - Puede significar **una persona mayor que vino (verbo venir)**.
  - O puede referirse a **un vino (bebida) añejo**.

  "Echo de menos" vs. **"Hecho de menos"**

  - *"Echo de menos"* significa extrañar a alguien.
  - *"Hecho de menos"* podría interpretarse como algo pequeño o escaso, aunque es menos común.

La ambigüedad se resuelve con el contexto de la conversación.

- **Ambigüedad funcional:** se observa si se emplea un término con doble función gramatical.
  Un ejemplo: **"Pepe vio a Luis corriendo."**
  Puede interpretarse de dos maneras:

  1. **Pepe estaba corriendo mientras veía a Luis.** (*"corriendo" funciona como adverbio, modificando "vio"*)
  2. **Luis estaba corriendo cuando Pepe lo vio.** (*"corriendo" funciona como adjetivo, modificando "Luis"*)

  La ambigüedad surge porque **"corriendo"** puede actuar como un modificador del sujeto o del complemento directo. El contexto es clave para resolver la interpretación correcta.

  "**He vuelto a oler"** (antes no tenía olfato y ahora sí; o bien, regresé a un lugar a oler algo para comprobar su aroma).

3.3 Formación del investigador en PLN

Un experto en sistemas de inteligencia artificial que desee adquirir conocimientos en procesamiento del lenguaje natural puede seguir muchos itinerarios. El aquí propuesto es una ruta tradicional, basada en una primera formación teórica y clásica, y una posterior aplicación de los métodos de procesamiento del lenguaje natural basados en redes neuronales. Por ello se aconseja la lectura del texto de Jurafsky, y posteriormente la lectura del manual base del Natural Learning ToolKit (NLTK) en lengua inglesa. A partir de ahí, continuar con Spacy en lenguas inglesa y española, y finalmente dar el paso a nVidia NeMo para ejecución de ASR, TTS y NLP en plataformas Jetbot y posteriormente crecer a un entorno de servidores tipo JARVIS/TRITON.

- **NLTK:** puede ser descargado desde el portal web propietario https://www.nltk.org/ y a partir de ahí leer las instrucciones de instalación https://www.nitk.org/install.html que son muy simples, y descargar los sets de datos mediante el comando nltk. download (all) dentro del entorno de Python 3 (y tras ejecutar el import correspondiente). Incluso puede instalar el Stanford CoreNLP API para NLTK desde https://github.com/nltk/nltk/wiki/Stanford-CoreNLP-API-in-NLTK que incluye modelo para nuestra lengua. Dentro de NLTK debería seguir el orden lógico que se muestra en el NLTK-BOOK público (disponible en https://www.nltk.org/book/):
    - Cargar un corpus.
    - Usar expresiones regulares.
    - Tokenizar y etiquetar.
    - Emplear lo anterior para hacer etiquetado POS.
    - Usar diccionarios de eliminación de stop-words.
    - Usar N-gramas.
    - Lematizar.
    - Programar un clasificador (por ejemplo, de películas según temática o de noticias).
    - Programar un comparador de textos.
    - Programar un analizador de sentimientos.
    - Programar un sistema de ideas clave o resumen.
    - Emplear un TTS (no desde NLTK).
    - Emplear un ASR (no desde NLTK).
    - Programar un procesador semántico real.

- **Spacy:** puede ser descargado e instalado siguiendo las instrucciones de https://spacy.io/ . En él se debería practicar lo anterior, además de:
    - Vectorización y comparativa por vectorización a partir de modelos pre entrenados disponibles en el programa como los empleados en este texto.
    - Lo mismo con otros modelos entrenados, como variaciones de BERT.
    - Generar un modelo propio de vectorizado por etiquetación, con un set muy cerrado, de tal manera que las similitudes sean más específicas al texto que se esté tratando.
    - NER, y análisis morfológico con visualización de grafos a través de displacy.
    - Traducción entre idiomas.

- **NeMo:** ha de ser empleado de manera global para NLP, TTS y ASR siguiendo los tutoriales y ejemplos de su GitHub, destacándose entre otros, los siguientes:

- **ASR:**
  - Detección de voz (Voice activity detection) a través de micrófono
  - Voz a Texto en PC x86, offline.
  - Voz a Texto en PC x86, online.
  - Voz a texto en Jetbot.
  - Comandos de voz (la base para un asistente).
- **TTS.**
- **NLP:**
  - Lo mismo que en Spacy.
  - Uso del modelo de Megatron.
  - Programar un Q&A tipo Jeopardy.

3.4 Elaboración de un sistema de procesamiento de lenguaje orientado a una tarea específica.

Se propone aunar todo lo aprendido en la generación de un código que tras adquirir vía ASR la conversación de un usuario con un robot de cocina, sea capaz de hacer que este funcione, teniendo en cuenta las siguientes limitaciones:

- El robot puede cocinar mediante calor (cocer). Esta operación requiere de una temperatura en grados Celsius y de un tiempo expresado en minutos,
- hasta un máximo de 120 °C.
- El robot puede preparar alimentos agitando un instrumento, como por ejemplo batir o remover algo (batir. Esta operación requiere de una velocidad expresada en un valor del 0-10 y de un tiempo expresado en minutos, hasta un máximo de 120.
- El robot puede parar (lo cual implica velocidad de giro O, temperatura 0°C, tiempo 0).
- El robot ha de poder entender frases alternativas, es decir, en lugar de batir, emplear el verbo remover, o agitar, o uso de verbos generales como «pon el robot a velocidad 5».
- El robot ha de poder entender frases compuestas, que especifiquen en un solo comando temperatura, velocidad y tiempo.

El abordaje de esta labor es sencillo:

5. Un ASR convierte la voz a texto.

6. El texto es procesado:

a) Se quitan las palabras inútiles (stop words).
b) Se unifican mayúsculas, minúsculas y signos.
c) Se tokeniza y se etiqueta.
d) Se localizan los tokens correspondientes a verbos de acción (cocer, batir).
e) Se localizan los tokens correspondientes a valores numéricos.
f) Se va haciendo agregación sintagmática.
g) Se activan las salidas en una GPIO (lo veremos en próximo tema).