
**IRIS: Asistente virtual para la redacción personalizada
de correos electrónicos**

IRIS: Virtual Assistant for Personalized Email Writing



CENTRO UNIVERSITARIO
DE TECNOLOGÍA Y ARTE DIGITAL

**Trabajo de Fin de Máster
Curso 2020–2021**

Autor
Carlos Moreno Morera

Director
Carlos Rodríguez Abellán

Máster en Data Science y Big Data
U-Tad

IRIS: Asistente virtual para la redacción personalizada de correos electrónicos

IRIS: Virtual Assistant for Personalized Email Writing

Trabajo de Fin de Máster en Data Science y Big Data

Autor
Carlos Moreno Morera

Director
Carlos Rodríguez Abellán

Convocatoria: Septiembre 2021
Calificación: *Nota*

Máster en Data Science y Big Data
U-Tad

3 de octubre de 2021

Dedicatoria

A mi madre y mi padre

Resumen

IRIS: Asistente virtual para la redacción personalizada de correos electrónicos

Un resumen en castellano de media página, incluyendo el título en castellano. A continuación, se escribirá una lista de no más de 10 palabras clave.

Palabras clave

Máximo 10 palabras clave separadas por comas

Índice

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Estructura del documento	2
2. Estado de la Cuestión	5
2.1. Correo electrónico	5
2.1.1. MIME	7
2.1.2. SMTP	9
2.1.3. POP	10
2.1.4. IMAP	11
2.2. Generación de Lenguaje Natural	12
2.2.1. ¿Qué es la Generación de Lenguaje Natural?	14
2.2.2. Arquitecturas para la Generación de Lenguaje Natural	15
2.2.3. Resumen abstractivo de textos	30
3. Descripción del Trabajo	33
3.1. Tecnologías utilizadas	33
3.1.1. spaCy	34
3.1.2. textaCy	35
3.1.3. Tensorflow	36
3.1.4. MongoDB	36
3.2. Análisis de los datos	36
3.2.1. Enron corpus	37
3.2.2. Preparación y limpieza de los datos	37
3.2.3. Procesado y almacenamiento	39
3.2.4. Análisis exploratorio de los datos	42
3.3. Implementación de una arquitectura realizer	47
3.4. Implementación de la arquitectura transformer	48
3.5. Evaluación de los resultados	49
4. Conclusiones y Trabajo Futuro	51
4.1. Conclusiones	51
4.2. Trabajo futuro	52

Índice de figuras

2.1. Estructura arbórea de tipos MIME de un e-mail de ejemplo	9
2.2. Mensaje MIME de la figura 2.1	10
2.3. Porcentaje de tiempo de un trabajador dedicado a cada tarea	13
2.4. Arquitectura modular secuencial propuesta por Reiter y Dale (2000) para la NLG	17
2.5. Ejemplo de mensaje	18
2.6. Modelo de arquitectura transformer	23
2.7. Esquema de atención query-key	27
2.8. Cálculo de atención (Query-Key-Value)	28
2.9. Multi-Head Attention	29
3.1. Benchmarks de los distintos analizadores sintácticos	34
3.2. Tiempo de procesamiento por documento de varias librerías de NLP	35
3.3. Ejemplo de un correo electrónico del corpus Enron	38
3.4. Ejemplo de documento de un Information Item	41
3.5. Distribución del número de palabras en los mensajes	42
3.6. Distribución del número de palabras en los mensajes delimitando los ejes	43
3.7. Distribución del número de Information Items en los mensajes	43
3.8. Distribución del número de Information Items en los mensajes delimitando los ejes	44
3.9. Distribución acumulada normalizada del número de Information Items	45
3.10. Distribución del número de palabras en función del número de Information Items	46

Índice de tablas

2.1. Previsión de usuarios de correo electrónico en todo el mundo (2021-2025)	6
2.2. Tráfico diario de correos electrónicos en todo el mundo (2021-2025)	6
3.1. Atributos de interés de la clase <i>Tokenizer</i>	35
3.2. Estadísticos descriptivos del número de palabras agrupados por el número de Information Items	45

Capítulo 1

Introducción

“¿Alguna vez has retirado un humano por error?”
— Rachael - Blade Runner (1982)

En este capítulo se presenta la motivación que incentiva el desarrollo de este trabajo (véase la sección 1.1) y los objetivos que se pretenden alcanzar (apartado 1.2).

1.1. Motivación

El reciente desarrollo y el consecuente auge que los *smartphones* han llevado a cabo en las últimas décadas, ha traído consigo innumerables cambios de paradigma, tanto en el término de lo tecnológico como en el ámbito social. Precisamente, el fácil y cada vez más sencillo acceso que supuso la aparición de los smartphones conectados a una red 3G a principios de los 2000, permitió acercar a un público emergente y cada vez mayor, herramientas, hoy en día consideradas esenciales, como las videoconferencias o la mensajería instantánea. Fue precisamente este sistema de mensajería “pionero” el que terminó evolucionando y desembocando en el desarrollo de distintas aplicaciones que han ayudado a la popularización de la comunicación en tiempo real, llegando a enviar hasta cien mil millones mensajes de manera diaria en 2020. Aunque esta serie de aplicaciones cuentan con la principal característica de ser más cercanas, informales e inmediatas –de modo que la necesidad de inmediatez y la urgencia de los emisores queda satisfecha prácticamente al segundo–, esto no les ha sido suficiente para relevar totalmente de sus tareas a sistemas de mensajería más “veteranos” como el correo electrónico, que requieren de un mayor tiempo de dedicación para su correcta utilización, dada la mayor complejidad, riqueza y elaboración del lenguaje que en ellos conviene presentar, debido a su carácter y “personalidad” más formal. Si bien es cierto que, aunque hoy en día, general y habitualmente el correo electrónico se destine a ámbitos más formales o institucionales, como exordio de una comunicación que probablemente evolucione hacia otro tipo de plataformas o, más informalmente, para el envío de archivos, este continúa siendo un medio de comunicación de uso habitual; en el caso de algunas personas, este uso se extiende incluso al ámbito diario, pero con la clara y evidente desventaja de que requiere una mayor atención por parte del usuario.

Resulta evidente que existe una clara brecha entre aplicaciones de mensajería instantánea y correo electrónico y que esta, debe ser solventada para facilitar la experiencia y el uso del correo a sus usuarios; por ello, se propone IRIS. El sistema IRIS (*Intelligent Response Instantly Sent*) se postula como un asistente virtual cuyo objetivo es mejorar la experiencia de usuario en la utilización del correo electrónico, abordando la problemática

de la dedicación requerida que este exige; en consecuencia, el *modus operandi* de IRIS es sencillo. Al utilizar IRIS, el usuario postula la idea que quiere transmitir y el tema en torno al que debe girar el cuerpo del mensaje y esto es recibida por el asistente, que mediante técnicas de generación de lenguaje natural, tratará de expresar dicha idea en un formato de texto de manera extensa, para posteriormente enviar el correo electrónico a su destinatario.

1.2. Objetivos

Los principales objetivos de este trabajo son el estudio, la implementación y la evaluación de las distintas técnicas de generación de lenguaje natural (NLG) que permiten redactar correos electrónicos de manera automática a partir de pequeñas representaciones semánticas. Para lograrlo, se entrará en detalle tanto de las distintas arquitecturas del campo de la NLG y se profundizará en las alternativas para producir estas representaciones semánticas que luego puedan dar lugar a mensajes completos. En concreto, se estudiarán los llamados Information Items, utilizados en el campo del resumen automático de textos, para implementar estas representaciones semánticas y evaluarlas con las dos arquitecturas principales hoy en día en el campo de la NLG: los modelos transformers y realizers.

Se pretende desarrollar soluciones capaces de, recibiendo como entrada los Information Items, redacten un correo electrónico por completo como lo haría el usuario. Para lograrlo, se elegirá un corpus suficientemente grande de mensajes que permita entrenar el modelo adecuadamente. Este conjunto de documentos, se analizará para conocer las características y propiedades que lo definen, se llevarán a cabo tareas de limpieza de los cuerpos de los correos electrónicos y de procesamiento y almacenamiento de los mismos. Para esta última fase, se definirá una estructura adecuada con un sistema de almacenamiento óptimo que permita gestionar la información de manera eficiente.

Una vez se cuente con un sistema de almacenamiento y se hayan llevado a cabo las tareas de análisis y procesamiento de los datos, se implementará el modelo de la arquitectura de generación de lenguaje natural que mejor se ajuste al problema propuesto y se entrenará con los datos del corpus. Finalmente, se pretende extraer resultados que den a conocer el rendimiento y performance del sistema desarrollado, y así poder evaluar la eficacia de la solución propuesta ante el problema de redacción automática de correos electrónicos con técnicas de generación de lenguaje natural.

1.3. Estructura del documento

Este documento presenta los detalles del trabajo desarrollado, mostrando la justificación y motivación para las diferentes decisiones y pasos que se han tomado, y deja patente los distintos resultados, conclusiones y razonamientos a los que se ha llegado tras analizar todo lo implementado.

Comienza, como se ha observado, con una introducción (capítulo 1) que pretende dar una idea del problema al que se intenta dar respuesta y la utilidad de la solución que se construye. A esta motivación (sección 1.1) le sigue la presentación de los objetivos (apartado 1.2) que se abarcarán a lo largo de todo el proyecto. Estos propósitos serán los que dirigirán las decisiones tomadas en todo el trabajo y hacia los que se pondrá el foco de atención en todo momento. La introducción acaba con esta sección en la que se explica la estructura del documento.

Una vez se ha introducido adecuadamente la motivación y los objetivos del estudio, se lleva a cabo una revisión general del estado de la cuestión (capítulo 2) de los temas que

aborda el proyecto. Con este capítulo, el lector podrá adoptar los conocimientos necesarios que constituyen la base práctica y teórica del desarrollo presentado. Comenzando con los fundamentos del correo electrónico (sección 2.1), se brinda una visión del panorama general de cómo este medio de comunicación influye notablemente en la sociedad actual. Además, también se dan las especificaciones técnicas necesarias para entender el formato de los emails (el protocolo MIME explicado en el apartado 2.1.1), cómo se envían (el protocolo SMTP quedado reflejado en la sección 2.1.2) y cómo se reciben (protocolos POP e IMAP que se presentan en las secciones 2.1.3 y 2.1.4, respectivamente).

En el capítulo 2, además de entrar en detalle acerca del correo electrónico, se ofrece los conceptos básicos necesario de la generación de lenguaje natural (apartado 2.2): en qué consiste esta rama de la inteligencia artificial y sus aplicaciones más comunes (sección 2.2.1), las principales arquitecturas más utilizadas en los problemas de NLG (apartado 2.2.2) y una breve introducción al ámbito del resumen abstractivo de textos (sección 2.2.3) del cual se utilizarán técnicas para abordar el problema que compete al proyecto.

A continuación, cuando ya se cuenta con los conocimientos requeridos para enfrentarse al problema, en el capítulo 3, se presenta todo el trabajo realizado. Primero se introducen brevemente las tecnologías que se manejan en la fase de desarrollo del proyecto (apartado 3.1): spaCy (sección 3.1.1), textaCy (apartado 3.1.2), tensorflow (punto 3.1.3) y MongoDB (sección 3.1.4). Se continúa reflejando todos los pasos que engloban el análisis de datos preliminar del corpus escogido (sección 3.2) y se finaliza explicando los problemas y detalles de las implementaciones de la solución utilizando la arquitectura realizer (apartado 3.3), transformer (punto 3.4) y los resultados obtenidos con el desarrollo (sección 3.5).

Por último, en el capítulo 4, se sacan a relucir las conclusiones que pueden extraerse de este estudio (apartado 4.1) y las distintas vías de trabajo abiertas que deja este proyecto sobre las que se podría continuar con él (sección 4.2).

Capítulo 2

Estado de la Cuestión

“Quien controla el presente controla el pasado y quien controla el pasado controlará el futuro.”
— 1984 - George Orwell (1949)

Como se ha explicado en el capítulo anterior, el objetivo de este trabajo es el de desarrollar un sistema de generación de lenguaje natural capaz de, dada una representación semántica de lo que se quiere transmitir, redacte un correo electrónico completo. El primer paso para conseguir esta meta es entender en qué consiste la estructura de un correo electrónico y cómo es su formato, ya que el corpus del que se parte consta de e-mails reales extraídos de un servidor de mensajes. Por ello, en la sección 2.1 se introducen los conceptos básicos así como datos acerca de la utilización actual de este método de comunicación que demuestran la utilidad de contar con una aplicación que sea capaz de generarlos automáticamente. En concreto, se entrará en detalle de los principales protocolos que describen el formato de los e-mails (protocolo MIME explicado en la sección 2.1.1), cómo se mandan entre servidores (protocolo SMTP que se presenta en el apartado 2.1.2) y cómo el usuario final los recibe y consulta (protocolos POP e IMAP desarrollados a lo largo de las secciones 2.1.3 y 2.1.4, respectivamente). A continuación, en el apartado 2.2, se profundiza en el ámbito de la inteligencia artificial conocido como Generación de Lenguaje Natural (NLG), explicando qué es la NLG, por qué es útil en este trabajo y sus aplicaciones más comunes (véase el apartado 2.2.1), las arquitecturas más utilizadas actualmente para desarrollar sistemas de NLG (consúltese la sección 2.2.2 en la que se explica el modelo *realizer* en el apartado 2.2.2.1 y la arquitectura *transformer* en el punto 2.2.2.2) y se finaliza con una breve introducción a las técnicas de resumen automático de textos (2.2.3) que serán de gran utilidad para el desarrollo de este estudio y que permiten abordar ciertos problemas que presenta la arquitectura *realizer* en los sistemas que no pueden enmarcarse en un dominio concreto.

2.1. Correo electrónico

El correo electrónico (Guide, 2005, Capítulo 11) es un servicio de comunicación que ha sido utilizado desde 1971 (Ibrahim et al., 2018), momento en el que a través de la primera red adaptada para el envío de e-mails se envió el texto “QWERTYUIOP”. Este correo se mandó a través de ARPAnet (cuyo nombre proviene de *Advanced Research Projects Agency Network*, que en inglés significa Red de la Agencia de Proyectos de Investigación Avanzada, y fue la primera red en la que se implementó el famoso protocolo TCP/IP) con

un protocolo experimental conocido como CYPNET. Actualmente los mensajes hacen uso de una arquitectura cliente-servidor, de manera que el correo electrónico es construido a través de un programa cliente y, posteriormente, es enviado al servidor. Desde dicho servidor, se redirige el mensaje al servidor del servicio de correo del destinatario y, desde este último, es enviado al receptor.

De acuerdo con Radicati y Levenstein (2021), el correo electrónico “sigue siendo la forma de comunicación dominante tanto para las empresas como para los consumidores particulares” y, aún hoy en día, cada año se continúa observando un constante crecimiento del número de cuentas de e-mail y de la cantidad de mensajes enviados. De hecho, en 2021 el número de usuarios de correo electrónico en todo el mundo alcanzará los 4.1 miles de millones (más de la mitad de la población mundial utiliza el servicio de e-mail) y se espera que esta cifra siga aumentando hasta que haya 4.5 miles de millones en 2025. El crecimiento del número de usuarios en este rango de años se ve reflejado en la tabla 2.1.

Año	2021	2022	2023	2024	2025
Miles de millones de usuarios en todo el mundo	4.147	4.258	4.371	4.481	4.594
Porcentaje de crecimiento	3 %	3 %	3 %	3 %	3 %

Tabla 2.1: Previsión de usuarios de correo electrónico en todo el mundo (2021-2025)
Tabla extraída de Radicati y Levenstein (2021).

Por otro lado, la evolución en el tráfico diario de e-mails en todo el mundo se presenta en la tabla 2.2, donde puede observarse la gigantesca cantidad de correos electrónicos enviados cada día y su crecimiento a lo largo de los próximos cuatro años. Con estos datos, podemos calcular el número medio de mensajes enviados por usuario cada día, obteniendo que en 2021 de media cada usuario manda aproximadamente 77 e-mails y esta cantidad continúa creciendo hasta alcanzar casi los 82 correos electrónicos diarios de media por usuario. Esto significa que, a medida que avanzan los años, no solo crece la cifra de personas que hace uso de este sistema de comunicación, sino que también aumenta la dedicación que cada usuario invierte en la utilización de esta herramienta.

Año	2021	2022	2023	2024	2025
Miles de millones de correos electrónicos enviados/recibidos al día en el mundo	319.6	333.2	347.3	361.6	376.4
Porcentaje de crecimiento	4.3 %	4.3 %	4.2 %	4.1 %	4.1 %

Tabla 2.2: Tráfico diario de correos electrónicos en todo el mundo (2021-2025)
Tabla extraída de Radicati y Levenstein (2021).

Para hacer posible el envío de todos estos correos electrónicos, existe un estándar que determina el formato que deben tener los mensajes y una amplia gama de protocolos de red que permiten el intercambio de e-mails entre máquinas distintas (las cuales a menudo poseen sistemas operativos distintos y utilizan diferentes programas de correo electrónico). A continuación se presenta dicho estándar de formato conocido como MIME (véase la sección 2.1.1), el cual resultará de gran utilidad de cara a procesar cada uno de los mensajes pertenecientes corpus inicial de partida (explicado en ??) y obtener la información necesaria de cada uno de ellos. También, con el fin de cerrar este apartado y tener un conocimiento general acerca de el funcionamiento de este medio de comunicación, se introducirán los principales protocolos de gestión de correos electrónicos tanto para la transmisión de los mismos (para dicha tarea se hace uso del protocolo SMTP expuesto en la sección 2.1.2)

como para el acceso por parte de los usuarios (en este caso se utilizan los protocolos POP e IMAP que son explicados en las secciones 2.1.3 y 2.1.4, respectivamente).

2.1.1. MIME

La especificación del formato que deben tener los correos electrónicos viene determinada por el estándar conocido como MIME (acrónimo de *Multipurpose Internet Mail Extensions*), el cual es utilizado para el intercambio de distintos tipos de archivos (texto, audio y vídeos, entre otros) que ofrece soporte a textos con caracteres no pertenecientes al formato ASCII, archivos adjuntos que no son de texto, mensajes con cuerpo con numerosas partes (conocidos como mensajes multiparte) e información de cabecera con caracteres no ASCII. Se encuentra definido en los documentos técnicos llamados *Request For Comments* (RFC) con identificadores: RFC 2045 (Freed y Borenstein, 1996b), RFC 2046 (Freed y Borenstein, 1996c), RFC 2047 (Moore, 1996), RFC 2049 (Freed y Borenstein, 1996a), RFC 2077 (Nelson y Parks, 1997), RFC 4288 (Freed y Klensin, 2005a) y RFC 4289 (Freed y Klensin, 2005b).

Prácticamente todos los correos electrónicos escritos por personas en Internet y una considerable proporción de estos mensajes generados automáticamente, se transmiten en formato MIME a través de SMTP (véase la sección 2.1.2). Los mensajes de correo electrónico de Internet están tan estrechamente relacionados con SMTP y MIME que suelen denominarse mensajes SMTP/MIME.

Los tipos de contenido englobados dentro del estándar MIME son de gran importancia también fuera del contexto de los correos electrónicos. Ejemplos de ello son algunos protocolos de red como el HTTP de la Web. Este protocolo requiere que los datos se transmitan en un contexto de mensaje de tipo e-mail, aunque los datos no sean un correo electrónico propiamente dicho.

Hoy en día, ningún programa de correo electrónico o navegador de Internet puede considerarse completo si no acepta MIME en sus distintas funcionalidades (formatos de texto y de archivo).

2.1.1.1. Nomenclatura de tipos

Como se ha mencionado anteriormente, MIME permite el intercambio de distintos tipos de archivos. Para lograrlo, este estándar utiliza una nomenclatura diferente para denotar a cada tipo. Los nombres utilizados siguen el formato “tipo/subtipo”, siendo tanto tipo como subtipo cadenas de caracteres. De esta manera, el tipo especificará la categoría general de los datos enviados y el subtipo determinará el tipo específico de la información mandada. Los valores que puede tomar tipo son los siguientes:

- *text*: informa de que el contenido es texto. Este tipo puede preceder a los subtipos *html*, *xml* y *plain*.
- *multipart*: indica que el mensaje contiene distintas partes (cada una de un tipo diferente) con datos independientes entre ellas. Puede anteceder a subtipos como *form-data* y *digest*.
- *message*: se utiliza para encapsular un mensaje existente, por ejemplo, cuando se quiere responder a un correo electrónico y añadir los mensajes anteriores. A este tipo le pueden seguir subtipos como *partial* y *rfc822*.
- *image*: especifica que el contenido se trata de una imagen. Le pueden suceder los subtipos *png*, *jpeg* y *gif*.

- *audio*: determina que el contenido se trata de un audio. Los subtipos *mp3* y *32kadpcm* son algunos ejemplos a los que puede anteceder este tipo.
- *video*: señala que el contenido se trata de un vídeo. Puede preceder a subtipos como *mpeg* y *avi*.
- *application*: denota a los datos de aplicación que pueden ser binarios. Algunos de sus subtipos correspondientes son *json* y *pdf*.
- *font*: significa que el contenido del mensaje es un archivo que define el formato de una fuente. Le pueden suceder subtipos como *woff* y *ttf*.

2.1.1.2. Cabeceras MIME

Cuando se codifica un correo electrónico siguiendo el estándar MIME, se estructura en diferentes cabeceras cuyo valor asociado nos dará información acerca del mensaje enviado. Las cabeceras más importantes son:

- *Content-Type*: el valor asociado a esta cabecera es el tipo y subtipo del mensaje con el mismo formato que se ha explicado anteriormente (véase la sección 2.1.1.1). Por ejemplo, si se observa la cabecera y el valor *Content-Type: text/plain*, indicará que el mensaje es un texto plano. El uso del tipo *multipart* hace posible la creación de correos electrónicos con partes y subpartes organizadas en una estructura arbórea, en la cual los nodos hoja pertenecen a cualquier tipo y el resto pueden tratarse de algún subtipo de *multipart* (Freed y Borenstein, 1992, Sección 7.2). Para poder entender mejor cómo se estructuran este tipo de e-mails, en la figura 2.1 se presenta un posible mensaje con una parte de texto plano, otra de texto HTML y una imagen. Para crear este correo electrónico, es necesario contar con un nodo raíz del tipo *multipart/mixed*. Además, como puede observarse en la figura 2.1, la utilización del tipo *multipart/alternative* permite la coexistencia en un mismo mensaje del cuerpo tanto en formato de texto plano como HTML. Sobra decir que, gracias a este formato de estructura en árbol de la cabecera *Content-Type*, es posible construir muchas otras variedades de mensajes (como adjuntar el mensaje original que ha sido reenviado utilizando *multipart/mixed* con una parte *text/plain* y otra *message/rfc822*).

Un detalle importante es el hecho de que si se comparan las figuras 2.1 y 2.2, se podrá observar que cada nodo de la estructura arbórea del correo electrónico (mostrada en la figura 2.1) es presentado en la figura 2.2 (que es como se reciben los e-mails realmente y como se encuentran almacenados en el corpus) habiendo recorrido el árbol en profundidad en preorden.

- *Content-Disposition*: esta cabecera indica la forma de presentación de la parte del mensaje a la que pertenece. Puede tener dos posibles valores: *inline* (que se utiliza cuando el contenido debe ser mostrado al mismo tiempo que el cuerpo del mensaje, como por ejemplo cuando se inserta una imagen en el texto y no como archivo adjunto) y *attachment* (este valor determinará que la parte del mensaje requerirá algún tipo de acción por parte del usuario para visualizar el contenido, como por ejemplo en el caso de adjuntar un archivo). Además, esta cabecera dispone de distintos campos que reflejan más información acerca del contenido, como puede ser el nombre del archivo y la fecha de creación o de modificación. A continuación se presenta un ejemplo extraído de Troost et al. (1997) de esta cabecera y los campos que pueden acompañarle:

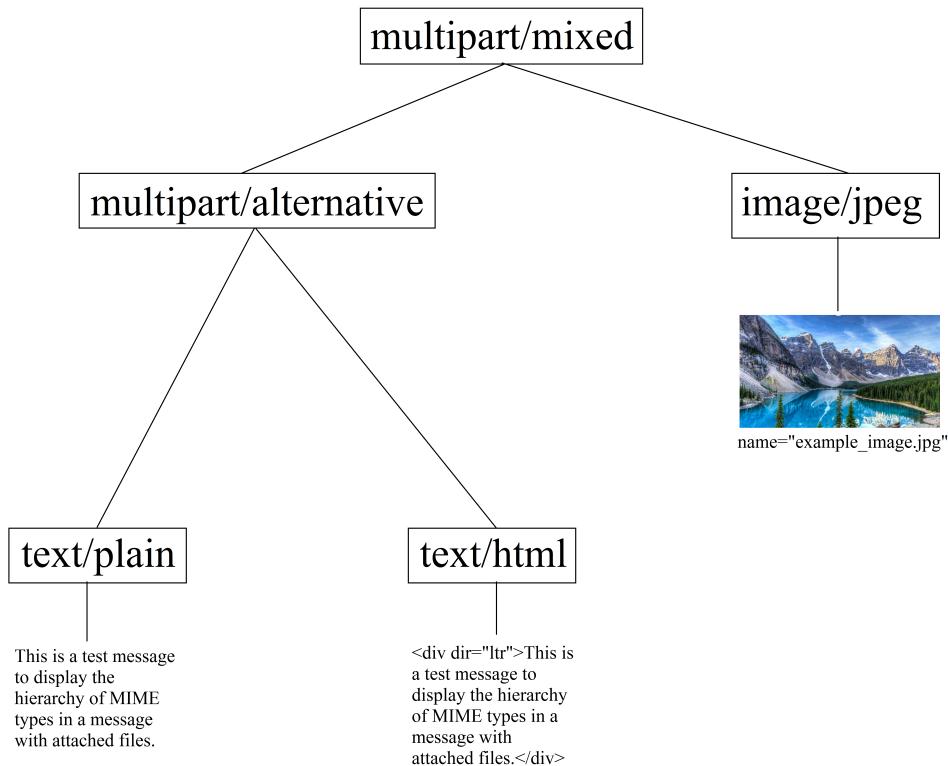


Figura 2.1: Estructura arbórea de tipos MIME de un e-mail de ejemplo
Imagen extraída de Moreno Morera (2020)

```
Content-Disposition: attachment; filename=genome.jpeg;
modification-date="Wed, 12 Feb 1997 16:29:51 -0500";
```

También puede observarse esta cabecera en la última parte del mensaje de ejemplo de la figura 2.2.

- *Content-Transfer-Encoding*: cuando se mandan archivos en un correo electrónico, a veces estos se codifican como 8-bit o archivos binarios, codificaciones no soportadas por determinados protocolos. Por este motivo, es necesario poseer un estándar que especifique cómo debe re-codificarse este tipo de información en un formato 7-bit. La cabecera *Content-Transfer-Encoding* (Freed y Borenstein, 1992, Sección 5) indica al cliente qué tipo de transformación se ha efectuado para que este sea capaz de recuperar los datos originales. Los posibles valores son *base64* (Josefsson, 2006; Freed y Borenstein, 1996b), *quoted-printable* (Borenstein y Freed, 1993), *8bit*, *7bit*, *binary* y *x-token*. Todos ellos hacen referencia a un tipo de codificación que se encuentran fuera del alcance de este trabajo y para las que se recomienda consultar las referencias bibliográficas en caso de querer profundizar en ellas.

2.1.2. SMTP

El SMTP (cuyas siglas hacen referencia a *Simple Mail Transfer Protocol*) es un protocolo de red orientado a conexión utilizado para el intercambio de correos electrónicos. Originalmente fue definido por Postel (1982) (para especificar cómo llevar a cabo el envío de mensajes) y Crocker (1982) (que presenta el formato que deben tener los e-mails). Ac-

```

MIME-Version: 1.0
From: Sender <sender@gmail.com>
Date: Fri, 4 Oct 2019 22:06:29 +0200
Message-ID: <CADkvQ12BgJJxjqApD6AcNcAhJJEZ5FBkPi-9Pw_XduXgN_3sQ@mail.gmail.com>
Subject: Example MIME message
To: addressee@gmail.com
Cc: addressee2@gmail.com
Content-Type: multipart/mixed; boundary="000000000000abadb805941b3c9d"

--000000000000abadb805941b3c9d
Content-Type: multipart/alternative; boundary="000000000000abadb605941b3c9b"

--000000000000abadb605941b3c9b
Content-Type: text/plain; charset="UTF-8"
Content-Transfer-Encoding: quoted-printable

This is a test message to display the hierarchy of MIME types in a message
with attached files.

--000000000000abadb605941b3c9b
Content-Type: text/html; charset="UTF-8"
Content-Transfer-Encoding: quoted-printable

<div dir="ltr">This is a test message to display the hierarchy of MIME types in a message with attached
files.</div>

--000000000000abadb605941b3c9b--
--000000000000abadb805941b3c9d
Content-Type: image/jpeg; name="example_image.jpg"
Content-Disposition: attachment; filename="example_image.jpg"
Content-Transfer-Encoding: base64
Content-ID: <16d985155126855b6e01>
X-Attachment-Id: 16d985155126855b6e01

--000000000000abadb805941b3c9d--

```

Figura 2.2: Mensaje MIME de la figura 2.1
Imagen extraída de Moreno Morera (2020)

tualmente, se deben consultar los RFC desarrollados por Klensin (2008) y Resnick (2008) que, respectivamente, sustituyen a los dos originales.

Al ser un protocolo de transferencia de mensajes, posee algunas limitaciones a la hora de recibir e-mails en el servidor de destino. Por ello, esta tarea se delega a otros protocolos como el POP (véase la sección 2.1.3) e IMAP (véase la sección 2.1.4), mientras que el SMTP se encarga única y exclusivamente del envío.

Cuando se hace uso del SMTP un correo electrónico es enviado (esta acción se denota con la palabra *push*) de un servidor a otro hasta que alcanza su destino. El mensaje se encamina en función del servidor de correo de destino, en lugar de hacerlo en función de los destinatarios individuales del mensaje especificados durante la conexión del cliente al servidor SMTP. Gracias a que este protocolo dispone de una función para iniciar el procesamiento de la cola de correo, un servidor de correo conectado de forma intermitente puede extraer mensajes de otro servidor remoto cuando sea necesario.

2.1.3. POP

El POP (cuyas siglas hacen referencia a *Post Office Protocol*) es un protocolo de aplicación en el modelo OSI utilizado para la obtención de e-mails almacenados en un servidor

remoto de Internet denominado servidor POP. Originalmente fue definido por Reynolds (1984), que especificó la primera versión de POP, también conocida como POP1. La versión actual de POP, POP3 (en general cuando se habla de POP se refiere a esta versión), fue detallada por Myers et al. (1996).

El protocolo POP posee numerosos comandos que hacen posible la conexión manual con el servidor POP3. Además, soporta otros como LIST, RETR y DELE, que permiten la gestión de los mensajes del usuario con acciones como mostrarlos, descargarlos o borrarlos, respectivamente.

POP3 fue diseñado para la tarea de recepción de correos electrónicos. Gracias a este protocolo, los usuarios con conexiones a Internet intermitentes o muy lentas pueden descargar sus mensajes mientras se encuentran conectados a la red y consultarlos estando *offline*. La sucesión de operaciones más común se produce cuando un cliente se conecta, descarga todos sus mensajes, los almacena en su dispositivo local como e-mails nuevos, se borran del servidor y, por último, el usuario se desconecta. Sin embargo, algunos clientes de correo incluyen la opción de dejar los mensajes almacenados en el servidor en lugar de borrarlos. Estos utilizan el comando UIDL (*Unique IDentification Listing*) el cual, a diferencia del resto de instrucciones de POP3, no identifica el correo electrónico a través del número ordinal asociado por el servidor, ya que generaría problemas si un cliente tratara de dejar ciertos mensajes en el servidor debido a que este número cambiaría de una conexión a otra. En lugar de ello, asigna a cada mensaje un identificador constituido por una cadena de caracteres única y permanente. De esta manera, se puede determinar fácilmente qué mensajes se quieren almacenar en el servidor a la vez que se descargan.

Al igual que otros protocolos más antiguos, POP3 utiliza un mecanismo de firma sin encriptación. De hecho, la transmisión de contraseñas POP3 en texto plano continúa ocurriendo. Hoy en día, POP3 tiene varios métodos de autenticación que ofrecen un amplio rango de niveles de protección contra accesos ilegales a las bandejas de correo de los usuarios.

La ventaja de POP3 frente a otros protocolos es que entre el cliente y el servidor no es necesario mandar un gran número de comandos para comunicarse. Este protocolo también resulta muy útil cuando no se cuenta con una conexión constante a Internet o a la red que aloja al servidor.

2.1.4. IMAP

El IMAP (cuyas siglas hacen referencia a *Internet Message Access Protocol*) es un protocolo de aplicación diseñado como alternativa al POP (véase la sección 2.1.3) en 1986, el cual permite el acceso a los mensajes almacenados en un servidor de Internet. Al igual que con el POP, con el IMAP es posible acceder a la cuenta de correo electrónico desde cualquier dispositivo con conexión a Internet. La versión actual del IMAP (IMAP versión 4 revisión 1 o IMAP4rev1) fue definida por Crispin (2003).

A diferencia del POP, el IMAP abre la puerta a la gestión de la misma bandeja de entrada por parte de múltiples clientes. Esta característica se produce gracias a las principales diferencias entre los dos protocolos: el IMAP no elimina los mensajes del servidor hasta que el cliente lo solicite explícitamente (mientras que el POP los borra por defecto, lo que hace imposible acceder a ellos desde otro dispositivo que no haya descargado previamente los correos electrónicos) y tampoco descarga los e-mails en el dispositivo del usuario, aunque opcionalmente es factible tener una copia local de los mismos. Esta última propiedad del IMAP da lugar a algunas ventajas frente al POP. Una de ellas es la posibilidad de notificar de manera inmediata de la llegada de un nuevo correo electrónico (ya que el IMAP fun-

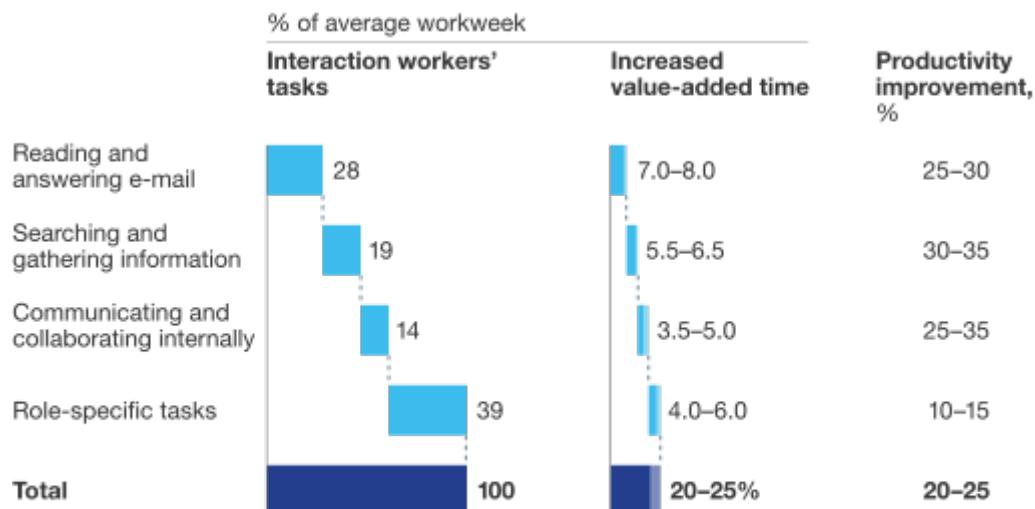
ciona con una conexión cliente-servidor permanente), mientras que el POP verifica si hay nuevos mensajes cada pocos minutos (lo cual provoca un aumento apreciable del tráfico y del tiempo que el usuario tiene que esperar para enviar una solicitud al servidor, ya que es necesario completar primero la descarga de todos los mensajes nuevos). Por otro lado, gracias al IMAP, los usuarios pueden crear carpetas compartidas con otras personas (esta funcionalidad depende del servidor de correo) y los e-mails no ocupan espacio de memoria en el dispositivo local, mientras que el POP los descarga independientemente de si van a ser leídos o no (estrictamente hablando, el IMAP tiene que descargar el mensaje cuando va a ser leído, pero se trata de archivos temporales y solo se extraen las cabeceras de los correos electrónicos a la hora de gestionar la bandeja de entrada). Precisamente, el hecho de evitar la descarga del e-mail, permite al usuario gestostrar carpetas, plantillas y borradores en el servidor además de poder llevar a cabo una búsqueda en la bandeja de entrada mediante palabras clave.

2.2. Generación de Lenguaje Natural

Como se ha descrito y observado gracias a la tabla 2.2, el tráfico de correos electrónicos diarios continúa en constante crecimiento hasta llegar, al menos, a la gigantesca cifra de 376 mil millones enviados al día en todo el mundo. Esto se revierte en una gran cantidad de tiempo invertido para la redacción de todos estos mensajes que no se mandan de manera automática. Sin embargo, esta gran dedicación al e-mail lleva produciéndose desde hace más de una década, cuando no nos encontrábamos con cifras de tráfico tan elevadas. Según Chui et al. (2012), de media los empleados invertían el 28 % de su tiempo semanal en la gestión del correo electrónico (como viene reflejado en la figura 2.3). Esto se traduce en más de once horas dedicadas únicamente a leer y contestar mensajes, enviando y recibiendo una media de 124 e-mails por día (Radicati y Levenstein, 2015). Por si estos datos no fueran suficientemente preocupantes, de cara a la productividad laboral y resolución eficiente de las tareas, según Segal (2021) este problema se ha agravado en los últimos años por diversas causas (entre las que se encuentra la pandemia de la Covid-19). En definitiva, hoy en día podemos afirmar que tanto en el ámbito profesional como personal se invierte una gran cantidad de esfuerzo y tiempo para gestionar nuestra cuenta de e-mail, lo cual plantea un problema en el que vemos que, en lugar de ser una herramienta útil, se convierte en una responsabilidad más que debe llevarse al día y de la que no es posible desprenderte ya que la capacidad de mandar estos mensajes es imprescindible para llevar a cabo tareas del día a día. Pero, ¿y si fuera posible ahorrar todo este tiempo de escritura de correos electrónicos?

Para lograr este propósito es imprescindible profundizar en la rama de la Inteligencia Artificial conocida como *Generación de Lenguaje Natural* (cuyas siglas son *NLG* por su nombre en inglés *Natural Language Generation*). Un buen ejemplo de aplicación de las técnicas de generación automática de textos son los 100.000 libros que Philip M. Parker puso a la venta en la plataforma *Amazon.com* incluyendo títulos de temáticas tan variadas como *El libro oficial del paciente sobre la estenosis espinal* (Parker, 2002), *Perspectivas mundiales de 2009 a 2014 de los envases de 60 miligramos de Fromage Frais* (Parker, 2008a), *Perspectivas de 2007 a 2012 de los tapetes de nudo, alfombras de baño y conjuntos que miden 6 pies por 9 pies o menos en la India* (Parker, 2006) y *Tesoro Quechua - Inglés* (Parker, 2008b).

Resulta evidente que dicha cantidad de libros no pudieron ser escritos por Parker, sino que debió hacerse uso de técnicas de generación automática de textos. El algoritmo utilizado para dicho propósito, se engloba dentro de los métodos de generación conocidos como *text-to-text* (texto a texto en castellano), dado que este tipo de técnicas toman como



Source: International Data Corporation (IDC); McKinsey Global Institute analysis

Figura 2.3: Porcentaje de tiempo de un trabajador dedicado a cada tarea

entrada textos ya existentes (normalmente escritos a mano y no generados automáticamente) y producen un nuevo texto coherente como salida. Otras aplicaciones de este tipo de métodos son la traducción automática de un idioma a otro (Hutchins y Somers, 2009; Oettinger, 2013), el resumen automático de textos (Mani y Maybury, 2001; Nenkova y McKeown, 2011), la simplificación de textos complejos, ya sea para hacerlos más accesibles para un público de lectores de bajo nivel de alfabetización (Siddharthan, 2014; Bautista et al., 2011) o niños (Macdonald y Siddharthan, 2016), corrección automática de ortografía, gramática y texto (Kukich, 1992; Ng et al., 2014), generación automática de revisiones de artículos científicos (Bartoli et al., 2016), generación de paráfrasis dada una frase de entrada (Bannard y Callison-Burch, 2005), generación automática de preguntas con fines didácticos y educativos (Brown et al., 2005), generación automática de relatos dada una descripción conceptual de la historia deseada (Gervás et al., 2004) o reescritura de textos (en concreto correos electrónicos) con estilo en función del destinatario (Moreno Morera, 2020).

Además de estos métodos text-to-text, existen los llamados *data-to-text* (datos a texto), en los cuales en lugar de recibir un texto como entrada, se genera el lenguaje a partir de datos. Estos pueden ser de todo tipo para dar lugar a informes o resúmenes como pueden ser de índole climatológica (Goldberg et al., 1994a; Ramos-Soto et al., 2014), financiera (Plachouras et al., 2016), ingenieril, como por ejemplo el trabajo desarrollado por Yu et al. (2007) para generar resúmenes de datos recopilados por sensores en turbinas de gas, sanitaria (Hüske-Kraus, 2003; Banaee et al., 2013), como la investigación llevada a cabo por Portet et al. (2009) para obtener informes textuales a partir de datos de cuidados intensivos neonatales, o, incluso, deportivos (Theune et al., 2001; Chen y Mooney, 2008). Además de informes o resúmenes, también se utilizan los métodos *data-to-text* para otros propósitos como la composición de discursos narrativos para relatos de varios personajes a partir de partidas de ajedrez (Gervás, 2014), redacción de periódicos electrónicos a partir de datos de sensores (Molina et al., 2011), generación de texto que aborda problemas medioambientales como el seguimiento de la fauna (Siddharthan et al., 2012; Ponnamperuma et al., 2013), la información medioambiental personalizada (Wanner et al., 2015) y la mejora del

compromiso de los ciudadanos científicos a través de los comentarios generados (Van der Wal et al., 2016) o producción de información interactiva sobre artefactos culturales (Stock et al., 2007), entre otros.

Debido a que el objetivo de este trabajo se centra en la generación de correos electrónicos a partir del asunto, exploraremos en detalle las técnicas de Generación de Lenguaje Natural y, en especial, los métodos text-to-text. Para profundizar en los algoritmos y arquitecturas empleados ante los problemas de tipo data-to-text, conviene consultar la investigación llevada a cabo por Gatt y Krahmer (2018), en la cual muestran el estado del arte de los trabajos realizados en este ámbito.

2.2.1. ¿Qué es la Generación de Lenguaje Natural?

Dado que tanto los sistemas text-to-text como data-to-text y todas sus aplicaciones mencionadas anteriormente pertenecen a la rama de Generación de Lenguaje Natural, esta no debe definirse en función de la entrada del sistema, sino en la salida. Según Reiter y Dale (2000) la NLG es la conceptualización del “campo de la inteligencia artificial y la lingüística computacional que se centra en los sistemas informáticos que son capaces de producir textos comprensibles en inglés u otra lengua humana. [...] Como área de investigación, la NLG presenta una perspectiva única ante problemas fundamentales de la inteligencia artificial, la ciencia cognitiva y la interacción. Estos incluyen cuestiones como por ejemplo cómo deben ser representados y cómo debe razonarse con la lingüística y el dominio del conocimiento, qué significa que un texto esté correctamente redactado y cómo es la mejor forma de comunicar información entre las computadoras y los usuarios.” Por lo tanto, la Generación de Lenguaje Natural se puede definir como el ámbito que engloba el estudio de la producción de lenguaje no artificial, así como el diseño e implementación de algoritmos y sistemas computacionales cuyo resultado debe ser un texto que imite la forma en que los humanos se comunican verbalmente (Vicente et al., 2015), ya sea oralmente o por escrito (del Socorro Bernardos, 2007). Es decir, independientemente de la entrada recibida, se precisa el significado de NLG a partir de la salida esperada por el problema planteado. Tanto es así, que, como hemos visto, la entrada del sistema puede variar excesivamente (McDonald, 1993): desde textos (que son precisamente los sistemas text-to-text) hasta datos de todo tipo como partidas de ajedrez (Gervás, 2014), pictogramas (González Álvarez y López Pulido, 2019) e, incluso, vídeos (Thomason et al., 2014). Sin embargo, autores como Dušek et al. (2020) acotan la definición de los sistemas de NLG estableciendo que la entrada deben ser representaciones semánticas, obviando así la primera tarea de la arquitectura propuesta por Reiter y Dale (2000) conocida como macro planificación o determinación del contenido (se explicará en la sección 2.2.2.1), que es precisamente el punto en el que se generan dichas representaciones semánticas.

Cabe destacar que, aunque desde un principio hayamos diferenciado entre métodos text-to-text y data-to-text, ni los límites entre las dos aproximaciones ni la pertenencia de algunas técnicas a ellas se encuentran claramente definidos. Un ejemplo de ello podemos encontrarlo en la generación automática de resúmenes de textos. En principio se caracterizaría claramente como un sistema text-to-text. No obstante, al hacer frente a este problema se han desarrollado soluciones con las conocidas técnicas abstractivas (Genest y Lapalme, 2011), que, como explican Hahn y Mani (2000), a diferencia de los métodos de extracción evitan recoger las frases completas y se limitan a tomar unidades semánticas. Este tipo de técnicas usadas, por ejemplo, en la obtención de opiniones de reseñas para la posterior generación de frases nuevas (Labbé y Portet, 2012), también provienen de problemas data-to-text. A la inversa, un sistema data-to-text puede hacer uso de técnicas que

principalmente son utilizadas en los casos de uso text-to-text (McIntyre y Lapata, 2009; Kondadadi et al., 2013). Por otro lado, podría parecer que los métodos de *deep learning* (Goodfellow et al., 2016) deben ser mayoritariamente utilizados en los problemas data-to-text utilizando el trabajo llevado a cabo por Mikolov et al. (2013a). Sin embargo, se han desarrollado extensamente esta clase de soluciones para la NLG con gran variedad de arquitecturas como las redes neuronales recurrentes (Cho et al., 2014; Tang et al., 2016), o muy a menudo combinadas con la memoria a corto plazo o LSTM (Chen et al., 2016), o las arquitecturas conocidas como *transformers* (Vaswani et al., 2017).

2.2.2. Arquitecturas para la Generación de Lenguaje Natural

Ante el problema de generación de lenguaje natural, se han propuesta diversas soluciones para abordarlo. Sin embargo, actualmente preponderan dos tipos de arquitecturas: la propuesta por Reiter y Dale (2000), también conocida como arquitectura *realizer* (toma el nombre de una de sus fases), que divide la generación en distintas subáreas y las aborda por separado, y la presentada por Vaswani et al. (2017), también conocida como arquitectura *transformer*, que expone una arquitectura constituida mayoritariamente por redes neuronales. Aunque en este trabajo hayamos hecho uso de esta última debido a la dificultad de establecer un dominio de lenguaje en los correos electrónicos, a continuación se hará una breve introducción a ambas.

2.2.2.1. Arquitectura realizer

A pesar de que en el trabajo de Reiter y Dale (2000) se centren principalmente en los sistemas de generación de lenguaje natural de tipo *data-to-text*, esta arquitectura también ha sido utilizada por sistemas *text-to-text* como para la generación automática de resúmenes a través de métodos abstractivos (Genest y Lapalme, 2011). De hecho, incluso es posible hacer uso de la conceptualización de la entrada del sistema planteada por Reiter y Dale (2000) para este tipo de problemas.

La filosofía de esta arquitectura se centra en la modularización de las diferentes tareas a abordar durante la generación de lenguaje natural. De esta manera, cada módulo se enfrenta a un reto específico con el que debe lidiar y se conecta con el módulo anterior haciendo coincidir la salida del previo con la entrada del actual y, lo mismo, con el módulo posterior. Asimismo, se construye un *pipeline* o arquitectura en secuencia de tareas con cada uno de los módulos.

La entrada de esta arquitectura viene determinada por una tupla de cuatro elementos (k, c, u, d) donde cada uno de ellos puede representarse de distintas maneras. El primer elemento de la tupla es la *base de conocimiento* (la letra viene dada por su denominación ingles *knowledge source*). Se trata de la información acerca del dominio de nuestro sistema de generación de lenguaje natural, la cual suele consistir en un conjunto de bases de datos y bases de conocimiento, como las ontologías (Fensel, 2001), que nuestra aplicación puede consultar durante su ejecución. Tanto la representación como el contenido de la base de conocimiento son altamente dependientes del tipo de aplicación que queramos construir, por ejemplo, en el trabajo de Reiter et al. (2005) la base de conocimiento consiste en parámetros meteorológicos numéricos de un modelo de predicción NWP, mientras que en el desarrollo presentado por Reiter et al. (1995) se utiliza como base de conocimiento un sistema de representación de conocimiento del mismo tipo que KL-ONE (Brachman y Schmolze, 1989) estructurado de manera jerárquica mediante relaciones *is-a* y *part-of*, del cual se pueden extraer diversas propiedades de las entidades que lo componen. Precisamente esta gran variabilidad en tan solo la primera componente de la entrada de la arquitectura

realizer, es lo que sustenta la afirmación de Reiter y Dale (2000) de que no es posible proporcionar una caracterización formal genérica de lo que es una base de conocimiento y dificulta el establecimiento de esta componente en el sistema desarrollado si se hubiera elegido esta opción de arquitectura, ya que los correos electrónicos versan de una amplia variedad de temáticas muy distintas.

La segunda componente de la tupla de entrada es el objetivo de la comunicación (en inglés *communicative goal*). Este describe el propósito del texto para el que se quiere generar. Es importante no confundirlo con el propósito general del sistema de generación de lenguaje natural. Por ejemplo, el objetivo final de sistema implementado por Turner et al. (2007) es generar resúmenes textuales de datos de predicción meteorológica numérica espacio-temporal. Sin embargo, su propósito comunicativo de una ejecución determinada es el de presentar predicciones meteorológicas de una localización geográfica y un momento temporal dados.

La letra u de la tupla se corresponde con el modelo de usuario (en inglés *user model*), el cual consiste en una caracterización del receptor o público objetivo al que va dirigido el texto generado. Al igual que con el propósito comunicativo, no debe confundirse con los espectadores del sistema. Por ejemplo, el sistema implementado por Reiter et al. (1999) tiene como objetivo dirigirse a personas que consumen tabaco, ya que trata de generar cartas que convengan a los pacientes con este hábito para que intenten superar su adicción. No obstante, no resulta adecuado utilizar el mismo tipo de técnicas de convicción y el mismo lenguaje para una persona que ha comenzado a fumar desde hace poco que a otra que lleva muchos años consumiendo tabaco. También, podría resultar interesante definir perfiles en función de la edad y otras características del paciente que permitirían personalizar aún más estas cartas. Son precisamente este tipo de propiedades las que englobaría el modelo de usuario que el sistema toma como entrada. No obstante, a pesar de que no existen demasiados ejemplos de trabajos que incluyan esta variable de entrada como lo desarrollan en su estudio Goldberg et al. (1994b), por la dificultad de variación de los textos en función de dichos perfiles, este problema se suele tratar de abordar a través del estudio de la estilometría (Moreno Morera, 2020).

El último componente de la entrada es la historia del discurso (en inglés *discourse history*), la cual consiste en un modelo de la información transmitida y los temas tratados en el texto producido hasta el momento de la ejecución del sistema. Esto permite a la aplicación conocer las entidades y propiedades ya mencionadas gracias a las cuales es posible hacer un uso adecuado de los recursos anafóricos como los pronombres. En los sistemas de generación de lenguaje natural de interacción única, es decir, aquellos cuya ejecución produce un único texto con independencia de los generados en ejecuciones previas, la historia discursiva comienza como una estructura de datos vacía y se construye y utiliza durante la redacción del texto a generar. Todo lo contrario son los sistemas de diálogo, como los *chatbots*, en los cuales la historia del discurso suele hacer referencia al registro de diálogo, utilizado como repositorio de información sobre las interacciones previas entre el usuario y la aplicación de NLG. En trabajos como el desarrollado por Milosavljevic et al. (1996), en el cual se espera que los usuarios interactúen con una serie de textos relacionados, la historia discursiva resulta ser una mezcla de la utilizada en los sistemas de interacción única con la preponderante en los sistemas de diálogo. De esta manera, esta última componente de la tupla de entrada facilita el hacer referencias en el nuevo texto a entidades o conceptos mencionados en el texto actual o previos, o utilizar marcadores discursivos como “como se ha mencionado anteriormente” en momentos en que el texto generado repite información ya presentada en los anteriores.

Aunque puede resultar obvio, es importante tener en cuenta que, en la mayoría de los

casos, será necesario un preprocesado del texto o los datos de entrada (sobre todo si han de ser analizados e interpretados) para facilitar el trabajo a la arquitectura presentada por Reiter y Dale (2000), de manera que sea más sencillo utilizar la entrada en cada una de las fases (Han et al., 2011). Este primer preprocesado es plenamente opcional y depende del origen de la entrada que se vaya a utilizar.

Una vez se conoce la entrada y salida establecida para un sistema de generación de lenguaje natural, se puede comenzar la presentación de la arquitectura modular secuencial. Según Reiter y Dale (2000), el proceso de generación puede descomponerse en tres fases: la macro planificación, la micro planificación y la realización (que da nombre a la arquitectura por ser la última fase). Los módulos que implementan cada una de ellas se conectan entre sí como muestra la figura 2.4.



Figura 2.4: Arquitectura modular secuencial propuesta por Reiter y Dale (2000) para la NLG

En términos generales, el trabajo del módulo de macro planificación es producir una especificación del contenido del texto y su estructura, mediante el uso del dominio y el conocimiento de la aplicación sobre qué información es la más adecuada teniendo en cuenta la tupla de entrada. Esta fase normalmente también requiere conocer cómo suelen estructurarse los documentos del dominio de nuestro sistema. Muchas de las técnicas empleadas para la implementación de la macro planificación suelen asemejarse a las utilizadas en el ámbito de los sistemas expertos.

Para que el texto sea coherente, es preciso estructurar el contenido del mismo en el orden correcto. Por este motivo, la salida del módulo de macro planificación, el plan del documento, suele implementarse como una estructura de datos, generalmente arbórea, donde en cada nodo se encapsula la información más importante que debe formar parte de un párrafo o frase, además de información de cómo se relaciona con el resto de nodos. Con el objetivo de generar esta salida, se dividen las tareas de este nodo en determinación del contenido y estructuración del documento.

La determinación del contenido es la primera tarea de la macro planificación y, por tanto, del proceso de generación. En ella el sistema decide qué información es relevante para ser incluida en el texto y cuál no. Por lo general, en los sistemas data-to-text, se puede extraer más información en los datos de la que se quiere transmitir y, por ese motivo, cobra importancia la capacidad de selección y existen numerosas investigaciones al respecto, como la de Yu et al. (2007). Aunque la determinación del contenido está presente en la mayoría de los sistemas de generación de lenguaje natural (Mellish et al., 2006), los enfoques suelen estar estrechamente relacionados con el dominio, de forma que se construye una estructura de datos ad hoc, cuyos elementos se denominan “mensajes” o también son llamadas “representaciones semánticas” por autores como Dušek et al. (2020). Dicha estructura de datos se crea con el fin de especificar la salida de esta tarea (un ejemplo conceptual de un mensaje para un sistema que genera textos sobre la situación meteorológica puede observarse en la figura 2.5). Este planteamiento de la determinación del contenido, dificulta el desarrollo de sistemas en los que el dominio es extenso, como es el caso de los correos electrónicos, ya que no es posible implementar una clase que englobe todas las casuísticas (como se verá en la sección 2.2.3, para resolver este problema se utilizará el concepto de *Information Items* el cual pertenece al ámbito del resumen automático de textos). Sin embargo, sí se ha tratado

de encontrar otro tipo de definición a la salida de esta tarea, como ocurre en el trabajo de Guhe (2007), el cual presenta una explicación cognitivamente plausible e incremental de la determinación del contenido, basada en estudios sobre las descripciones de observadores de eventos dinámicos a medida que se desarrollan.

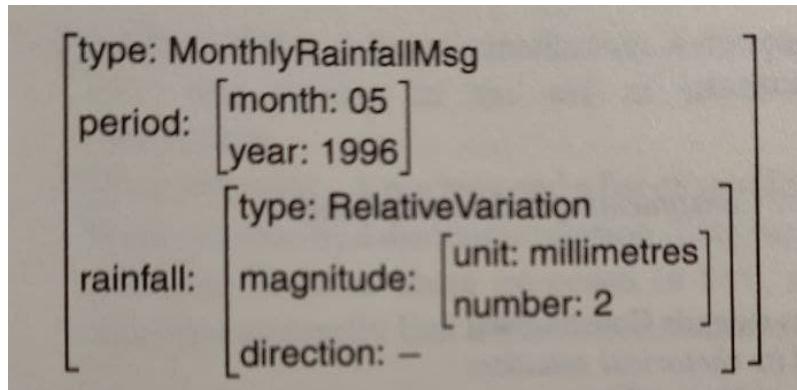


Figura 2.5: Ejemplo de mensaje
Imagen extraída de Reiter y Dale (2000)

La elección del contenido que debe ser expresado en el texto depende de varios factores, entre los que se encuentran: el propósito comunicativo, el modelo de usuario, las restricciones de la salida por limitaciones del dominio (como la longitud máxima que debe tener un texto) y la fuente de información (o base de conocimiento) subyacente disponible (es importante, aunque parezca evidente, no pretender generar información sobre la que el sistema no posee conocimiento o no es capaz de deducirlo con sus módulos de razonamiento correspondientes).

Como se ha mencionado anteriormente, además de la determinación del contenido, el módulo de macro planificación se encarga de abordar la tarea de estructuración del documento. Esta engloba todas las decisiones relativas a cómo cada parte del contenido debe ser agrupada en el texto y cómo debe ser relatada en términos retóricos. La estructura subyacente de un texto puede ser vista de manera jerárquica con los elementos textuales como cláusulas y las frases como constituyentes de largos fragmentos de texto, que son a su vez elementos de otros fragmentos más largos. Un texto puede ser analizado como una estructura arbórea cuyos elementos vienen relacionados por el tema que tratan (todas las frases sobre una misma temática podrían situarse en el mismo párrafo) y las relaciones discursivas (por ejemplo, es posible pasar de hablar en términos generales a entrar en detalles). Para abordar esta tarea la mayoría de trabajos, como el de Williams y Reiter (2008), suelen hacer uso de la llamada teoría de la estructura retórica (Mann y Thompson, 1987), cuyas siglas son RST por su nombre en inglés *Rhetorical Structure Theory*. La idea básica subyacente de la RST es que la coherencia y cohesión vienen dadas en virtud de las relaciones que mantienen los elementos del texto. También se pueden encontrar investigaciones, como es el caso de la realizada por McKeown (1992), que lo resuelven mediante esquemas, es decir, patrones que determinan cómo se debe construir un documento a partir de elementos constituyentes que pueden ser mensajes o instancias de otros esquemas. Además, permiten incluir elementos opcionales que solo se incluirían cuando se cumplen ciertas condiciones de activación. Un ejemplo de implementación de los esquemas son las gramáticas independientes del contexto (Cremers y Ginsburg, 1975).

En definitiva, el primer módulo de la arquitectura realizer es el de macro planificación, el cual construye primero los mensajes (seleccionando los datos, resumiéndolos si fuera

necesario, razonando sobre ellos y adaptándolos en función de la tupla de entrada) y, a continuación, define una relación entre todos ellos (normalmente jerárquica), generando así lo que se conoce como plan del documento.

El plan del documento es la entrada del módulo de micro planificación, que es el encargado de las tareas de agregación (decidir cómo el plan del documento se convertirá en estructuras lingüísticas como frases y párrafos, y el orden de las mismas si esto no se ha resuelto en la tarea de estructuración del documento), lexicalización (decidir qué recursos lingüísticos, como palabras o construcciones sintácticas, serán utilizados) y generación de expresiones referenciales (decidir qué expresiones serán utilizadas para las distintas entidades). Por ejemplo, para un sistema de generación de reportes meteorológicos, el plan de documento podría especificar que en el texto debería incluirse el hecho de que en febrero de 2020 se observaron temperaturas por debajo de la media. No obstante, la decisión de como referirse a febrero de 2020 (existen numerosas opciones como “el mes”, “febrero”, “febrero de 2020”, “el segundo mes del año”, etcétera) es tarea del módulo de micro planificación. O también la salida de la macro planificación puede añadir, además de la bajada media de temperaturas, un descenso en el mismo mes en la media de precipitaciones. Será entonces el módulo de micro planificación el encargado de determinar si estos dos hechos deben expresarse en frases separadas o combinados en una misma utilizando, quizás, la estructura de una oración coordinada copulativa.

Resulta razonable pensar que no cada mensaje generado en el plan del documento necesita ser expresado en una frase separado del resto. El hecho de combinar varios mensajes en una misma oración probablemente produzca un texto más fluido y legible (Dalianis, 1999; Cheng y Mellish, 2000). Precisamente esta decisión es la que se debe tomar en la fase de agregación, junto con la transformación de los mensajes en estructuras lingüísticas y elementos textuales más amplios como pueden ser los párrafos. Como explican Reiter y Dale (2000), esta tarea no necesariamente precede a la de lexicalización, sino que suelen implementarse de manera que se ejecuten ambas en paralelo interactuando una con la otra, ya que suelen estar fuertemente relacionadas la elección de los términos a utilizar (lexicalización) y las estructuras lingüísticas empleadas.

En cuanto a la definición técnica de la agregación, esta tarea se ha interpretado de distintas maneras, desde la eliminación de redundancias hasta la combinación de las estructuras lingüísticas. Tanto es así, que Reape y Mellish (1999) hacen una distinción entre agregación a un nivel semántico y a nivel sintáctico. Esta fase se ha desarrollado ampliamente centrándose en las restricciones y reglas específicas del dominio de la aplicación, como es el caso de las investigaciones de Hovy (1987) y Shaw (1998), aunque también se han implementado aplicaciones utilizando métodos dirigidos por corpus de ejemplos en los que las reglas se adquieren a partir de estos datos (Walker et al., 2001; Stent et al., 2004).

La lexicalización, tarea del módulo de micro planificación, consiste en la elección de las distintas palabras (nombres, verbos, adjetivos y adverbios) que se requieren para expresar los distintos mensajes del plan del documento. La complejidad de este proceso radica en la cantidad de alternativas que el sistema de generación de lenguaje natural puede contemplar. A menudo, las restricciones contextuales (como intentar evitar la repetición de ideas o entidades) pueden ser de gran ayuda. Si el objetivo es generar textos con cierta variación (Theune et al., 2001), una solución puede ser elegir de manera aleatoria la opción de lexicalización. Sin embargo, puede ocurrir que aparezcan restricciones estilísticas o haya que tener en cuenta otras consideraciones como la actitud o postura afectiva acerca de lo que se está redactando (Fleischman y Hovy, 2002, Sección 5). Este tipo de limitaciones dependerán del dominio de la aplicación.

Por otro lado, la lexicalización podría complicarse por dos razones (Bangalore y Ram-

bow, 2000): puede suponer la elección entre palabras de semántica similar, sinónimo o taxonómicamente relacionadas (Stede, 2000; Edmonds y Hirst, 2002), y no siempre es sencillo modelar la lexicalización en términos de un mapeo nítido entre conceptos y palabras. Una fuente de dificultad es la imprecisión e inexactitud del lenguaje, que surge, por ejemplo, con términos que denotan propiedades que son graduables. Un buen ejemplo de ello es al seleccionar adjetivos como ancho o alto basado en las dimensiones de una entidad. Esto requiere que el sistema razone sobre la anchura o altura de objetos similares, pudiendo utilizar un estándar de comparación (Van Deemter, 2012; Kennedy y McNally, 2004).

La otra tarea de la que se encarga el módulo de micro planificación es la generación de expresiones referenciales, la cual consiste en redactar expresiones que se refieran a las distintas entidades del texto que permitan al receptor identificar dicha entidad dado el contexto. Resulta un reto en la generación de lenguaje natural ya que hay distintas maneras de generar estas referencias, tanto la primera vez que se menciona la entidad (referencia inicial) como cuando esta es nombrada después de haber sido introducida en el discurso (referencia subsiguiente). Esta descripción de la generación de expresiones referenciales, puede parecer que guarda varias similitudes con la lexicalización, pero Reiter y Dale (2000) indican que la diferencia esencial entre ambas tareas es que, la que ahora ataÑe al texto, es “una tarea de discriminación en la cual el sistema necesita comunicar suficiente información de cara a distinguir una entidad del dominio frente al resto de entidades” evitando la ambigüedad. Esta fase de la micro planificación ha sido muy llamativa para varios investigadores como Siddharthan et al. (2011), quienes han centrado sus esfuerzos en presentar diversas soluciones que permitan la implementación de esta tarea. La primera aproximación que se nos puede ocurrir es la llamada forma referencial, en la cual las entidades son descritas mediante pronombres, nombres propios o una descripción concreta. Esta elección depende, en cierta forma, en la medida en que la entidad se encuentre en el foco de atención de la estructura lingüística o se quiera destacar (Poesio et al., 2004). De hecho, tales nociones subyacen a muchas investigaciones acerca de la generación de pronombres (McCoy y Strube, 1999; Callaway y Lester, 2002; Kibble y Power, 2004).

Precisamente elegir la forma referencial ha sido objeto de diversos trabajos, como el de Belz et al. (2009), sobre la generación de expresiones referenciales aprovechando el contexto que engloba lo que se quiere redactar, y la investigación de Ferreira et al. (2017) que pone de manifiesto la problemática en la generación de nombres propios para las entidades.

Como puede observarse en la figura 2.4, la salida del módulo de micro planificación es la especificación del texto. Se trata de un objeto que proporciona una definición completa y precisa del documento a generar. Análogo al plan de documento, la especificación del texto suele implementarse con una estructura arbórea en la cual las hojas son los objetos que Reiter y Dale (2000) llaman especificación de oración (las cuales pueden conceptualizarse de distintas formas: fragmento de texto enlatado, estructura sintáctica abstracta o un frame de casos lexicalizados), mientras que los nodos internos suelen contener la información acerca de la estructura del documento en términos de párrafos, secciones, etcétera. La estructura que describe la especificación del texto es la estructura lógica del documento, no la física (cómo el material es distribuido a lo largo de las páginas, columnas, líneas), la cual es tarea del último módulo de la arquitectura *realizer*. En esta salida del módulo de micro planificación, no se suelen encontrar relaciones discursivas, los nodos internos normalmente no las especifican. Si fuera necesario comunicarlas, el módulo de micro planificación llevaría a cabo los cambios convenientes a nivel de especificación de oración. Otro dato importante de esta estructura de datos es que el orden de los constituyentes (los nodos hijos) está determinado por cómo se recorrerá el árbol de la especificación del texto.

Por último, el módulo de realización (por el cual se le da nombre a la arquitectura

y es el que producirá el texto final) llevará a cabo las tareas de realización lingüística y realización estructural. La primera consiste en recibir las representaciones abstractas de las frases (especificación de oración), producidas en la micro planificación, y convertirlas en texto. Al igual que los documentos no son secuencias de oraciones ordenadas de forma aleatoria, las frases no son secuencias de palabras ordenadas al azar. Cada idioma se define, al menos en parte, por una serie de reglas que determinan cuándo una sentencia está bien construida. Dichas reglas son dirigidas por el campo de la morfología y la sintaxis. La realización lingüística suele definirse como el problema de aplicar reglas gramaticales a representaciones abstractas con el fin de producir un texto correcto a nivel sintáctico y morfológico. La complejidad de esta tarea dependerá entonces de cuán distinta sea la representación abstracta de su redacción final.

Para resolver el problema de realización lingüística, en el cual se debe ordenar y especificar los distintos constituyentes de una frase así como generar su correcta forma morfológica (incluyendo conjugación de verbos y concordancia de persona, género y número de las distintas entidades lingüísticas); se han propuesto distintas aproximaciones: plantillas preconstruidas, sistemas basados en la gramática preconstruidos y métodos estadísticos. La primera aproximación suele resultar de utilidad cuando el dominio de la aplicación es pequeño y la variación esperada es ínfima (McRoy et al., 2003). Consiste en oraciones construidas previamente en las cuales solo es necesario llenar ciertas entidades y ajustar algunas características morfológicas. La ventaja de las plantillas es que se tiene un control absoluto sobre la calidad de la salida y evita la generación de estructuras gramaticalmente incorrectas. Además, pueden utilizarse complejos sistemas de reglas para llenar estos “huecos” haciendo que las plantillas sean difíciles de distinguir de otros métodos más sofisticados (Deemter et al., 2005). Las desventajas de esta aproximación son que se caracterizan por resultar excesivamente laboriosas si se construyen a mano, aunque se han desarrollado investigaciones para su generación automática a partir de corpus como la de Angeli et al. (2012) y la de Kondadadi et al. (2013); y que no son una solución escalable para aplicaciones que requieran variaciones lingüísticas considerables.

Una alternativa a las plantillas son los sistemas en la gramática. Dichas gramáticas pueden ser construidas manualmente (Elhadad y Robin, 1996) o mediante inferencia (Parekh et al., 2000) a través de corpus. La mayor dificultad de los sistemas basados en gramática es cómo tomar decisiones entre varias opciones válidas y relacionadas entre sí. La otra alternativa son los métodos estadísticos, en los que se pueden encontrar el aprovechamiento de técnicas como la gramática categórica combinatoria (Steedman, 2000), como en el trabajo de Espinosa et al. (2008), el método de construcción de árbol adyacente a la gramática (Gardent y Narayan, 2015) y todo tipo de técnicas de aprendizaje automático como las máquinas de soporte de vectores (Bohnet et al., 2010), entre otros.

La última tarea es la denominada realización estructural. En ella se espera convertir las estructuras abstractas, como párrafos y secciones, en símbolos de marcado entendidos por el componente de presentación del documento (como por ejemplo las directivas HTML si se va a presentar en formato web). Algunos autores como Gatt y Krahmer (2018), no consideran esta fase como parte de la arquitectura *realizer*. En cualquier caso, la salida del módulo de realización será el texto final generado.

En resumen, la arquitectura *realizer* consta de tres módulos: macro planificación, micro planificación y realización. El primero de ellos recibe una entrada en forma de tupla que incluye la base de conocimiento, el propósito comunicativo, el modelo de usuario y la historia del discurso. Con estos cuatro elementos, lleva a cabo las tareas de determinación del contenido y estructuración del documento, generando el plan del documento, que será la entra del módulo de micro planificación. Este último será el encargado de la agregación,

lexicalización y la generación de expresiones referenciales, produciendo lo que se conoce como especificación del texto. Dicha entidad será recibida por el módulo de realización y ejecutará la realización lingüística y la realización estructural (cuando sea necesario), generando así el texto de lenguaje natural que se espera del sistema.

2.2.2.2. Arquitectura transformer

Como se ha señalado anteriormente, una de las grandes desventajas que presenta la arquitectura *realizer*, es su gigantesca dependencia del dominio del discurso en muchas de sus componentes. De hecho, la solución propuesta para implementar la determinación del contenido es una estructura de datos construida ad hoc, y esta es solo la primera fase de todo el pipeline. Esto no supone un problema cuando se plantea el desarrollo de sistemas de generación de lenguaje natural que pueden enmarcarse dentro de un ámbito específico (como los informes meteorológicos, las locuciones deportivas, etcétera). Sin embargo, cuando se busca generar correos electrónicos, el usuario podría plantearse escribir un email sobre diversas temáticas muy variadas que resulta complicado restringir dentro de un campo semántico. Este tipo de problemas, dada su complejidad, suelen abordarse utilizando arquitecturas compuestas por redes neuronales (Goldberg, 2016), normalmente de gran cantidad de capas ocultas (*deep learning* o aprendizaje profundo).

La aplicación de las arquitecturas de redes neuronales en la NLG se remontan, al menos, al trabajo desarrollado por Kukich (1987). Desde entonces el interés por este tipo de soluciones comienza a despertar en la comunidad de inteligencia artificial y, concretamente, en el ámbito del lenguaje natural (Elman, 1990, 1993; Chang et al., 2006). Las redes neuronales están diseñadas para aprender representaciones a niveles crecientes de abstracción explotando la propagación hacia atrás (LeCun et al., 2015), también denotada por su terminología inglesa *backpropagation*. Estas arquitecturas son densas, de baja dimensión y distorsionadas, lo que las hace ideales para captar generalizaciones gramaticales y semánticas (Mikolov et al., 2013b; Luong et al., 2013; Pennington et al., 2014). Además han cosechado notables éxitos en la modelización secuencial utilizando redes *feedforward* (Bengio et al., 2003; Schwenk y Gauvain, 2005), modelos logarítmicos bilineales (Mnih y Hinton, 2007) y redes neuronales recurrentes (Mikolov et al., 2010), incluyendo las RNN (redes neuronales recurrentes) a las que se les dota de unidades de memoria a corto plazo o LSTM (Hochreiter y Schmidhuber, 1997). Antes de que aparecieran en el panorama científico las arquitecturas transformers, estas últimas dominaban las soluciones de generación de lenguaje natural con redes neuronales, ya que su principal ventaja frente a otros modelos de lenguaje estándar es que compaginan la capacidad de hacer frente a secuencias de distintas longitudes y la de evitar tanto la escasez de datos como la explosión del número de parámetros. Precisamente una demostración del potencial de las redes neuronales recurrentes dotadas de unidades de memoria a corto plazo fue desarrollada en el trabajo de Sutskever et al. (2011).

El tipo de arquitectura que dio paso a los transformers, fue la conocida como *Encoder-Decoder* (Sutskever et al., 2014), codificador-decodificador en castellano, cuya filosofía se basa en utilizar una red neuronal recurrente para codificar la entrada en un vector suficientemente representativo que será la entrada del decodificador, también implementado con una RNN. Esta distinción entre codificador y decodificador posibilita compartir el mismo vector de codificación entre diversas tareas de procesamiento de lenguaje natural (Dong et al., 2015; Luong et al., 2015). Además, facilita la adaptación del codificador con el fin de generar texto a partir de representaciones semánticas abstractas, al igual que se ha hecho en este trabajo tomando como referencia la investigación realizada por Ferreira et

al. (2017).

A la arquitectura Encoder-Decoder hay que añadirle mecanismos de atención antes de llegar a la arquitectura transformer. Estos fuerzan al codificador, durante el entrenamiento, a darle más importancia (atribuyendo valores mayores en los pesos) a ciertas partes de la entrada del mismo de cara a la generación de determinados fragmentos de la salida en la decodificación (Tang et al., 2016; Xu et al., 2015). Esta técnica evita la necesidad de conectar neuronas de capas ocultas con la entrada, ya que los modelos de atención son capaces de aprender la correspondencia entre entrada y salida basado en fragmentos sueltos que representen a la entrada y textos de salida (Dušek y Jurčíček, 2016).

Es precisamente en el paradigma Encoder-Decoder y los mecanismos de atención en lo que se basa la conocida arquitectura transformer propuesta por Vaswani et al. (2017), como puede observarse en la figura 2.6. El rectángulo gris izquierdo de la imagen se corresponde con el codificador y el derecho con el decodificador. Dentro de ambos se pueden identificar módulos de atención. Expliquemos en detalle en qué consiste este modelo.

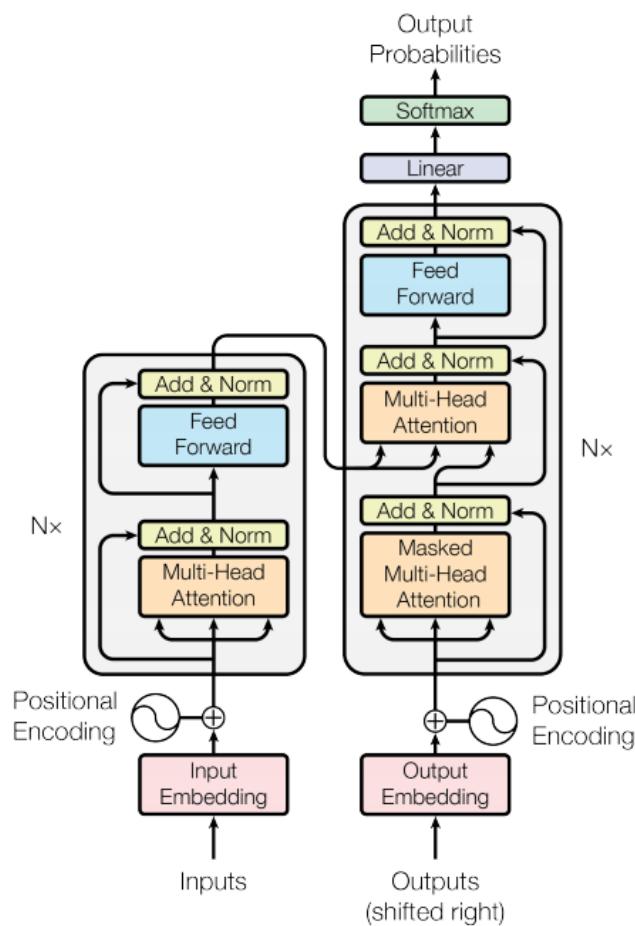


Figura 2.6: Modelo de arquitectura transformer
Imagen extraída de Vaswani et al. (2017)

Como se ha explicado anteriormente, la entrada de la arquitectura transformer (que puede ser un texto que debe completar añadiendo al final la composición lingüística correspondiente, un documento del que se espera obtener un resumen, un texto que quiere

traducirse, ...) será responsabilidad del codificador (parte izquierda de la imagen). Tras haber pasado por este módulo, se enviará al decodificador, el cual, como puede observarse en la figura, también tiene otra entrada distinta. Además de recibir la salida del codificador, el decodificador será informado de toda la secuencia de texto generada anteriormente, de manera que durante el entrenamiento, se evaluará su rendimiento palabra a palabra de cada uno de los textos que se espera que produzca y se llevará a cabo el backpropagation en función del token que debe dar como resultado. En la primera iteración de un ejemplo de entrenamiento, momento en el que no se ha generado ninguna palabra, el decodificador recibirá como entrada un token especial que indicará el inicio de un texto (también existirá otro para determinar el final que deberá ser generado por la arquitectura). Por ejemplo, si se entrena el modelo con citas famosas de películas que estén incompletas, cuyo objetivo es acabarlas, y se quiere completar la famosa intervención del protagonista de la película *Forrest Gump* “Mamá siempre decía que la vida era como una caja de chocolates. Nunca sabes qué te va a tocar” a partir de la primera parte “Mamá siempre decía que la vida era como una caja de chocolates.”, la entrada de la arquitectura (que pasará al codificador) será siempre la tokenización (acción de transformar texto a vectores numéricos) de dicha primera parte. Por otro lado, en la primera iteración, la entrada adicional del decodificador será el token especial de inicio (en la literatura suele denominarse por *<start>*). De esta manera, tras pasar por el decodificador, la salida que se obtendrá se corresponderá con un vector con tantas componentes como palabras compongan el vocabulario de salida (que no necesariamente es el mismo que el vocabulario de entrada, como en el caso de los traductores). Cada elemento de dicho vector indicará la probabilidad de que sea la palabra que le corresponde la siguiente en generar. Asimismo, se escoge el token con mayor probabilidad y ese es el que es incluye al texto.

Supongamos que el transformer entrenado de ejemplo es suficientemente bueno y ha generado la palabra correcta: “Nunca”. Entonces, con la misma entrada del codificador, se vuelve a llamar al modelo, pero esta vez, dándole como parámetros de entrada al decodificador la tokenización resultante al concatenar el token especial de inicio y la palabra “Nunca”. Si el resultado es “sabes”, se continuará manteniendo la entrada del codificador y facilitándole al decodificador la tokenización de “*<start>Nunca sabes*”. Este proceso se repite hasta que el transformer generar el token especial de fin (normalmente denotado por *<end>*). En resumen, si facilitándole a nuestro modelo la entrada “Máma siempre decía que la vida era como una caja de chocolates.” se desea obtener “Nunca sabes qué te va a tocar”, la entrada de codificador será siempre la tokenización de “Máma siempre decía que la vida era como una caja de chocolates.”, y la del decodificador será la tokenización de “*<start>*” la primera vez, luego “*<start>Nunca*” (si ha generado la palabra “Nunca”) en la segunda iteración, después la entrada del decodificador será la tokenización de “*<start>Nunca sabes*” (si ha producido la palabra “sabes” en la iteración anterior) y, así sucesivamente, se le añadirán los tokens generados en la iteración anterior hasta que el modelo produzca el token “*<end>*”.

Una vez se ha comprendido lo que la arquitectura espera como entrada de sus módulos, se procede a entrar en detalle de las distintas partes de la misma, comenzando por las capas de *Embedding* (Mikolov et al., 2013a). Como se ha dicho anteriormente, no es posible dar como entrada del modelo valores de tipo texto, por lo que es necesario transformarlo a formato numérico. Existen distintas opciones para llevar a cabo esta conversión: asignación de números arbitraria, codificación *one-hot-encoding*, etcétera. Independientemente de la elegida, la representación de los tokens acarreará consigo problemas como la inconsistencia entre las relaciones numéricas como la distancia y las relaciones semánticas de las palabras a las que se corresponden (en el caso de la asignación arbitraria) o la equidistancia entre

todas las codificaciones entre sí (en el caso del one-hot-encoding). Por este motivo, es necesario utilizar la capa de embeddings, la cual consiste en una red neuronal que, a partir de la tokenización elegida, produce vectores de cada uno de los tokens y ajusta sus pesos en función de la finalidad del sistema. De esta manera, no solo se contempla la semántica de las distintas palabras pudiendo calcular su similitud hallando la distancia entre los vectores que las representan o el ángulo entre ambos, sino que también se genera un espacio n-dimensional (donde n es el tamaño de los vectores de salida, por lo general en los transformers se elige 512) en función del propósito de la red neuronal en el que cada dimensión determina una característica diferente de las palabras.

Sin embargo, como la capa de embeddings transforma token a token independientemente de su posición en el texto de entrada, el vector de salida (llamado vector embedding) no contará con dicha información que es necesaria, ya que, a diferencia de las redes LSTM que computan los tokens secuencialmente, los transformers, para ser más eficientes, procesan todos los vectores embedding a la vez. La razón por la que es recomendable tener en cuenta la información de la posición de cada token, es que la semántica de un texto puede verse drásticamente modificada en función de dicho orden. Por ejemplo, las frases “aunque no ganara el premio, estaba satisfecha” y “aunque ganar el premio, no estaba satisfecha” comparten los mismos tokens, no obstante, el orden de estos, en concreto de la palabra “no”, cambia completamente el significado y resulta una información crítica para entenderlas y diferenciarlas. Por esta razón, a los vectores embeddings, se le suman representaciones vectoriales (vectores de posición) de la posición del token con el que se corresponden, obteniendo así lo que se llama embeddings de posición. La pregunta es entonces, qué vectores se le deben sumar a los vectores embeddings y cómo deben construirse. Una primera idea podría ser que los vectores de posición constaran de todos los elementos iguales indicando el índice de la palabra en el texto (la primera palabra tendría un vector nulo, la segunda un vector con todos los elementos 1, la tercera sería un vector lleno con 2, y así consecutivamente). Desgraciadamente, esta solución no funciona correctamente, ya que la suma distorsionaría mucho los vectores embeddings producidos, especialmente aquellos que aparecen más tarde en el texto (por ejemplo, un texto de 100 palabras le sumaría a cada elemento del vector el gigantesco número 99, lo que introduciría mucho ruido).

Una solución que evita esta gran distorsión podría ser la de sumar vectores con todos los elementos iguales cuyos valores sean fracciones menores que 1. Una implementación de esta solución sería que el valor repetido que se le va a sumar a todas las posiciones de cada vector embedding sea $p/(N - 1)$ donde p es la posición del token en el texto (comenzando por 0) y N el número de tokens del mismo. Sin embargo, este apañío produciría vectores de posición distintos para token en el mismo lugar, pero en textos con distinta longitud. Probablemente esto generaría confusión en el modelo, complicaría su entrenamiento y reduciría su precisión. Quizás se podría arreglar definiendo N como una constante que sea la longitud máxima de un texto de entrada aunque el que se utilice sea más corto, pero este hecho evitaría que hubiera gran diferencia entre el comienzo y el final de un documento cuando su longitud es considerablemente más pequeña que la N elegida. Como consecuencia de ello, la información de posición se disiparía enormemente ante la suma con el vector de embedding y perdería su relevancia. Ante este problema, Vaswani et al. (2017) proponen las siguientes parametrizaciones para los vectores de posición:

$$PE_{(p,2i)} = \sin\left(\frac{p}{10000^{\frac{2i}{d}}}\right)$$

$$PE_{(p,2i+1)} = \cos\left(\frac{p}{10000^{\frac{2i}{d}}}\right)$$

donde p es la posición del token, d es la dimensión del vector embedding (512 en transformers) y los elementos $2i$ y $2i + 1$ a la izquierda de las igualdades indican la componente del vector de posición. De esta manera, se tendría que $PE_{(7,32)}$ es la componente 32 (se obtendría $i = 16$) del vector de posición del octavo token. La razón por la que se añade la variable i dentro de la función trigonométrica, es que si solo se contara con la posición p , con una fórmula como $PE_p = \sin(p)$, al tratarse de funciones periódicas, se generarían valores repetidos para posiciones diferentes. De esta forma, colocando la i como exponente del denominador, se consiguen sinusoides con frecuencias muy variadas que permiten que, aunque es posible obtener valores iguales en posiciones distintas, siempre se diferencien las posiciones en alguna componente del vector de posición.

Ya se han explicado los primeros módulos, tanto del codificador como del decodificador. A continuación se entrará en detalle acerca de la principal capa que da nombre al artículo de Vaswani et al. (2017), es el componente en el que se centra la atención de los autores y, gracias a él, se consiguen tan buenos resultados y tan excelente rendimiento en las tareas de generación de lenguaje natural: la capa de atención. Cuando se procesa texto, siempre ocurre que entre las distintas palabras del mismo existen relaciones (tanto sintácticas como semánticas) que el sistema debe ser capaz de detectar. Esto, supone un problema para las redes neuronales LSTM cuando la frase es suficientemente extensa, ya que, al procesar las entradas secuencialmente, la influencia de las palabras procesadas se va diluyendo conforme se reciben más tokens. Así, en la cita de la película de V de Vendetta “Su hermoso imperio tardó tanto en construirse, y ahora, con un chasquido de los dedos de la historia, es derrumbado”, resulta importante percibirse de que la forma pasiva “es derrumbado” hace referencia al sujeto “su hermoso imperio”, pero al encontrarse estos tokens tan alejados unos de otros una red LSTM “olvidaría” dicho sujeto al llegar al final de la oración, lo cual es un problema de gran relevancia.

Para encontrar las relaciones de las distintas palabras, independientemente de la distancia entre ellas, se utilizan los llamados mecanismos de atención. Estos, en la figura 2.6, se encuentran en la componente de nombre *Multi-Head Attention*. La idea subyacente a los mecanismos de atención, es ser capaces de, para cada palabra de la frase, calcular su relación con todas las de la oración (incluido consigo misma), tanto las posicionadas antes que ellas como las que se sitúan después. Para lograrlo, el mecanismo cuentan con tres redes neuronales. Una de ellas, la llamada *Query*, pretende ser la que extraiga las posibles características que, si otro token posee, podrían evidenciar una relación entre las dos palabras. Es decir, esta red trata de, dado un token, dar mayor peso a las componentes del vector de salida (vector query) que hagan referencia a las características “que busca” la palabra de entrada. Otra red neuronal del mecanismo de atención, denominada *Key*, es la encargada de caracterizar la palabra de entrada, o, expresándolo de otra manera, será la que especifique qué características “tiene” el token de entrada (la salida se llama vector key). En definitiva, con la query, se consiguen las propiedades que busca cada palabra en otras para tener una relación, y con la key se obtienen las características de la propia palabra. Así, basta con cruzar la información de la query de cada token (como se puede ver en la figura 2.7, en la que se cruza el vector query de la palabra i-ésima con el vector key de todas), con la key de todos los tokens para conocer el valor de la relación de cada palabra con todas las de la oración.

Dados dos vectores (el vector query y el vector key), matemáticamente se puede calcular su similitud hallando el coseno del ángulo que forman. Este se calcula con la siguiente fórmula:

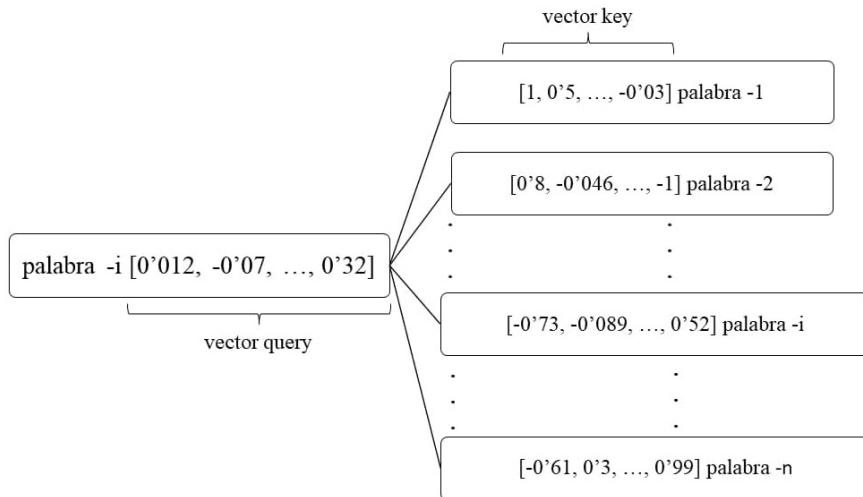


Figura 2.7: Esquema de atención query-key

$$\cos(\alpha) = \frac{\langle q, k \rangle}{|q| \cdot |k|}$$

donde $\langle q, k \rangle$ es el producto escalar entre ambos vectores y $|v|$ es el módulo de v . Asimismo es posible conocer un valor de relación entre cada palabra con todas las de la oración. La fórmula anterior puede generalizarse como sigue:

$$similitud(q, k) = \frac{\langle q, k \rangle}{e}$$

donde e es un factor de escalado preestablecido. Como lo que se busca es computar la similitud de todas los token con todos, se debe pasar la expresión anterior a forma matricial (con los vectores query de cada palabra como filas se conforma la matriz Q y, análogamente, se crea la matriz K con los vectores key de cada token):

$$similitud(Q, K) = \frac{Q \cdot K^T}{e}$$

La fórmula anterior genera otra matriz (llamada filtro de atención) cuyos elementos determinan la similitud entre las palabras, concretamente es posible afirmar que el elemento de la fila i -ésima y la columna j -ésima representa la medida de similitud entre la palabra i -ésima como query y el token j -ésimo como key. De hecho a cada fila de esta matriz se le denomina vector de atención de la palabra a la que corresponde la query, el cual indica qué palabras el modelo considera relevantes para dar contexto al token, o dicho de otra forma, a qué parte de la frase se está prestando atención.

Vaswani et al. (2017) determinan el factor de escalado como \sqrt{d} , donde d es la dimensión del vector query (o del vector key que poseen el mismo número de componentes). Además, a cada elemento resultante del filtro de atención, le aplican la función softmax para delimitar sus valores entre 0 y 1, quedando el cálculo de similitud de la siguiente manera:

$$similitud(Q, K) = softmax\left(\frac{Q \cdot K^T}{\sqrt{d}}\right)$$

Como se ha comentado anteriormente, los mecanismos de atención constan de tres redes neuronales. Se han presentado la query y la key. La tercera, conocida como *Value*, generará

un vector (vector value) cuyos elementos representarán las características de una palabra de cara a producir el resultado esperado por el sistema. Esto se consigue permitiendo que el backpropagation le afecte en mayor medida al no introducirlo en los cálculos anteriores. Su utilidad será aplicarle la matriz de filtro, que potenciará aquellas características a las que haya que prestarle mayor atención, es decir, aquellos elementos de la matriz que tengan valores más cercanos a 1. De esta forma, el cálculo final de la matriz de atención viene dado por la siguiente expresión:

$$\text{atencion}(Q, K, V) = \text{softmax} \left(\frac{Q \cdot K^T}{\sqrt{d}} \right) \cdot V$$

En resumen, si contamos con tres redes neuronales que, dados los tokens, obtengan los vectores query, key y value, es posible generar la matriz de atención combinando la matriz query con la matriz key, escalando el filtro de atención resultante, aplicando la función softmax para tener valores entre 0 y 1 y, por último aplicando el filtro a la matriz value (este flujo además de quedar formalizado por la expresión anterior, viene esquematizado en la imagen 2.8).

Scaled Dot-Product Attention

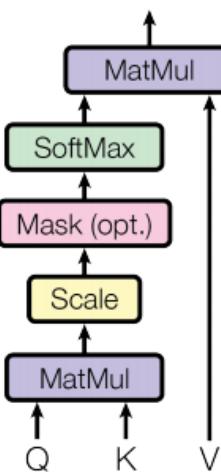


Figura 2.8: Cálculo de atención (Query-Key-Value)
Imagen extraída de Vaswani et al. (2017)

No obstante, nos queda por especificar cómo se generan las matrices query, key y value. Vaswani et al. (2017) proponen la utilización de una capa lineal de neuronas, que consiste en una capa de neuronas plenamente conectadas sin función de activación. Suelen resultar extremadamente útiles cuando se desea cambiar las dimensiones de la entrada de la red neuronal.

Con este mecanismo de atención explicado, es posible conocer las características más importantes en las que el sistema debe fijarse gracias al filtro de atención. Sin embargo, existen una amplia variedad de filtros que permiten concentrarse en otro tipo de conjunto de características que podrían enriquecer la salida de nuestro componente de atención. Por ese motivo, la arquitectura transformer trabaja generando ocho matrices distintas

de atención ($h = 8$), concatenándolas y reescalando la matriz final para ajustarse a las dimensiones. Este conjunto de operaciones es lo que se conoce como *Multi-Head Attention* (véase la figura 2.9 que resume toda la arquitectura) y se utiliza en varios puntos de la arquitectura propuesta por Vaswani et al. (2017) (véase la figura 2.6).

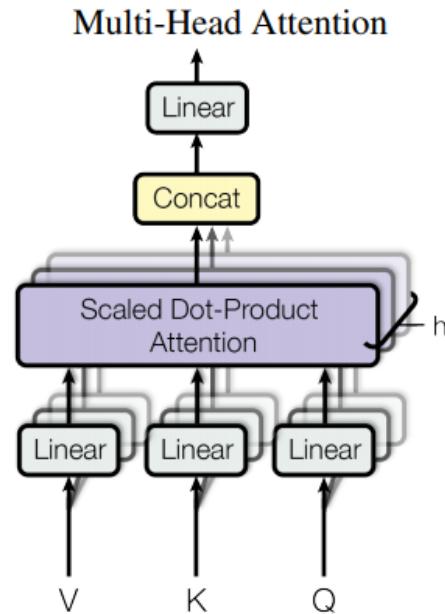


Figura 2.9: Multi-Head Attention
Imagen extraída de Vaswani et al. (2017)

Si se analiza en detalle la figura 2.6, es fácil percibirse que existen conexiones residuales, es decir, conexiones que, en lugar de ir de una componente de la arquitectura a la siguiente acaban en capas posteriores. Una de las ventajas de contar con este tipo de conexiones, es preservar parte del conocimiento de entrada, ya que conforme se profundiza y avanza en las distintas capas, la entrada se modifica considerablemente, lo que puede causar que se pierda información. Una solución a esto, es precisamente conectar los valores de entrada, no solo con la primera capa, sino también con capas ocultas de mayor profundidad, de manera que “se refresca” la información a la red neuronal. Para ello, basta con sumar los vectores que llegan a través de la conexión residual con los que son resultado del procesamiento de las capas anteriores. En la arquitectura que se está presentando, tras esta suma, se aplica una estandarización (cambiar la distribución de la muestra a una normal de media 0 y desviación típica 1) de los valores de la matriz resultante, a lo largo de sus columnas (las características de los tokens).

También se pueden observar componentes *feed forward*, los cuales se componen de dos capas lineales de neuronas con una capa con función de activación ReLU. Así queda explicada por completo la arquitectura del codificador, y faltaría presentar una pequeña variación de la componente de Multi-Head Attention detallada anteriormente que se puede encontrar en el decodificador: el enmascaramiento. Esta técnica simplemente consiste en multiplicar al filtro de atención una matriz triangular superior con la diagonal nula para que, durante el entrenamiento, solo se tomen como entradas del decodificador las palabras que ya se han generado y no las que quedan por producir. Esta pequeña variación sirve para no introducir como entrada del codificador lo que se espera como salida.

En resumen, la arquitectura transformer se divide en codificador y decodificador. El primero recibe la entrada del sistema y el segundo recibe como input el texto que ha ido generando hasta el momento. Cuando queremos usar este tipo de redes neuronales, primero se tokeniza el texto de partida y este pasa a una capa de embeddings que extrae propiedades de las diferentes palabras. A continuación se genera el embedding posicional sumándole al vector embedding el vector de posición que incluye la información de cómo se ordenan las palabras en la frase. A continuación, estos vectores que almacenan propiedades y posición de las palabras, entran por el módulo de Multi-Head Attention, del que se obtiene la matriz de atención. A esta se le suman los embedding posicionales y luego se estandariza la matriz. Luego, el resultado pasa por la componente feed forward y, de nuevo, volvemos a utilizar una conexión residual para sumarle al resultado de dicha capa la matriz de atención estandarizada. Al estandarizar el resultado se produce la salida del codificador, la cual se incluye como query y key de un módulo de Multi-Head Attention del decodificador. La clave de esta componente será el resultado de obtener la matriz de atención de la entrada del decodificador. El último paso de antes de la salida es de nuevo una capa feed forward, seguida de una conexión residual. Por último, la salida del decodificador llega a una capa lineal de neuronas que tendrá tantas neuronas como el tamaño del vocabulario de salida. Con esta capa, se calculan los pesos asignados a cada palabra del vocabulario que, al pasarlo por la función softmax final, se convierten en probabilidades que pueden ser interpretadas fácilmente y nos indican el token que el sistema recomienda generar.

2.2.3. Resumen abstractivo de textos

Se finaliza esta sección con una breve introducción al área de resumen abstractivo de textos, un pequeño ámbito dentro de la generación de lenguaje natural.

Normalmente, cuando se quiere diseñar un sistema capaz de generar resúmenes automáticos de textos, es posible encontrarse con dos aproximaciones: extractiva o abstractiva (Mani y Maybury, 2001). La primera de ellas consiste en seleccionar algunas de las distintas oraciones de la entrada y utilizarlas, con ligeras modificaciones, para generar el resumen. En los sistemas de NLG que implementan esta aproximación, la clave suele residir en qué frases elegir. Para ello, algunas investigaciones, como la de Plaza et al. (2010) y la de Pal y Saha (2014) aprovechan la estructura construida en WordNet (Miller, 1995) y las métricas de similitud semántica de dicha red (Pedersen et al., 2004). Otros trabajos, como el de Macdonald y Siddharthan (2016), definen su propia métrica de evaluación de la información contenida en cada frase en base a cálculos como la frecuencia de las distintas palabras.

En contraposición a los métodos extractivos, se encuentran los abstractivos. Estos requieren un análisis menos superficial del texto y la habilidad para generar nuevas oraciones. A pesar de resultar más complejos, demuestran ser capaces de reducir considerablemente la redundancia y mantener una comprensión razonable. Según Genest et al. (2013) hay un límite empírico intrínseco a los métodos de extracción pura en comparación con los de abstracción.

La dificultad que se encuentra para implementar los resúmenes abstractivos es en el paso de determinación del contenido (véase la sección 2.2.2.1) de la arquitectura realizer. Algunas propuestas para generar resúmenes con sistemas abstractivos evitan este paso utilizando la compresión de oraciones (Knight y Marcu, 2000; Cohn y Lapata, 2009), la fusión de frases (Barzilay y McKeown, 2005) o la revisión (Tanaka et al., 2009). Por otro lado, algunos autores como Genest y Lapalme (2011) abordan este problema para construir una aproximación puramente abstractiva en la que se separa el proceso del análisis del texto, del de la generación de las distintas oraciones del resumen. Con este propósito, Genest y

Lapalme (2011) definen lo que denominan *Information Items* (InIts) para generar una representación abstracta del contenido del documento de entrada. Un InIt es la entidad más pequeña de información coherente en un texto u oración. Puede implementarse como algo tan simple como la propiedad de una entidad o como algo tan complejo como una descripción completa de un evento o acción. El objetivo con esta aproximación es la identificación de todas las entidades del texto, sus propiedades, predicados entre ellas y características de dichos predicados. Este propósito es extremadamente dependiente a la implementación de los Information Items, ya que estos determinarán la información abstracta disponible para generar el resumen.

Los Information Items pueden desarrollarse utilizando técnicas de etiquetado del rol semántico (Palmer et al., 2010) o de análisis de predicados lógicos (Epstein, 2018). Además, estos métodos de análisis semántico, podrían complementarse con procedimientos de desambiguación del sentido de las palabras (Navigli, 2009), resolución de co-referencia (Soon et al., 2001) o análisis de similitud de palabras (Islam y Inkpen, 2008). Sin embargo, Genest y Lapalme (2011) optan por un desarrollo más simplista con el que consiguen resultados relativamente notables: definen un InIt como una tupla sujeto-verbo-objeto. Debe tenerse en cuenta que, aunque parezca que esta definición de Information Item se trata de una técnica de compresión de oraciones, resultan métodos distintos por tres razones principales: la frase que se generará solo contendrá un elemento de la oración (el InIt, aunque es posible combinar más de uno en una misma frase) y no toda la información de la frase original, una oración de entrada podría producir más de una frase de salida si cuenta con varios Information Item (mientras que la compresión de oraciones genera solo una sentencia de salida por cada frase de entrada) y la sentencia producida puede incluir palabras que no aparezcan en la oración original.

La implementación de los Information Items como tuplas sujeto-verbo-objeto, llevada a cabo por Genest y Lapalme (2010), incluye también un sistema de reglas creado manualmente que cubre un considerable número de situaciones sintácticas. De esta manera, utilizando este desarrollo, es posible generar representaciones semánticas genéricas independientemente del dominio de los textos, dificultad que se encontraba en la estructura de datos construida ad hoc que proponían Reiter y Dale (2000) para resolver la fase de determinación del contenido (explicada en la sección 2.2.2.1).

Capítulo 3

Descripción del Trabajo

“¡Datos! ¡Datos! ¡Datos! - exclamó con impaciencia - No puedo hacer ladrillos sin arcilla”

— El misterio de Copper Beeches
Arthur Conan Doyle (1892)

Tras comprender los conceptos necesarios para hacer frente al problema de construir un sistema capaz de redactar correos electrónicos a partir de pequeñas representaciones semánticas, se puede proceder a la explicación de lo desarrollado en este trabajo. En primer lugar es introducen brevemente las tecnologías utilizadas (véase la sección 3.1). A continuación se realiza un análisis de datos previo para conocer las propiedades y detalles de los datos con los que se va a trabajar y entrenar el modelo (consultar el apartado 3.2). Después se entra en detalle acerca de la arquitectura elegida, razones por las que no se han conseguido resultados con la arquitectura descartada y los ajustes concretos que se han implementado para abordar el problema que compete a este trabajo (secciones 3.3 y 3.4). Por último, se presentan los resultados obtenidos y una reflexión acerca de a qué se deben (léase el apartado 3.5).

3.1. Tecnologías utilizadas

Antes de comenzar con el desarrollo del trabajo realizado, en esta sección se introducen las distintas tecnologías que han sido empleadas para la implementación del proyecto. Dado que el estudio gira en torno a los cuerpos de los correos electrónicos, es imprescindible un analizador sintáctico que facilite el estudio del corpus de partida. Para cubrir esta necesidad, se ha elegido spaCy, herramienta que se detalla en la sección 3.1.1. Por otro lado, en lo relativo a las tareas relacionadas con el procesamiento de lenguaje natural, también se necesita poder obtener los Information Items dado un texto de entrada. Este problema puede implementarse gracias a la librería textaCy (véase el apartado 3.1.2).

En cuanto al desarrollo de la arquitectura transformer, la cual consta de numerosas capas de redes neuronales, como se explicó en la sección 2.2.2.2, será de gran ayuda la librería desarrollada por Google llamada Tensorflow (consúltese el apartado 3.1.3). Gracias a ella será posible presentar un modelo de aprendizaje automático que aborde todos los problemas de la generación de lenguaje natural.

Por último, se introduce el sistema de almacenamiento con el que se ha trabajado para contar con una gestión eficiente de los datos que se manejan: MongoDB (sección 3.1.4).

3.1.1. spaCy

Para ser capaces de llegar a conclusiones y estudiar el corpus de correos electrónicos seleccionado (véase la sección 3.2.1), se debe poder analizar el cuerpo de los mismos. Es decir, se necesita un analizador sintáctico que permita separar los diferentes textos en tokens (o dicho de otro modo, segmentar el texto en palabras, signos de puntuación, etcétera) y obtener diferentes características de ellos (como su categoría gramatical). Para conseguir ese objetivo, se va a utilizar la librería spaCy¹.

En esta sección se explican las razones por las que se elige spaCy (véase la sección 3.1.1.1) y su utilidad en el proyecto (véase la sección 3.1.1.2).

3.1.1.1. spaCy frente a otros analizadores sintácticos

Se ha elegido spaCy como analizador sintáctico frente a otros por varias razones, apoyadas por investigaciones publicadas como la realizada por Choi et al. (2015), y que se explican a continuación.

SYSTEM	YEAR	LANGUAGE	ACCURACY	SPEED (WPS)
spaCy v2.x	2017	Python / Cython	92.6	n/a ⓘ
spaCy v1.x	2015	Python / Cython	91.8	13,963
ClearNLP	2015	Java	91.7	10,271
CoreNLP	2015	Java	89.6	8,602
MATE	2015	Java	92.5	550
Turbo	2015	C++	92.4	349

Figura 3.1: Benchmarks de los distintos analizadores sintácticos
Imagen extraída de <https://spacy.io/usage/facts-figures#benchmarks>

Una evaluación publicada por *Yahoo! Labs* y la Universidad Emory, como parte de un estudio de las tecnologías de análisis sintáctico actuales (Choi et al., 2015), observó que “spaCy es el analizador sintáctico voraz más rápido” y su precisión está dentro del 1% de los mejores existentes (como podemos ver en la figura 3.1). Los pocos sistemas que son más precisos son, al menos, 20 veces más lentos. La velocidad es un factor importante cuando se quiere implementar sistemas complejos que se enfrentan a textos largos o a un gran número de documentos (como es el caso de este trabajo, en el que se quieren analizar todos los correos electrónicos posibles).

Los resultados de Choi et al. (2015) y las discusiones posteriores ayudaron a spaCy a desarrollar una novedosa técnica para mejorar la precisión del sistema, la cual fue publicada en un trabajo conjunto con la Universidad de Macquarie (Honnibal y Johnson, 2015). Por este motivo, se ha elegido una versión de spaCy que aprovecha esta técnica.

Además, no sólo en general, sino en cada tarea particular (tokenización, etiquetado y análisis sintáctico), spaCy es la más rápida si la comparamos con otras librerías de procesamiento del lenguaje natural. Esto se muestra en la figura 3.2, donde se puede observar

¹<https://spacy.io/>

SYSTEM	ABSOLUTE (MS PER DOC)			RELATIVE (TO SPACY)		
	TOKENIZE	TAG	PARSE	TOKENIZE	TAG	PARSE
spaCy	0.2ms	1ms	19ms	1x	1x	1x
CoreNLP	0.18ms	10ms	49ms	0.9x	10x	2.6x
ZPar	1ms	8ms	850ms	5x	8x	44.7x
NLTK	4ms	443ms	n/a	20x	443x	n/a

Figura 3.2: Tiempo de procesamiento por documento de varias librerías de NLP
Imagen extraída de <https://spacy.io/usage/facts-figures#benchmarks>

tanto los tiempos absolutos (en milisegundos) como el rendimiento relativo (normalizado a spaCy). Los sistemas que tienen valores más bajos son más rápidos en sus tareas.

3.1.1.2. Utilidades de spaCy

Se puede definir spaCy como una biblioteca de procesamiento de lenguaje natural de Python diseñada específicamente para ser una biblioteca útil para implementar sistemas listos para la producción. Por esta razón, tiene una gran cantidad de utilidades diferentes. Sin embargo, sólo se necesitarán las que realizan el *Tokenizer*.

La clase *Tokenizer* de spaCy se encarga de dividir el mensaje dado en las diferentes palabras que lo constituyen y obtener varias características sobre ellas. Interesan los atributos que se pueden observar en la tabla 3.1. Además de su categoría gramatical, da más información (que no nos interesa) en función de su categoría léxica, como su género, número, tiempo verbal o, incluso, el tipo de adverbio.

Atributo	Tipo	Explicación
is_punct	bool	Indica si el token es un signo de puntuación
is_right_punct	bool	Indica si el token es un signo de puntuación derecho (como el paréntesis cerrado).
is_left_punct	bool	Indica si el token es un signo de puntuación izquierdo
is_bracket	bool	Indica si el token es un paréntesis
like_url	bool	Indica si el token es una url
like_email	bool	Indica si el token es una dirección de correo electrónico
lema_	string	Forma base del token sin sufijos o inflexiones
is_stop	bool	Indica si el token es una stop word
pos_	string	Categoría grammatical
text	string	Verbatim text content.
idx	integer	The character offset of the token within the parent document.

Tabla 3.1: Atributos de interés de la clase *Tokenizer*

3.1.2. textaCy

Sobre la librería de spaCy se han desarrollado múltiples soluciones para distintos problemas en el ámbito del procesamiento de lenguaje natural. Uno de estos proyectos es

textaCy². Se trata de una librería que cuenta con la capacidad de extraer los Information Items (véase la sección 2.2.3), definidos como tuplas sujeto-verbo-objeto, de un texto. Bas-
ta con añadir la tarea de extracción de los InIts al pipeline de spaCy y, cuando se procese
un texto, se obtendrán de forma automática estas tuplas. De esta manera, se puede generar
una entrada para los correos electrónicos del corpus (que serían la salida del sistema) con
la que entrenar el modelo construido.

3.1.3. Tensorflow

Mientras que spaCy y textaCy son herramientas fundamentales durante el análisis y
procesamiento del corpus, la librería de código abierto Tensorflow (Abadi et al., 2016) es
la piedra angular del desarrollo de la arquitectura transformer implementada. Posee una
amplia cantidad de funcionalidades para trabajar con tensores de manera eficiente, está
especializada en los modelos de aprendizaje automático y permite ejecutar Keras utilizando
Tensorflow como base, la cual es una librería especialmente diseñada para implementar
arquitecturas de deep learning y que facilita la construcción, entrenamiento y evaluación
de las mismas.

3.1.4. MongoDB

Como se justificará más adelante en la sección 3.2.3, se necesita almacenar una gran
cantidad de correos electrónicos con el fin de poder trabajar de forma eficiente con ellos.
Para esta tarea se ha elegido MongoDB que es un sistema de base de datos NoSQL de
código abierto y orientado a documentos.

En lugar de almacenar los datos en tablas, como se hace en las bases de datos relacio-
nales, MongoDB almacena estructuras de datos BSON (una especificación similar a JSON)
con un esquema dinámico, lo que facilita y agiliza la integración de datos en determinadas
aplicaciones (Győrödi et al., 2015). Además, no se requieren recursos potentes para tra-
bajar con ella y, gracias a la flexibilidad que ofrece el ser una base de datos NoSQL, se puede
realizar fácilmente modificaciones en el modelo conceptual de la base de datos sin tener que
preocuparse por los cambios problemáticos entre claves primarias y foráneas entre tablas.
Además, cuenta con drivers oficiales para el lenguaje de programación Python con el que
se desarrolla la solución.

3.2. Análisis de los datos

El primer paso en todo trabajo de ciencia de datos e inteligencia artificial es el análisis
de los datos con los que se cuenta para entrenar y probar el modelo. Por ese motivo,
antes de explicar el desarrollo del sistema que será entrenado, se lleva a cabo un estudio
preliminar que permitirá la concepción de una idea acerca de cómo se distribuyen los
datos con los que se va a trabajar. Se comenzará la sección explicando el origen de los
datos y cómo, en bruto, se estructuran (véase la sección 3.2.1). A continuación, dado que
es conveniente adaptar el formato original para el propósito del sistema a desarrollar, el
corpus debe prepararse y limpiarse adecuadamente con el fin de eliminar problemas como
el ruido y las redundancias (léase el apartado 3.2.2). Una vez finaliza la importante fase de
limpieza de los datos, estos se procesarán para extraer las características que puedan ser
de interés y almacenarán en un sistema que mejore la eficiencia respecto a la que se tiene
con la estructura original de los datos en bruto (consúltese la sección 3.2.3). Por último, se

²<https://spacy.io/universe/project/textacy>

realiza un análisis exploratorio de los datos con el que se describirá el corpus de trabajo y gracias al que se podrá extraer cierta información que será de utilidad en el desarrollo de nuestro sistema de generación de lenguaje natural (el análisis exploratorio de los datos se encuentra en el apartado 3.2.4).

3.2.1. Enron corpus

Para llevar a cabo cualquier trabajo relacionado con la ciencia de datos e inteligencia artificial es necesario contar con un conjunto de datos con el que poder entrenar al modelo desarrollado. Cuando se trata de un estudio relacionado con la generación de lenguaje natural, el conjunto de datos se llama corpus y suele contener ejemplos de documentos reales redactados por humanos similares a los que se desea producir. En concreto, para este trabajo, se ha elegido el corpus conocido como Enron³, dado que los correos electrónicos que contiene pertenecieron a la empresa con el mismo nombre. Precisamente se hicieron públicos tras una investigación legal llevada a cabo a esta compañía por parte de la Comisión Federal de Regulación de la Energía⁴ de Estados Unidos.

Enron corpus contiene 517.401 correos electrónicos escritos en inglés de 150 usuarios distintos. Además de la ventaja de la gran cantidad de elementos pertenecientes a este dataset, también ha sido elegido por encontrar diversos trabajos sobre este mismo conjunto de e-mails, como el llevado a cabo por Klimt y Yang (2004).

Este corpus está organizado en una jerarquía de directorios (uno por cada usuario), en la que dentro de los principales se encuentran las correspondientes carpetas de la cuenta de correo electrónico del usuario en cuestión, como bandeja de entrada, enviados y directorios personalizados. En estas carpetas se encuentran, cada uno en un archivo separado, los distintos correos electrónicos del corpus. Un ejemplo de cómo se presentan los diferentes e-mails viene reflejado en la figura 3.3.

Como puede observarse en la figura 3.3, cada uno de los correos electrónicos se encuentra en un archivo distinto estructurado según el formato MIME explicado anteriormente (véase la sección 2.1.1). Claramente, se pueden diferenciar las distintas cabeceras de este formato con la información correspondiente a cada una de ellas. Después de todas estas cabeceras, se encuentra el cuerpo del mensaje. Por cada uno de los 517.401 correos electrónicos, se encuentra un archivo diferente con esta estructura.

3.2.2. Preparación y limpieza de los datos

El primer problema que aparece ante el conjunto de datos elegidos es el formato en que se presentan cada uno de los correos electrónicos, ya que en cada archivo hay mucha información que generaría ruido y dificultaría el entrenamiento del modelo si no se eliminara (como las cabeceras MIME). Por ese motivo, es necesario preprocesar cada uno de ellos. Asimismo, se conseguiría un dataset apto para el propósito de construir un sistema de generación de lenguaje natural que redacte correos electrónicos.

Para extraer la información relevante de cada uno de ellos, el lenguaje de programación *Python* cuenta con una librería⁵ capaz de parsear correos electrónicos almacenados en archivos y cadenas de caracteres en formato MIME. Además, una vez procesadas las cadenas, la clase devuelta cuenta con una serie de métodos que facilitan la obtención de la información de las cabeceras y recorrer el árbol de partes MIME (véase la sección 2.1.1 o

³<http://www-2.cs.cmu.edu/~enron/>

⁴<https://www.ferc.gov/>

⁵<https://docs.python.org/3/library/email.parser.html>

```

Message-ID: <15613608.1075858196357.JavaMail.evans@thyme>
Date: Tue, 18 Jul 2000 08:03:00 -0700 (PDT)
From: mike.carson@enron.com
To: mcarson@gtemail.net
Subject: FW: Fishingtrip Nat gas /electricity
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-From: Mike Carson
X-To: mcarson@gtemail.net
X-cc:
X-bcc:
X-Folder: \Mike_Carson_Dec2000\Notes Folders\Sent
X-Origin: Carson-M
X-FileName: mcarson2.nsf

----- Forwarded by Mike Carson/Corp/Enron on 07/18/2000
03:49 PM -----

"Wodenshek, Chris" <CWodenshek@cantor.com> on 07/18/2000 02:11:36 PM
To: "'john.zufferli@enron.com'" <john.zufferli@enron.com>,
"'rogers.herndon@enron.com'" <rogers.herndon@enron.com>,
"'mike.carson@enron.com'" <mike.carson@enron.com>
cc:

Subject: FW: Fishingtrip Nat gas /electricity

Joh,Rogers and Mike,
The attached details a fishing trip we would like to invite the three of you
to. We need to reserve boats and hotels soon so check and let me know if
you can make it.

Thanks
Chris and Buck

```

Figura 3.3: Ejemplo de un correo electrónico del corpus Enron

consúltese el ejemplo de árbol representado en la figura 2.1). De manera que solo ha sido necesario desarrollar un método que vaya recorriendo la estructura arbórea, comprobando el tipo MIME del nodo, y extrayendo el cuerpo del mensaje cuando lo encuentre.

Una vez se ha resuelto el problema de recuperar el correo electrónico dado el archivo perteneciente al corpus, es posible enfrentarse a otro reto antes del análisis exploratorio de los datos: la limpieza del cuerpo del mensaje para contar con un texto plano de cara al entrenamiento del modelo. Siempre, en todo trabajo en ciencia de datos, se presenta una fase de limpieza de los datos. Este paso resulta ligeramente más complicado cuando se está tratando con cadenas de caracteres. En el caso que ocupa a este trabajo, la limpieza consiste en encontrar patrones externos al cuerpo del mensaje, como la firma o la inclusión del mensaje al que se responde debajo de la respuesta, que no constituyen texto escrito por los usuarios, sino que ha sido incluido por el servidor de correo electrónico. Por ejemplo, en la figura 3.3, se muestra un e-mail el cual simplemente está reenviando otro mensaje distinto sin incluir más información. Esto resulta evidente por el comienzo del cuerpo del correo electrónico, que indica el reenvío. Este e-mail no debería incluirse en el entrenamiento de nuestro modelo, porque no añade información nueva y repite un mensaje que ya se tiene en otro archivo (dado que se cuenta con todos los correos del usuario, si el usuario está reenviando un e-mail, significa que es posible encontrar el mensaje original en la bandeja

de entrada o alguna de las otras carpetas de su cuenta de correo). Por este motivo, es indispensable limpiar el cuerpo de los todos correos electrónicos del corpus.

Tras analizar concienzudamente el corpus de correos electrónicos se detectan varios patrones que deben abordarse y cambios que tienen que llevarse a cabo en los mensajes. El primero de ellos consiste en que, cuando un usuario contesta a un mensaje, el servidor incluye el mensaje que es respondido debajo de la contestación. En este caso solo interesa la respuesta y no el texto original (pues estará en otro archivo). No obstante, esta casuística no constituye un problema complicado de solventar, ya que se puede distinguir la respuesta del mensaje original porque siempre se incluye una cabecera de texto que no varía. Es decir, basta con encontrar la cabecera como subcadena en el cuerpo y eliminar todo lo que le preceda. La misma solución puede aplicarse al patrón que aparece cuando se envía un correo electrónico modificando una convocatoria de reunión. Resulta que el servidor genera una cabecera para diferenciar la modificación de la convocatoria original.

Ligeramente más complicado respecto a los patrones anteriores, aunque no demasiado, resulta el caso que se muestra en la figura 3.3. Se trata del reenvío de un e-mail. Cuando esto ocurre, la cabecera producida por el servidor no se trata de una subcadena fija, sino que incluye como información variable adicional el nombre del usuario que reenvía el correo, así como la fecha y hora en que se produce dicho reenvío. Aunque la solución sea la misma, eliminar lo que precede a esta cabecera, no basta con buscar una subcadena que no varía, se requiere la utilización de expresiones regulares (Thompson, 1968). Como la gran mayoría de lenguajes de programación, Python cuenta con un módulo para implementar expresiones regulares⁶ que ha facilitado enormemente esta tarea y ha hecho posible la limpieza de los mensajes con este patrón.

Otro problema que ha sido necesario abordar es el de la firma de los servidores de correo (frases al final de los mensajes como “Get your FREE download of MSN Explorer”). Al ser siempre iguales, simplemente ha bastado con detectarlos y eliminar dicha subcadena del cuerpo de los mensajes. La dificultad de este tipo de limpieza de texto en realidad reside en detectar estos patrones, ya que, al tratarse de cadenas de caracteres, es complicado detectar incongruencias o errores.

Por último, debido al formato establecido por el protocolo MIME (por lo general depende de la codificación especificada para el mensaje), los servidores de correo electrónico incluyen tabulaciones y saltos de línea con una determinada frecuencia entre las palabras del texto, incluso aunque el usuario no los haya introducido. Dado que el salto de línea o la tabulación no es una información que se considere relevante de cara a la generación de texto de este modelo, se decide transformar estos caracteres en espacios y, a continuación, aplicar las expresiones regulares para detectar las subcadenas en las que se observe más de un espacio en blanco seguido y sustituirlas por uno solo.

Con esto concluye la limpieza de los cuerpos de los mensajes y la fase de preparación de los datos para adaptar los correos electrónicos a un formato adecuado para el sistema de almacenamiento elegido.

3.2.3. Procesado y almacenamiento

Como se ha explicado en el apartado 3.2.1, por defecto el corpus se almacena localmente estructurado en una jerarquía de directorios y contando con un archivo por cada correo electrónico. Sin embargo, esta forma de almacenamiento no resulta la más adecuada debido a que imposibilita cualquier método de búsqueda eficiente (sería necesario, en el peor de los casos, abrir todos los archivos para encontrar, por ejemplo, un e-mail en función

⁶<https://docs.python.org/3/library/re.html>

del identificador del mensaje) y resulta excesivamente lento cuando se pretende procesar todos los mensajes (requiere recorrer la jerarquía como una estructura de datos arbórea entrando en todos los directorios). Por estas razones, la decisión de cambiar el sistema de almacenamiento es acertado para agilizar tanto la carga del corpus como las distintas operaciones que se pueden querer llevar a cabo sobre el mismo.

Como los elementos del conjunto son textos, es decir, datos no estructurados, un almacenamiento clásico en archivos de extensión `.csv` podría provocar problemas como la elección del separador de los distintos campos, habría que utilizar un carácter que no apareciera en ningún cuerpo de mensaje ni en sus otras propiedades (asunto, identificador, destinatario, emisor, etcétera). Por lo tanto, un sistema relacional no parece que sea la mejor opción de almacenamiento para los correos electrónicos. Ante esta situación, se ha elegido el uso del sistema de base de datos NoSQL MongoDB (léase la sección 3.1.4) alojado en la nube⁷. La decisión de no implantarlo en un repositorio local se sustenta sobre la ventaja que ofrece la nube de disponibilidad desde cualquier dispositivo, lo cual ha sido de gran ayuda en el proceso de desarrollo del trabajo.

Una vez se ha seleccionado la forma de almacenamiento, queda por determinar las colecciones que se van a crear y los campos de los que dispondrán los documentos. Una colección que es indispensable es la que albergará los correos electrónicos, la cual contará con la siguiente estructura:

```
{
    # Identificador del documento mongo
    '_id' : ObjectId,
    # Identificador MIME del mensaje
    'Message-ID' : string,
    # Emisor del mensaje
    'From' : string,
    # Destinatario/s del mensaje
    'To' : string,
    # Asunto del mensaje
    'Subject' : string,
    # Cuerpo del mensaje
    'Body' : string,
    # Numero de palabras del cuerpo del mensaje
    'Number_of_Words' : integer,
    # Numero de Information Items del cuerpo del mensaje
    'Number_of_Inits' : integer
}
```

A excepción de los dos últimos campos, los demás se pueden extraer directamente mediante el uso de la librería específica de Python para mensajes en formato MIME (como se ha explicado en el apartado 3.2.2). Para obtener los dos últimos, es necesario procesar el cuerpo del mensaje antes de su almacenamiento en la nube. El primero, el número de palabras que contiene el cuerpo del correo electrónico, es posible hallarlo gracias al uso de spaCy (véase la sección 3.1.1), librería que no solo tokeniza el texto, sino que nos permite distinguir entre los tokens que son palabras del resto, como, por ejemplo, signos de puntuación.

El último campo del documento que representa a los correos electrónicos, indica el número de Information Items (véase la sección 2.2.3) que pueden extraerse del documento siguiendo la implementación llevada a cabo por Genest y Lapalme (2010). Estos InIts, que se definen como tuplas sujeto-verbo-objeto, se obtienen gracias a la librería textaCy

⁷www.mongodb.com/cloud

(consúltese el apartado 3.1.2), construida a partir de spaCy. De hecho, las representaciones semánticas abstractas que constituyen los Information Items serán la entrada de nuestro sistema de generación de lenguaje natural, por lo que, para evitar volver a extraerlos (su obtención es una operación computacionalmente costosa), será conveniente almacenarlos como otro documento de MongoDB. Dicha colección poseerá la siguiente estructura:

```
{
    # Identificador del documento mongo
    '_id' : ObjectId,
    # Identificador MIME del mensaje del que se extrajo el InIt
    'Message-ID' : string,
    # Sujeto del InIt
    'Subject' : string,
    # Verbo del InIt
    'Verb' : string,
    # Objeto del InIt
    'Object' : string
}
```

Por ejemplo, si en el cuerpo del mensaje aparece la construcción lingüística “I got change” (en castellano “tengo cambios”), se obtendrá el Information Item que se muestra en la figura 3.4. De esta forma, es posible relacionar fácilmente un correo electrónico con sus InIts y viceversa.

```
_id: ObjectId("614c2793aec3887e258e4952")
Message-ID: "<2025571.1075842929696.JavaMail.evans@thyme>"
Subject: "I"
Verb: "got"
Object: "changes"
```

Figura 3.4: Ejemplo de documento de un Information Item

Durante el proceso de almacenamiento, también se ha llevado a cabo un primer filtrado de mensajes que no son de utilidad para el propósito que persigue este trabajo y que, por tanto, pueden ser descartados sin guardarlos en MongoDB. Estos son: los correos electrónicos que carecen de cuerpo (puede ser porque originalmente no poseían o porque tras las operaciones de limpieza del mismo que han sido explicadas en la sección 3.2.2, se produzca esta situación), los e-mails de los que no se extrae ningún Information Item (ya que los InIts serán la entrada de nuestro sistema de generación de lenguaje natural) y los mensajes que son consecuencia de un error en el servidor de mensajería (por ejemplo, al intentar mandar un e-mail con una dirección de destinatario inexistente el servidor de correo siempre manda un mensaje informando de este error). De esta forma, se detectan 35.110 mensajes sin caracteres en el cuerpo, 121.799 correos electrónicos de los que no se extrae ningún Information Item y 7 e-mails consecuencia de errores en el servidor de mensajería, almacenando 360.485 correos electrónicos en la base de datos de MongoDB acompañados de 3.407.099 Information Items.

3.2.4. Análisis exploratorio de los datos

Tras llevar a cabo las tareas de preparación, limpieza de los datos, procesamiento y almacenamiento, con un filtrado a priori, se pueden observar las distribuciones numéricas del corpus desarrollando un análisis exploratorio de los datos. Este estudio preliminar ofrecerá una descripción acerca de los correos electrónicos y sus Information Items.

El primer, y probablemente más sencillo, parámetro que puede analizarse es el número de palabras de cada correo. Para observar cómo se distribuye esta variable en el conjunto de datos, se han calculado las frecuencias requeridas para generar la figura 3.5. En ella puede deducirse que, a pesar de poseer correos electrónicos con una gigantesca cantidad de palabras, la mayoría de mensajes muestran un número más razonable para tratarse de e-mails.

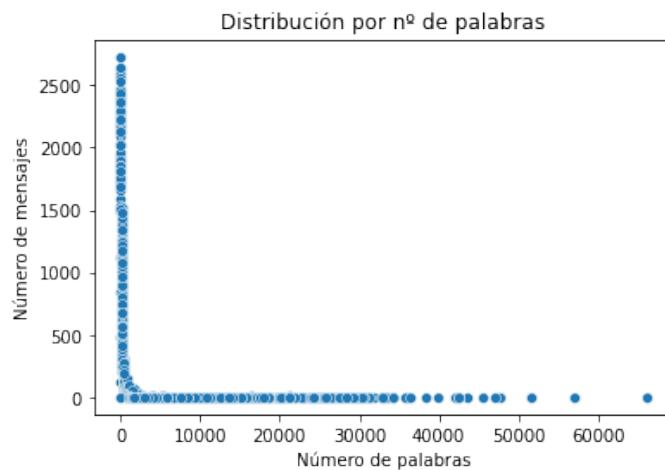


Figura 3.5: Distribución del número de palabras en los mensajes

Resulta evidente que la representada en la figura 3.5, se trata de una distribución con muchos *outliers* por la derecha. En lo que se refiere al apuntamiento, claramente se observa un intervalo relativamente pequeño en el que se concentran una gran cantidad de correos electrónicos. Sin embargo, de dicha gráfica resulta complicado extraer más conclusiones de las mencionadas, por la falta de detalle en la parte más apuntada de la distribución. Por esta razón, se ha delimitado el eje de abscisas y, de esta forma, se puede estudiar en mejores condiciones cómo se distribuye la variable del número de palabras a lo largo del conjunto de datos. El resultado de tomar un intervalo de los primeros valores del eje x (el número de palabras) es la figura 3.6.

Observando la figura 3.6, sí se pueden extraer conclusiones más fundamentadas y con mayor exactitud. Lo más relevante, probablemente sea el hecho de que una amplia cantidad de mensajes del corpus (seguramente un número mayoritario de e-mails) posean menos de doscientas palabras. Esta observación concuerda más con la naturaleza intrínseca del correo electrónico: suelen estar constituidos por un número reducido de oraciones simples y cortas. Resultaría razonable reducir la muestra del corpus en función del número de palabras vista la distribución presentada, pero la decisión del número límite para diferenciar entre los escogidos y los que no resultaría una elección un tanto arbitraria y precipitada si no analizáramos otros parámetros que se ven más adelante. Así que lo más prudente, antes de reducir drásticamente el conjunto de datos en base a un primer vistazo de una variable, es estudiar otras características del corpus.

La otra variable numérica con la que se cuenta en el conjunto de datos es el número de

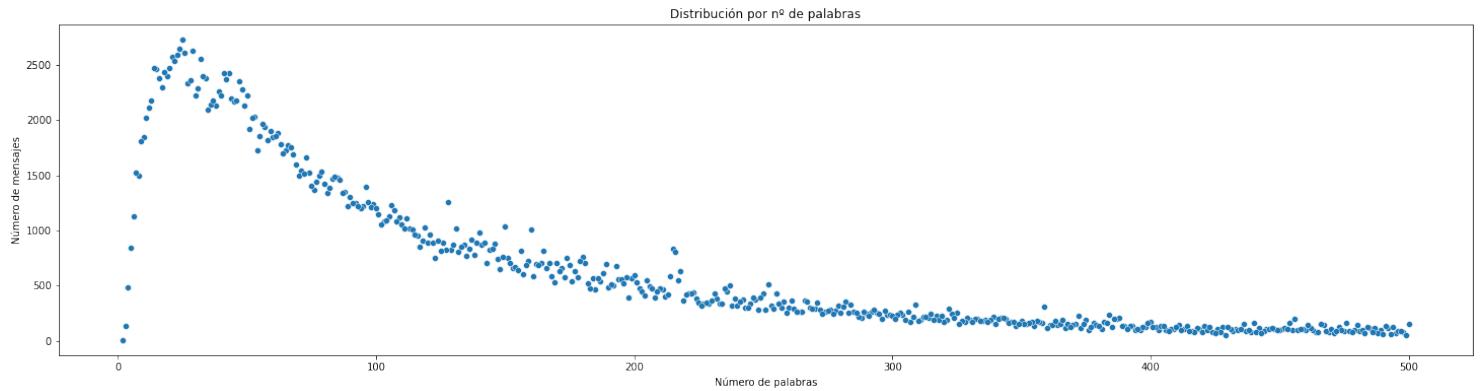


Figura 3.6: Distribución del número de palabras en los mensajes delimitando los ejes

Information Items extraídos del cuerpo del mensaje. Si se analiza la distribución general de esta variable, se obtiene un resultado similar al de la figura 3.5, pero, como cabe esperar, con valores en el eje de abscisas considerablemente menores (el máximo valor se encuentra cerca de 2500 InIts). Tener menos cantidad de Information Items que de palabras es un resultado coherente con la definición de InIt como tupla sujeto-verbo-objeto. No obstante, la distribución del número de Information Items parece ser, a primera vista, extremadamente más apuntada que la de los mensajes, como queda patente en la figura 3.7. No obstante, conviene tener en cuenta que aunque siempre se van a extraer menos Information Items que palabras en un correo electrónico, resulta mucho más improbable que dos e-mails coincidan en la cantidad de palabras, la cual varía fácilmente, que en el número de InIts. Esta, es probablemente la causa de que la distribución del número de palabras sea menos apuntada y su curva más suave que la de la cantidad de Information Items.

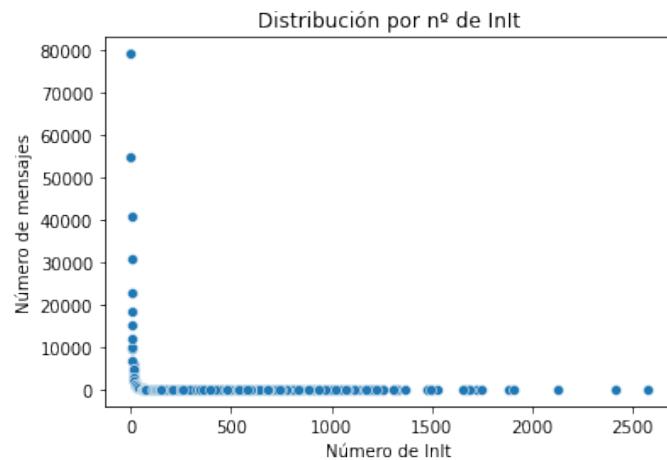


Figura 3.7: Distribución del número de Information Items en los mensajes

De nuevo, la figura 3.7, apenas aporta información detallada acerca de esta variable, por lo que es deseable volver a limitar el eje de abscisas para estudiar el comportamiento de la distribución donde se sitúan la inmensa mayoría de los correos electrónicos del corpus. El resultado de esta restricción del eje x viene dado por la figura 3.8, en la cual sí se distinguen fácilmente los distintos valores del eje horizontal y ayuda a sacar conclusiones de la distribución de esta variable.

A diferencia de lo que ocurría en la distribución del número de palabras por mensaje,

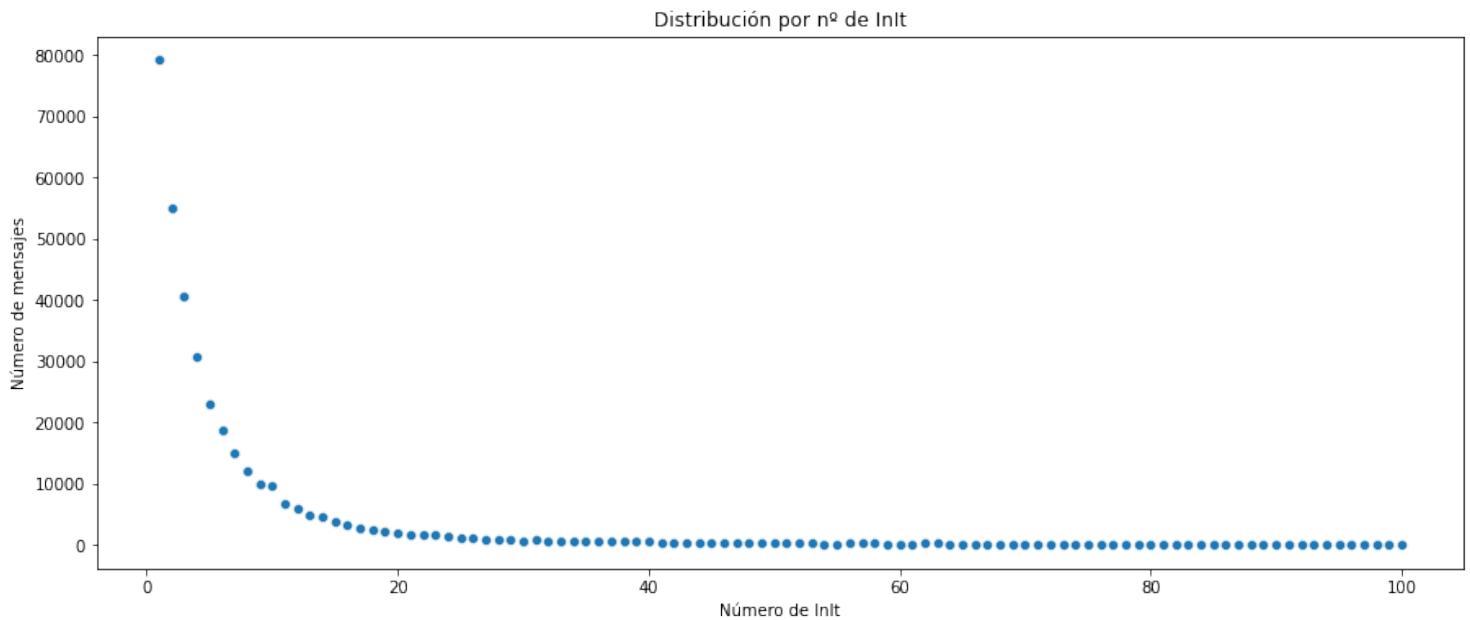


Figura 3.8: Distribución del número de Information Items en los mensajes delimitando los ejes

en la que se observaba un crecimiento inicial hasta alcanzar el máximo valor, en el caso de la cantidad de Information Items, la frecuencia de cada valor disminuye gradualmente. Probablemente, esta observación tenga su origen en el hecho de que con un número no tan elevado de InIts, ya se necesite un correo electrónico más largo de lo habitual. Además, existen diferencias sumamente considerables entre las frecuencias de los primeros valores y el resto. Prueba de ello es que se pueden encontrar más de 30.000 correos electrónicos con cuatro Information Items (que si se compara con los 80.000 e-mails con un único InIt también supone una diferencia abismal), mientras que con siete apenas se tienen 15.000 mensajes (ni la mitad).

Antes se comentaba la arbitrariedad que suponía el hecho de reducir el conjunto de datos tomando un límite de número de palabras tan solo habiendo observado las gráficas de su distribución. No obstante, con los Information Items no resulta tan subjetiva esta decisión. Si se conciben los InIts como las entidades informacionales que indican qué se desea tratar en el correo electrónico, resulta razonable pensar que en un e-mail no suelen utilizarse más de cinco o seis Information Items en los casos de mensajes más extensos. Esta percepción viene apoyada por el hecho de que en la distribución los órdenes de magnitud de la frecuencia con la que aparecen los mensajes con muy pocos InIts, son notablemente superiores a la que presentan los correos electrónicos que poseen tres o cuatro Information Items más. Por estos motivos, se plantea la posibilidad de seleccionar aquellos e-mails que no superen un número determinado de InIts. Pero, para elegir este límite se profundizará ligeramente más en la distribución del número de Inits, observando su distribución acumulada normalizada que se muestra en la figura 3.9.

La intuición de que un mensaje de gran extensión solía rondar los cinco o seis Information Items, resulta no estar muy distanciada de la realidad ya que, si se observa el valor porcentual correspondiente a la distribución acumulada de una cantidad de seis InIts, se obtendrá una medida cercana al 70 % del total de correos electrónicos del corpus. Concretamente, si se calcula, se obtiene que si se cogieran mensajes con, a lo sumo, seis

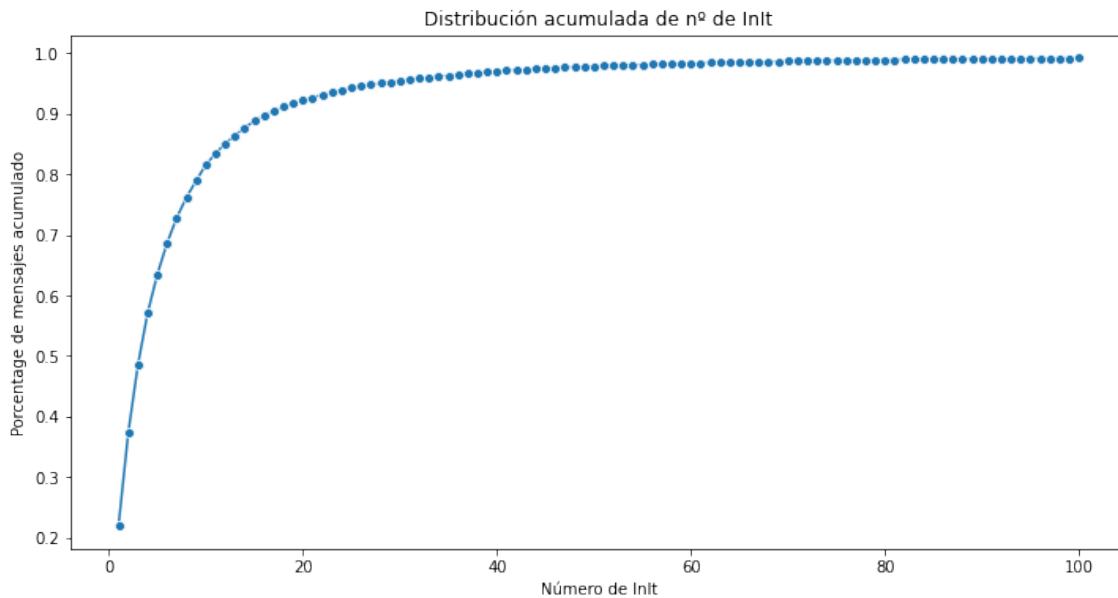


Figura 3.9: Distribución acumulada normalizada del número de Information Items

Information Items, se estaría escogiendo aproximadamente el 68.65 % del corpus almacenado en el sistema de bases de datos, lo que se corresponde con un total de 246.831 correos electrónicos. Esta cantidad suele ser suficientemente grande como para entrenar un modelo de aprendizaje automático satisfactoriamente, por lo que esta reducción de la muestra no solo no afectaría negativamente ni produciría una falta de elementos de entrenamiento, sino que homogeneizaría el conjunto de correos electrónicos de manera que todos tendrían un tamaño razonable acorde a su naturaleza lingüística y versarían sobre un número de temas no excesivamente extenso.

Por todos los motivos expuestos anteriormente, se toma la decisión de llevar a cabo un filtro de mensajes en función de su número de Information Items, teniendo en cuenta aquellos cuya cantidad no supere los seis InIts. A continuación, se entra en detalle acerca de cómo esta reducción ha afectado a la variable del número de palabras en el cuerpo de los mensajes.

Nº de InIts	1	2	3	4	5	6
Nº de mensajes	79165	54911	40628	30638	22911	18578
Media	43.48	70.42	95.95	118.9	138.77	176.15
Desviación Típica	54.15	82.43	97.89	106.17	100.21	163.81
Mínimo	2	6	10	14	18	29
25 %	16	31	48	65	83	101
50 %	29	50	73	94	117	142
75 %	51	82	111	138	163	195
Máximo	3105	4054	2647	1587	2921	5218
$Q3 + 1,5 \cdot RI$	103.5	158.5	205.5	247.5	283	336

Tabla 3.2: Estadísticos descriptivos del número de palabras agrupados por el número de Information Items

En la tabla 3.2, se muestran los estadísticos descriptivos de la variable que indica el número de palabras de un mensaje, habiendo agrupado en función del número de Infor-

mation Items. El detalle que más salta a la vista es el hecho de que, en todos los casos, el valor máximo del conjunto se encuentra siempre muy distante del resto de valores, incluido el tercer cuartil. Esto indica que hay cierto número de outliers en todos los grupos del conjunto de datos. Como esto produce una gran diferencia respecto al resto de mensajes, se considera la posibilidad de eliminarlos de la muestra, razón por la cual se ha incluido como última fila de la tabla el límite superior a partir del cual se definen los outliers o, dicho de otra manera, aquellos elementos que sobrepasan dicho límite se considerarán outliers. Este límite se calcula con la fórmula $Q3 + 1,5 \cdot RI$, donde $Q3$ es el tercer cuartil (representado por 75 % en la tabla 3.2) y RI es el rango intercuartílico, es decir, la diferencia entre el tercer cuartil y el primero.

Como el hecho de eliminar los outliers de estos conjuntos apenas supone una pérdida para el conjunto de datos, se quedaría con 231.288 mensajes de los 246.831, se lleva a cabo esta segunda reducción y se estudia la distribución de la variable de número de palabras en función con la cantidad de Information Items. El resultado de este análisis es la figura 3.10.

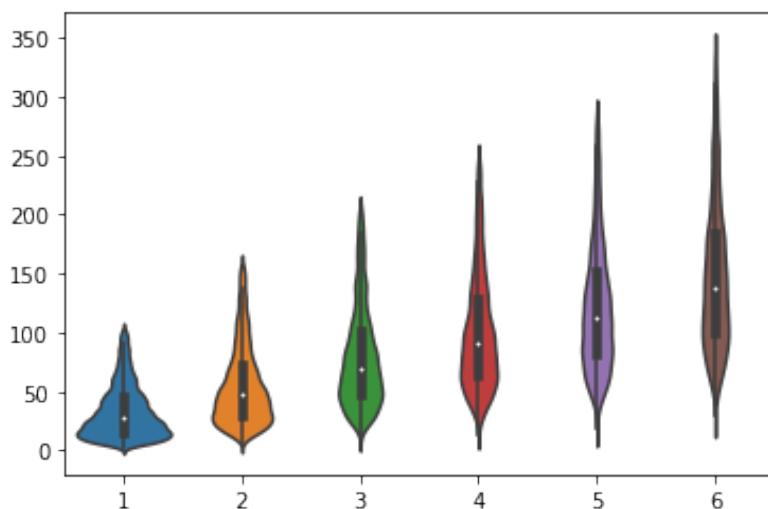


Figura 3.10: Distribución del número de palabras en función del número de Information Items

Las distribuciones presentes en la figura 3.10, resultan bastante razonables en cuanto al número de palabras se refiere cuando se habla de correos electrónicos. Todas ellas presentan asimetría positiva y, conforme se aumenta el número de Information Items, el apuntamiento se reduce considerablemente, lo que es consecuencia de que los elementos estén menos concentrados alrededor de la media y la mediana. Conviene también resaltar, que las distribuciones son coherentes con la definición de InIt, ya que conforme aumenta la cantidad de Information Items, la media, mediana y cuartiles del número de palabras aumenta.

Con esta última reducción se cuenta con un conjunto de datos bastante homogéneo y acorde con el tipo de correos electrónicos que se pretende generar: con una longitud razonable no muy extensa (hasta aproximadamente 350 palabras) y con un número de Information Items menor o igual a seis. Resumiendo el análisis exploratorio de datos, inicialmente se ha observado una distribución del número de palabras excesivamente dispersa, al igual que el número de InIts. Para solventar este problema se ha filtrado el conjunto de datos eliminando aquellos mensajes que poseyeran más de seis Information Items. A

continuación, estudiando el conjunto resultante, se han quitado los outliers, obteniendo la distribución final representada en la figura 3.10.

3.3. Implementación de una arquitectura realizer

El planteamiento inicial de este estudio era el de desarrollar una arquitectura realizer que fuera capaz de redactar correos electrónicos de manera automática. Como se explicó detalladamente en la sección 2.2.2.1, el primer problema que surge ante esta propuesta son las estructuras de datos ad hoc generadas en función del dominio del que se quiere producir los textos de lenguaje natural. Esto resulta un inconveniente debido a que, a diferencia de aplicaciones con un dominio definido como pueden ser aquellas cuyo propósito es el de generar informes meteorológicos, la redacción automática de correos electrónicos resulta extremadamente difícil de enmarcar dentro de uno o varios dominios específicos. Además, el corpus que se ha elegido para entrenar el modelo, no se restringe a un tipo de temática de e-mails, sino que pueden encontrarse mensajes que versan sobre una gran variedad de temáticas. Esto complica más la implementación porque, aunque se limitara a un número razonable de asuntos y este hecho no fuera desvelado, el modelo podría ser capaz de aprender el lenguaje específico y las características lingüísticas inherentes a los dominios (por supuesto con mayor dificultad que si se fuera consciente de esta ventaja). Sin embargo, al no contar con esta facilidad, no existen entidades concretas o propiedades comunes más allá de las que posee el lenguaje general.

Si se estudia en detalle la arquitectura realizer, se observa que el mayor problema de restricción de dominio se encuentra en la fase de determinación del contenido. Por supuesto que en el resto de fases también está presente en mayor o menor medida, por ejemplo en la estructuración del documento, pero no posee tanta dependencia como esta primera tarea localizada en el pipeline de las arquitecturas de generación de lenguaje natural. La razón es que resulta sumamente complicado conceptualizar en una estructura de datos concreta, cualquier posible intención que se pueda tener a la hora de transmitir cualquier información. Aún así, la solución traída desde el ámbito del resumen abstractivo de textos, en la que dichas intenciones se materializan en tuplas sujeto-verbo-objeto, solucionaba en gran medida la dificultad del paso de determinación de contenido. Es decir, esta fase se resolvía mediante la generación de Information Items que indican los temas que se desean tratar a lo largo de todo el correo electrónico. De hecho, la gran ventaja de llevar a cabo esta aproximación es que, no solo abordaba el problema de determinación del contenido que, en principio, parecía insalvable, sino que también ofrecía una solución para generar la entrada para el entrenamiento del sistema. Dicho de otro modo, el corpus elegido no proporciona más que los correos electrónicos, que en teoría son la salida final del sistema de generación de lenguaje natural, por lo que no se cuenta con una entrada. Con la solución de las tuplas sujeto-verbo-objeto, es posible conseguir una supuesta entrada desde la que podrían haberse redactado los mensajes. Esto da paso a la utilización de técnicas de aprendizaje supervisado y evita tener que producir una entrada escrita a mano. En resumidas cuentas, con los Information Items “matábamos dos pájaros de un tiro”: se conseguía un método de generación automática de una entrada para los correos electrónicos ya generados y se solventaba el problema de no ser capaces de concretar el dominio de los e-mails del corpus de cara a diseñar estructuras de datos ad hoc necesarias para implementar la fase de determinación del contenido. Sin embargo, los Information Items introducen un problema sumamente complicado de abordar. Al tratarse de estructuras genéricas que pueden versar sobre cualquier temática, esto impide que puedan usarse técnicas que añadan algún tipo de información adicional en la generación del texto. El origen de este gran escollo es que

las fases subsiguientes a la determinación de contenido son altamente dependientes de la salida de esta última. Cuando el dominio es concreto se puede razonar sobre el contenido (con técnicas como las ontologías) o contar con entidades preestablecidas para añadir información extra al texto producido, mientras que al no poder enmarcarse dentro de uno o varios dominios, actualmente no existe la forma de razonar sobre conocimiento general. En consecuencia, el sistema de generación de lenguaje natural se limitaría a recibir las tuplas sujeto-verbo-objeto y construir estructuras lingüísticas que incluyeran única y exclusivamente la información semántica que transmiten los InIts en cuestión. Es decir, simplemente se podrían producir, a lo sumo, tantas oraciones como Information Items se recibiera y la única tarea sería la de generar las escasas estructuras sintácticas que faltaran, así como ajustar algunas categorías morfológicas como el tiempo de los verbos y el género y número de las palabras. Esta técnica de Information Items solo ha sido empleada en los sistemas de resumen automático de textos, precisamente porque impide que en el resto de fases se pueda añadir información adicional más allá de estructuras sintácticas necesarias para la correcta construcción de las oraciones.

Ante este panorama en que las fases subsiguientes a la determinación de contenido poseen una dependencia excesivamente alta cuando se utilizan los Information Items, se descartó la posibilidad de desarrollar este sistema siguiendo el esquema de arquitectura realizer y se optó por generar la aplicación siguiendo el modelo transformer.

3.4. Implementación de la arquitectura transformer

A diferencia de los problemas que se encuentran al tratar de construir una arquitectura realizer para un problema sin dominio específico como es la redacción automática de correos electrónicos, los transformers son capaces de afrontar la generación de lenguaje natural sin la necesidad de enmarcar los textos dentro de un ámbito concreto. De hecho, normalmente se suele acudir a esta arquitectura cuando se pretende hacer frente a problemas en los que podría requerirse un conocimiento general.

Como en el apartado 2.2.2.2, se entró bastante en detalle en la estructura de la arquitectura, cada uno de sus módulos y la filosofía detrás de cada uno de ellos, esta sección se limitará a exponer las vicisitudes específicas del problema que compete a este trabajo, empezando por la entrada de la aplicación.

La entrada del modelo debe ser un conjunto de no más de seis Information Items construidos como tuplas sujeto-verbo-objeto. Sin embargo, esto no coincide del todo con la definición de la entrada del modelo transformer. Para resolverlo, en primer lugar se tomará la lista de todos los InIts y se tokenizarán por separado. La tarea de tokenización se lleva a cabo utilizando un tokenizador preentrenado⁸.

Con una lista de las tuplas Inits tokenizadas, se concatenan como si constituyeran un solo tensor. Esto podría generar la preocupación de que es necesario antes homogeneizar los tensores asegurándose de que todos los sujetos poseen la misma longitud mediante la técnica de *padding* (todos los tensores se adaptan a la longitud del mayor de ellos rellenando con ceros las últimas dimensiones), y lo mismo con los verbos y objetos. No obstante, además de ser una operación computacionalmente costosa, no es necesario para el correcto funcionamiento de nuestro modelo. El motivo por el que se pueden concatenar los InIts (sobra decir que siempre todos ellos deben seguir el orden sujeto-verbo-objeto, ya que esto sí podría dificultar el entrenamiento y confundir al modelo) y luego hacer padding es que, como se explicó en la sección 2.2.2.2, existen tokens especiales de inicio y fin. Estos

⁸https://www.tensorflow.org/text/api_docs/python/text/BertTokenizer

se añaden a cada sujeto, verbo y objeto por separado, quedando así todos los elementos claramente delimitados. Es decir, la red neuronal aprenderá a distinguir dónde acaba un elemento de la tupla InIt y dónde comienza el siguiente sin necesidad de incluir ceros entre ellos. Este hecho, no solo ahorra tiempo computacionalmente hablando, sino que también ahorra en memoria, pues los vectores tokenizados de InIts son notablemente más pequeños, y, al reducir el tamaño de los vectores de Information Items, disminuye la entrada de la red y, por ende, el número de parámetros a entrenar.

Tras construir la tokenización de los Information Items, se tokeniza, con el mismo tokenizador preentrenado, el correo electrónico “resultante”. Así ya se cuenta con la entrada, el tensor concatenado de InIts, y la salida de la red, el tensor del cuerpo del mensaje tokenizado. De esta manera, se está en disposición de entrenar el modelo y evaluar los resultados obtenidos.

3.5. Evaluación de los resultados

Para este modelo se ha utilizado el 80 % de los datos para el entrenamiento y el 20 % restante para el testeo, es decir, 185030 correos electrónicos con sus respectivos Information Items han entrenado el modelo y 46258 se han empleado para determinar la precisión del mismo.

A pesar de contar con un número tan elevado de correos electrónicos, en el entrenamiento se consiguió una precisión máxima de 0.4183 y en el conjunto de testeo de 0.2148. Hay que tener en cuenta que estos resultados se calculan como la probabilidad de predecir correctamente la próxima palabra dada una entrada y el texto generado en las iteraciones anteriores.

Una posible explicación que se le puede dar a estos resultados tan bajos es que, la entrada de los InIts no brinda suficiente contenido semántico como para generar todo el cuerpo del correo electrónico, es decir, el modelo transformer también se ve afectado por la dificultad que se encontraba en la arquitectura realizer de añadir información extra no incluida en los Information Items. A diferencia de otros casos de uso, como la traducción automática, en los que el modelo de atención sí puede hacer una correspondencia casi directa entre la entrada y la salida, esto no ocurre facilitándole como entrada estas tuplas sujeto-verbo-objeto, ya que la correspondencia de atención apenas se encuentra adecuadamente correspondida por contener la salida información semántica extra que resulta difícil generar sin dicho conocimiento.

Por lo tanto, aunque no debe descartarse la arquitectura transformer para este caso de uso, sí que conviene replantearse la definición de estas representaciones semánticas de entrada que quizás requieran ser más complejas para lograr unos resultados más satisfactorios.

Capítulo 4

Conclusiones y Trabajo Futuro

“Difícil de ver es. Siempre en movimiento el futuro está”
— Yoda - Star Wars: Episodio III - La venganza de los Sith
(2005)

Tras el desarrollo de este trabajo, en este capítulo se presentan las conclusiones que pueden extraerse de este estudio (se explican en la sección 4.1). justo después, las posibles opciones para la continuación de este trabajo quedan expuestas en la sección 4.2, con el fin de continuar con el estudio de la generación de lenguaje natural a partir de representaciones semánticas.

4.1. Conclusiones

Hoy en día, el correo electrónico es un sistema de comunicación que se utiliza tanto en el ámbito profesional como en el personal. A través de él se establecen conversaciones sobre el trabajo, los estudios, relaciones íntimas, etcétera. Sin embargo, el gran número de e-mails que se reciben y envían cada día, comienza a obligar a sus usuarios a dedicar una notable cantidad de tiempo a atender su bandeja de entrada y pensar cuál es la mejor forma de redactar los mensajes, para transmitir la idea que se tiene en mente. Pero, ¿y si se pudiera escribir de forma automática con tan solo introducir como entrada dicha idea o concepto sobre el que giraría el cuerpo del mensaje? Este problema puede ser resuelto mediante el uso de técnicas de generación de lenguaje natural, un campo de la inteligencia artificial que se enfrenta al reto de producir textos imitando la forma en que los humanos se comunican entre sí. Dentro de esta rama, destacan dos tipos de arquitecturas: los modelos realizer y los modelos transformer. La primera aproximación consiste en un pipeline de tareas que, poco a poco, construyen el texto de salida; mientras que la segunda hace uso de los modelos de atención y las arquitecturas codificador-decodificador de deep learning. En este trabajo se trata de evaluar la viabilidad de implementar cada una de ellas para enfrentarse al problema de redacción automática de correos electrónicos, desarrollar la solución y valorar los resultados obtenidos.

Para representar esa idea que el usuario posee, se hace uso de los llamados Information Items, entidades que almacenan la mínima información semántica. Este elemento proviene de la rama del resumen automático de textos. Los Information Items serán la entrada del sistema desarrollado, representando ese concepto que posee el usuario acerca de lo que quiere redactar en el correo electrónico. Concretamente, se implementan mediante tuplas sujeto-verbo-objeto. Sin embargo, como se ha mostrado en el trabajo, esta definición no

funciona adecuadamente en ninguna de las dos arquitecturas, ya que obliga al sistema de generación de lenguaje natural a producir construcciones lingüísticas con elementos semánticos no transmitidos por el usuario y que no puede obtener de otra manera.

En lo que se refiere al modelo *realizer*, las tuplas sujeto-verbo-objeto, solventan el problema de construcción de estructuras ad hoc para hacer frente a la fase de determinación del contenido, es decir, permiten implementar dicha tarea a pesar de que los correos electrónicos no se enmarquen en uno o varios dominios específico. No obstante, como en esta arquitectura el resto de fases poseen una alta dependencia de la salida de la determinación del contenido, que serían los *Information Items*, el texto producido se restringe a cada uno de ellos sin poder aportar más información semántica. Esto se debe a que no se cuenta con una base de conocimiento o módulo de razonamiento general de los que sacar dicha información extra que se pueda añadir al texto producido.

Respecto al modelo *transformer*, podría ocurrir que se añadiera información extra que aumentara el tamaño y la riqueza del cuerpo del correo electrónico. Sin embargo, al no poseer más contexto que los *Information Items*, si se incluyera, se trataría de estructuras lingüísticas basadas en la frecuencia de aparición del resto de correos electrónicos. Esto, limita enormemente la salida del sistema, así que requiere una inmensa cantidad de documentos en el corpus, mayor de la que se tiene, para poder producir el texto como esperaría el usuario final. Todas estas razones justifican los resultados obtenidos con dicha arquitectura y demuestran que la aproximación con esta definición de las representaciones semánticas que constituyen los *Information Items*, no permite alcanzar una generación de lenguaje natural de buena calidad, coherencia y cohesión.

4.2. Trabajo futuro

En vista de los resultados obtenidos, la principal vía de trabajo futuro es el estudio de la redacción automática de correos electrónicos utilizando otras implementaciones más complejas de los *Information Items*. La clave reside en que almacenen la suficiente información como para ser capaces de generar por completo el mensaje, pero no excesiva como para que el usuario tenga que redactar prácticamente todo el texto. Varias alternativas se han propuesta en el capítulo 2, que pueden ser estudiadas individualmente o en conjunto. Por ejemplo, sería posible combinar el etiquetado del rol semántico con técnicas de desambiguación del sentido de las palabras.

No obstante, no se debe descartar la posibilidad de que quizás, con un mayor conjunto de entrenamiento, la propuesta desarrollada a lo largo de este trabajo obtenga resultados satisfactorios. Por esa razón, otra opción para continuar el trabajo desarrollado es la reutilización de los módulos implementados (que pueden encontrarse en el repositorio correspondiente¹) y entrenarlos con un corpus mayor que permite exprimir la utilidad de los InIts.

Por otro lado, es posible estudiar mejoras de cara a la arquitectura *realizer* con las que quizás, se podrían obtener resultados más satisfactorios. Estas mejoras van desde la reformulación de los métodos de redacción de pequeñas partes del mensaje, como la implementación de un modelo probabilístico para la generación automática del saludo en los correos electrónicos, hasta el desarrollo de un sistema de razonamiento con el que deducir otros *Information Items* e incluirlos al cuerpo del mensaje.

En cuanto a trabajo futuro en la arquitectura *transformer*, es razonable pensar que la división del problema de redacción en distintas partes, como lo que se ha propuesto

¹<https://github.com/carlosmmorera/NLG-AI-Master-Thesis>

antes de redactar fragmentos del e-mail, pueda concluir en resultados más satisfactorios. Además, también podría estudiarse la posibilidad de incluir más información de entrada, como los metadatos del correo electrónico (destinatario, asunto, etcétera) para extraer de ellos información extra que se pueda incluir en la redacción.

Estas son algunas vías posibles de trabajo futuro, de cara al estudio del caso de uso de redacción automática de correos electrónicos utilizando técnicas avanzadas de inteligencia artificial y, en concreto, del campo de la generación de lenguaje natural.

Bibliografía

Y así, del mucho leer y del poco dormir, se le secó el cerebro de manera que vino a perder el juicio.

Miguel de Cervantes Saavedra

ABADI, M., BARHAM, P., CHEN, J., CHEN, Z., DAVIS, A., DEAN, J., DEVIN, M., GHEMWAT, S., IRVING, G., ISARD, M. ET AL. Tensorflow: A system for large-scale machine learning. En *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, páginas 265–283. 2016.

ANGELI, G., MANNING, C. D. y JURAFSKY, D. Parsing time: Learning to interpret time expressions. En *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, páginas 446–455. 2012.

BANAE, H., AHMED, M. U. y LOUTFI, A. Towards nlg for physiological data monitoring with body area networks. En *14th European Workshop on Natural Language Generation, Sofia, Bulgaria, August 8-9, 2013*, páginas 193–197. 2013.

BANGALORE, S. y RAMBOW, O. Corpus-based lexical choice in natural language generation. En *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, páginas 464–471. 2000.

BANNARD, C. y CALLISON-BURCH, C. Paraphrasing with bilingual parallel corpora. En *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, páginas 597–604. 2005.

BARTOLI, A., DE LORENZO, A., MEDVET, E. y TARLAO, F. Your paper has been accepted, rejected, or whatever: Automatic generation of scientific paper reviews. En *International Conference on Availability, Reliability, and Security*, páginas 19–28. Springer, 2016.

BARZILAY, R. y MCKEOWN, K. R. Sentence fusion for multidocument news summarization. *Computational Linguistics*, vol. 31(3), páginas 297–328, 2005.

BAUTISTA, S., LEÓN, C., HERVÁS, R. y GERVÁS, P. Empirical identification of text simplification strategies for reading-impaired people. En *Everyday Technology for Independence and Care*, páginas 567–574. IOS Press, 2011.

- BELZ, A., KOW, E., VIETHEN, J. y GATT, A. Generating referring expressions in context: The grec task evaluation challenges. En *Empirical methods in natural language generation*, páginas 294–327. Springer, 2009.
- BENGIO, Y., DUCHARME, R., VINCENT, P. y JANVIN, C. A neural probabilistic language model. *The journal of machine learning research*, vol. 3, páginas 1137–1155, 2003.
- BOHNET, B., WANNER, L., MILLE, S. y BURGA, A. Broad coverage multilingual deep sentence generation with a stochastic multi-level realizer. En *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, páginas 98–106. 2010.
- BORENSTEIN, N. y FREED, N. Mime (multipurpose internet mail extensions) part one: Mechanisms for specifying and describing the format of internet message bodies. Informe Técnico RFC 1521, Internet Engineering Task Force (IETF), 1993.
- BRACHMAN, R. J. y SCHMOLZE, J. G. An overview of the kl-one knowledge representation system. *Readings in artificial intelligence and databases*, páginas 207–230, 1989.
- BROWN, J., FRISHKOFF, G. y ESKENAZI, M. Automatic question generation for vocabulary assessment. En *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, páginas 819–826. 2005.
- CALLAWAY, C. B. y LESTER, J. C. Narrative prose generation. *Artificial Intelligence*, vol. 139(2), páginas 213–252, 2002.
- CHANG, F., DELL, G. S. y BOCK, K. Becoming syntactic. *Psychological review*, vol. 113(2), página 234, 2006.
- CHEN, D. L. y MOONEY, R. J. Learning to sportscast: a test of grounded language acquisition. En *Proceedings of the 25th international conference on Machine learning*, páginas 128–135. 2008.
- CHEN, Q., ZHU, X., LING, Z., WEI, S., JIANG, H. y INKPEN, D. Enhanced lstm for natural language inference. *arXiv preprint arXiv:1609.06038*, 2016.
- CHENG, H. y MELLISH, C. Capturing the interaction between aggregation and text planning in two generation systems. En *INLG'2000 Proceedings of the First International Conference on Natural Language Generation*, páginas 186–193. 2000.
- CHO, K., VAN MERRIËNBOER, B., GULCEHRE, C., BAHDANAU, D., BOUGARES, F., SCHWENK, H. y BENGIO, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- CHOI, J. D., TETREAULT, J. y STENT, A. It depends: Dependency parser comparison using a web-based evaluation tool. En *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, páginas 387–396. 2015.
- CHUI, M., MANYIKA, J., BUGHIN, J., DOBBS, R., ROXBURGH, C., SARRAZIN, H., SANDS, G. y WESTERGREN, M. The social economy: Unlocking value and productivity through social technologies. *McKinsey Global Institute*, 2012.

- COHN, T. A. y LAPATA, M. Sentence compression as tree transduction. *Journal of Artificial Intelligence Research*, vol. 34, páginas 637–674, 2009.
- CREMERS, A. y GINSBURG, S. Context-free grammar forms. *Journal of Computer and System Sciences*, vol. 11(1), páginas 86–117, 1975.
- CRISPIN, M. Internet message access protocol - version 4rev1. Informe Técnico RFC 3501, University of Washington, 2003.
- CROCKER, D. H. Standard for the format of arpa internet text messages. Informe Técnico RFC 822, Dept. of Electrical Engineering, University of Delaware, 1982.
- DALIANIS, H. Aggregation in natural language generation. *Computational Intelligence*, vol. 15(4), páginas 384–414, 1999.
- DEEMTER, K. v., THEUNE, M. y KRAHMER, E. Real versus template-based natural language generation: A false opposition? *Computational linguistics*, vol. 31(1), páginas 15–24, 2005.
- DONG, D., WU, H., HE, W., YU, D. y WANG, H. Multi-task learning for multiple language translation. En *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, páginas 1723–1732. 2015.
- DUŠEK, O. y JURČÍČEK, F. Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings. *arXiv preprint arXiv:1606.05491*, 2016.
- DUŠEK, O., NOVIKOVA, J. y RIESER, V. Evaluating the state-of-the-art of end-to-end natural language generation: The e2e nlg challenge. *Computer Speech & Language*, vol. 59, páginas 123–156, 2020.
- EDMONDS, P. y HIRST, G. Near-synonymy and lexical choice. *Computational linguistics*, vol. 28(2), páginas 105–144, 2002.
- ELHADAD, M. y ROBIN, J. An overview of surge: A reusable comprehensive syntactic realization component. 1996.
- ELMAN, J. L. Finding structure in time. *Cognitive science*, vol. 14(2), páginas 179–211, 1990.
- ELMAN, J. L. Learning and development in neural networks: The importance of starting small. *Cognition*, vol. 48(1), páginas 71–99, 1993.
- EPSTEIN, R. L. Predicate logic. Advanced Reasoning Forum, 2018.
- ESPINOSA, D., WHITE, M. y MEHAY, D. Hypertagging: Supertagging for surface realization with ccg. En *Proceedings of ACL-08: HLT*, páginas 183–191. 2008.
- FENSEL, D. Ontologies. En *Ontologies*, páginas 11–18. Springer, 2001.
- FERREIRA, T. C., CALIXTO, I., WUBBEN, S. y KRAHMER, E. Linguistic realisation as machine translation: Comparing different mt models for amr-to-text generation. En *Proceedings of the 10th International Conference on Natural Language Generation*, páginas 1–10. 2017.

- FLEISCHMAN, M. y HOVY, E. Towards emotional variation in speech-based natural language processing. En *Proceedings of the International Natural Language Generation Conference*, páginas 57–64. 2002.
- FREED, N. y BORENSTEIN, N. Mime (multipurpose internet mail extensions). Informe Técnico RFC 1341, Internet Engineering Task Force (IETF), 1992.
- FREED, N. y BORENSTEIN, N. Multipurpose internet mail extensions (mime) part five: Conformance criteria and examples. Informe Técnico RFC 2049, Internet Engineering Task Force (IETF), 1996a.
- FREED, N. y BORENSTEIN, N. Multipurpose internet mail extensions (mime) part one: Format of internet message bodies. Informe Técnico RFC 2045, Internet Engineering Task Force (IETF), 1996b.
- FREED, N. y BORENSTEIN, N. Multipurpose internet mail extensions (mime) part two: Media types. Informe Técnico RFC 2046, Internet Engineering Task Force (IETF), 1996c.
- FREED, N. y KLENSIN, J. Media type specifications and registration procedures. Informe Técnico RFC 4288, Internet Engineering Task Force (IETF), 2005a.
- FREED, N. y KLENSIN, J. Multipurpose internet mail extensions (mime) part four: Registration procedures. Informe Técnico RFC 4289, Internet Engineering Task Force (IETF), 2005b.
- GARDENT, C. y NARAYAN, S. Multiple adjunction in feature-based tree-adjoining grammar. *Computational Linguistics*, vol. 41(1), páginas 41–70, 2015.
- GATT, A. y KRAHMER, E. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, vol. 61, páginas 65–170, 2018.
- GENEST, P.-E. y LAPALME, G. Text generation for abstractive summarization. En *TAC*. 2010.
- GENEST, P.-E. y LAPALME, G. Framework for abstractive summarization using text-to-text generation. En *Proceedings of the workshop on monolingual text-to-text generation*, páginas 64–73. 2011.
- GENEST, P.-E., LAPALME, G. y YOUSFI-MONOD, M. Hextac: the creation of a manual extractive run. *Génération de résumés par abstraction*, página 7, 2013.
- GERVÁS, P. Composing narrative discourse for stories of many characters: a case study over a chess game. *Literary and Linguistic Computing*, vol. 29(4), páginas 511–531, 2014.
- GERVÁS, P., DÍAZ-AGUDO, B., PEINADO, F. y HERVÁS, R. Story plot generation based on cbt. En *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, páginas 33–46. Springer, 2004.
- GOLDBERG, E., DRIEDGER, N. y KITTREDGE, R. I. Using natural-language processing to produce weather forecasts. *IEEE Expert*, vol. 9(2), páginas 45–53, 1994a.
- GOLDBERG, E., KITTREDGE, R. y DRIEDGER, N. Fog: a new approach to the synthesis of weather forecast text. *IEEE Expert (Special Track on NLP)*, 1994b.

- GOLDBERG, Y. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, vol. 57, páginas 345–420, 2016.
- GONZÁLEZ ÁLVAREZ, S. y LÓPEZ PULIDO, J. M. Traductor de pictogramas a texto. 2019.
- GOODFELLOW, I., BENGIO, Y. y COURVILLE, A. *Deep learning*. MIT press, 2016.
- GUHE, M. Incremental conceptualization for language production. 2007.
- GUIDE, S. *Red Hat Enterprise Linux 4: Reference Guide*. Red Hat Inc., 2005. <http://web.mit.edu/rhel-doc/OldFiles/4/RH-DOCS/rhel-rg-en-4/index.html>.
- GYŐRÖDI, C., GYŐRÖDI, R., PECHERLE, G. y OLAH, A. A comparative study: Mongodb vs. mysql. En *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, páginas 1–6. IEEE, 2015.
- HAHN, U. y MANI, I. The challenges of automatic summarization. *Computer*, vol. 33(11), páginas 29–36, 2000.
- HAN, J., PEI, J. y KAMBER, M. *Data mining: concepts and techniques*. Elsevier, 2011.
- HOCHREITER, S. y SCHMIDHUBER, J. Long short-term memory. *Neural computation*, vol. 9(8), páginas 1735–1780, 1997.
- HONNIBAL, M. y JOHNSON, M. An improved non-monotonic transition system for dependency parsing. En *Proceedings of the 2015 conference on empirical methods in natural language processing*, páginas 1373–1378. 2015.
- HOVY, E. Generating natural language under pragmatic constraints. *Journal of Pragmatics*, vol. 11(6), páginas 689–719, 1987.
- HÜSKE-KRAUS, D. Text generation in clinical medicine—a review. *Methods of information in medicine*, vol. 42(01), páginas 51–60, 2003.
- HUTCHINS, W. J. y SOMERS, H. L. *An introduction to machine translation*. 2009.
- IBRAHIM, M. S., KASIM, S., HASSAN, R., MAHDIN, H., RAMLI, A. A., FUDZEE, M. F. M., SALAMAT, M. A. ET AL. Information technology club management system. *Acta Electronica Malaysia*, vol. 2(2), páginas 01–05, 2018.
- ISLAM, A. y INKPEN, D. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 2(2), páginas 1–25, 2008.
- JOSEFSSON, S. The base16, base32, and base64 data encodings. Informe Técnico RFC 4648, Internet Engineering Task Force (IETF), 2006.
- KENNEDY, C. y McNALLY, L. Scale structure, degree modification and the semantics of gradable predicates. *Language*, 2004.
- KIBBLE, R. y POWER, R. Optimizing referential coherence in text generation. *Computational Linguistics*, vol. 30(4), páginas 401–416, 2004.
- KLENSIN, J. Simple mail transfer protocol. Informe Técnico RFC 5321, Internet Engineering Task Force (IETF), 2008.

- KLIMT, B. y YANG, Y. Introducing the enron corpus. En *CEAS*. 2004.
- KNIGHT, K. y MARCU, D. Statistics-based summarization-step one: Sentence compression. *AAAI/IAAI*, vol. 2000, páginas 703–710, 2000.
- KONDADADI, R., HOWALD, B. y SCHILDER, F. A statistical nlg framework for aggregated planning and realization. En *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, páginas 1406–1415. 2013.
- KUKICH, K. Where do phrases come from: Some preliminary experiments in connectionist phrase generation. En *Natural language generation*, páginas 405–421. Springer, 1987.
- KUKICH, K. Techniques for automatically correcting words in text. *Acm Computing Surveys (CSUR)*, vol. 24(4), páginas 377–439, 1992.
- LABBÉ, C. y PORTET, F. Towards an abstractive opinion summarisation of multiple reviews in the tourism domain. En *The First International Workshop on Sentiment Discovery from Affective Data (SDAD 2012)*, páginas 87–94. 2012.
- LECUN, Y., BENGIO, Y. y HINTON, G. Deep learning. *nature*, vol. 521(7553), páginas 436–444, 2015.
- LUONG, M.-T., LE, Q. V., SUTSKEVER, I., VINYALS, O. y KAISER, L. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.
- LUONG, M.-T., SOCHER, R. y MANNING, C. D. Better word representations with recursive neural networks for morphology. En *Proceedings of the seventeenth conference on computational natural language learning*, páginas 104–113. 2013.
- MACDONALD, I. y SIDDHARTHAN, A. Summarising news stories for children. *ACL*, 2016.
- MANI, I. y MAYBURY, M. T. Automatic summarization. 2001.
- MANN, W. C. y THOMPSON, S. A. *Rhetorical structure theory: A theory of text organization*. University of Southern California, Information Sciences Institute Los Angeles, 1987.
- MCCOY, K. F. y STRUBE, M. Generating anaphoric expressions: Pronoun or definite description? En *The Relation of Discourse/Dialogue Structure and Reference*. 1999.
- MCDONALD, D. D. Issues in the choice of a source for natural language generation. *Computational Linguistics*, vol. 19(1), páginas 191–197, 1993.
- MCINTYRE, N. y LAPATA, M. Learning to tell tales: A data-driven approach to story generation. En *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, páginas 217–225. 2009.
- MCKEOWN, K. *Text generation*. Cambridge University Press, 1992.
- McROY, S. W., CHANNARUKUL, S. y ALI, S. S. An augmented template-based approach to text realization. *Natural Language Engineering*, vol. 9(4), páginas 381–420, 2003.
- MELLISH, C., SCOTT, D., CAHILL, L., PAIVA, D., EVANS, R. y REAPE, M. A reference architecture for natural language generation systems. *Natural language engineering*, vol. 12(1), páginas 1–34, 2006.

- MIKOLOV, T., CHEN, K., CORRADO, G. y DEAN, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.
- MIKOLOV, T., KARAFIÁT, M., BURGET, L., CERNOCKÝ, J. y KHUDANPUR, S. Recurrent neural network based language model. En *Interspeech*, vol. 2, páginas 1045–1048. Makuhari, 2010.
- MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G. S. y DEAN, J. Distributed representations of words and phrases and their compositionality. En *Advances in neural information processing systems*, páginas 3111–3119. 2013b.
- MILLER, G. A. Wordnet: a lexical database for english. *Communications of the ACM*, vol. 38(11), páginas 39–41, 1995.
- MILOSAVLJEVIC, M., TULLOCH, A. y DALE, R. Text generation in a dynamic hypertext environment. *Australian Computer Science Communications*, vol. 18, páginas 417–426, 1996.
- MNIH, A. y HINTON, G. Three new graphical models for statistical language modelling. En *Proceedings of the 24th international conference on Machine learning*, páginas 641–648. 2007.
- MOLINA, M., STENT, A. y PARODI, E. Generating automated news to explain the meaning of sensor data. En *International Symposium on Intelligent Data Analysis*, páginas 282–293. Springer, 2011.
- MOORE, K. Multipurpose internet mail extensions (mime) part three: Message header extensions for non-ascii text. Informe Técnico RFC 2047, Internet Engineering Task Force (IETF), 1996.
- MORENO MORERA, C. Un modelo de análisis estilométrico de correos electrónicos para la redacción personalizada basada en el destinatario. 2020.
- MYERS, J., MELLON, C. y ROSE, M. Post office protocol - version 3. Informe Técnico RFC 1939, Dover Beach Consulting, Inc., 1996.
- NAVIGLI, R. Word sense disambiguation: A survey. *ACM computing surveys (CSUR)*, vol. 41(2), páginas 1–69, 2009.
- NELSON, S. y PARKS, C. The model primary content type for multipurpose internet mail extensions. Informe Técnico RFC 2077, Internet Engineering Task Force (IETF), 1997.
- NENKOVA, A. y McKEOWN, K. *Automatic summarization*. Now Publishers Inc, 2011.
- NG, H. T., WU, S. M., BRISCOE, T., HADIWINOTO, C., SUSANTO, R. H. y BRYANT, C. The conll-2014 shared task on grammatical error correction. En *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, páginas 1–14. 2014.
- OETTINGER, A. G. *Automatic language translation*. Harvard University Press, 2013.
- PAL, A. R. y SAHA, D. An approach to automatic text summarization using wordnet. En *2014 IEEE international advance computing conference (IACC)*, páginas 1169–1173. IEEE, 2014.

- PALMER, M., GILDEA, D. y XUE, N. Semantic role labeling. *Synthesis Lectures on Human Language Technologies*, vol. 3(1), páginas 1–103, 2010.
- PAREKH, R., HONAVAR, V. ET AL. Grammar inference, automata induction, and language acquisition. *Handbook of natural language processing*, páginas 727–764, 2000.
- PARKER, P. M. *The Official Patient's Sourcebook on Spinal Stenosis*. Icon Group International Incorporated, 2002.
- PARKER, P. M. *The 2007-2012 Outlook for Tufted Washable Scatter Rugs, Bathmats, and Sets That Measure 6-Feet by 9-Feet or Smaller in India*. Icon Group International Incorporated, 2006.
- PARKER, P. M. *The 2009-2014 World Outlook for 60-milligram Containers of Fromage Frais*. Icon Group International Incorporated, 2008a.
- PARKER, P. M. *Webster's Quechua - English Thesaurus Dictionary*. Icon Group International Incorporated, 2008b.
- PEDERSEN, T., PATWARDHAN, S., MICHELIZZI, J. ET AL. Wordnet:: Similarity-measuring the relatedness of concepts. En *AAAI*, vol. 4, páginas 25–29. 2004.
- PENNINGTON, J., SOCHER, R. y MANNING, C. D. Glove: Global vectors for word representation. En *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, páginas 1532–1543. 2014.
- PLACHOURAS, V., SMILEY, C., BRETZ, H., TAYLOR, O., LEIDNER, J. L., SONG, D. y SCHILDER, F. Interacting with financial data using natural language. En *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, páginas 1121–1124. 2016.
- PLAZA, L., DÍAZ, A. y GERVÁS, P. Automatic summarization of news using wordnet concept graphs. *IADIS International Journal on Computer Science and Information Systems*, vol. 5(1), páginas 45–57, 2010.
- POESIO, M., STEVENSON, R., EUGENIO, B. D. y HITZEMAN, J. Centering: A parametric theory and its instantiations. *Computational linguistics*, vol. 30(3), páginas 309–363, 2004.
- PONNAMPERUMA, K., SIDDHARTHAN, A., ZENG, C., MELLISH, C. y VAN DER WAL, R. Tag2blog: Narrative generation from satellite tag data. En *Proceedings of the 51st annual meeting of the association for computational linguistics: System demonstrations*, páginas 169–174. 2013.
- PORTEL, F., REITER, E., GATT, A., HUNTER, J., SRIPADA, S., FREER, Y. y SYKES, C. Automatic generation of textual summaries from neonatal intensive care data. *Artificial Intelligence*, vol. 173(7-8), páginas 789–816, 2009.
- POSTEL, J. B. Simple mail transfer protocol. Informe Técnico RFC 821, Information Sciences Institute, University of Southern California, 1982.
- RADICATI, S. y LEVENSTEIN, J. Email statistics report, 2015-2019. *The Radicati Group, INC., A Technology Market Research Firm, Palo Alto, CA, USA, Tech. Rep, February*, 2015.

- RADICATI, S. y LEVENSTEIN, J. Email statistics report, 2021-2025. *The Radicati Group, INC., A Technology Market Research Firm, Palo Alto, CA, USA, Tech. Rep, February, 2021.*
- RAMOS-SOTO, A., BUGARIN, A. J., BARRO, S. y TABOADA, J. Linguistic descriptions for automatic generation of textual short-term weather forecasts on real prediction data. *IEEE Transactions on Fuzzy Systems*, vol. 23(1), páginas 44–57, 2014.
- REAPE, M. y MELLISH, C. Just what is aggregation anyway. En *Proceedings of the 7th European Workshop on Natural Language Generation*, páginas 20–29. Citeseer, 1999.
- REITER, E. y DALE, R. *Building Natural Language Generation Systems*. Cambridge University Press, 2000. ISBN 9780521620369.
- REITER, E., MELLISH, C. y LEVINE, J. Automatic generation of technical documentation. *Applied Artificial Intelligence an International Journal*, vol. 9(3), páginas 259–287, 1995.
- REITER, E., ROBERTSON, R. y OSMAN, L. Types of knowledge required to personalise smoking cessation letters. En *Joint European Conference on Artificial Intelligence in Medicine and Medical Decision Making*, páginas 389–399. Springer, 1999.
- REITER, E., SRIPADA, S., HUNTER, J., YU, J. y DAVY, I. Choosing words in computer-generated weather forecasts. *Artificial Intelligence*, vol. 167(1-2), páginas 137–169, 2005.
- RESNICK, P. Internet message format. Informe Técnico RFC 5322, Qualcomm Incorporated, 2008.
- REYNOLDS, J. K. Post office protocol. Informe Técnico RFC 918, Information Sciences Institute, 1984.
- SCHWENK, H. y GAUVAIN, J.-L. Training neural network language models on very large corpora. En *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, páginas 201–208. 2005.
- SEGAL, E. Survey finds email fatigue could lead 38 % of workers to quit their jobs. *Forbes*, 2021.
- SHAW, J. Clause aggregation using linguistic knowledge. En *Natural Language Generation*. 1998.
- SIDDHARTHAN, A. A survey of research on text simplification. *ITL-International Journal of Applied Linguistics*, vol. 165(2), páginas 259–298, 2014.
- SIDDHARTHAN, A., GREEN, M. J., VAN DEEMTER, K., MELLISH, C. S. y VAN DER WAL, R. Blogging birds: Generating narratives about reintroduced species to promote public engagement. En *Proceedings of the 7th International Natural Language Generation Conference (INLG 2012)*. ACL Anthology, 2012.
- SIDDHARTHAN, A., NENKOVA, A. y MCKEOWN, K. Information status distinctions and referring expressions: An empirical study of references to people in news summaries. *Computational Linguistics*, vol. 37(4), páginas 811–842, 2011.
- DEL SOCORRO BERNARDOS, M. ¿qué es la generación de lenguaje natural? una visión general sobre el proceso de generación. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, vol. 11(34), páginas 105–128, 2007.

- SOON, W. M., NG, H. T. y LIM, D. C. Y. A machine learning approach to coreference resolution of noun phrases. *Computational linguistics*, vol. 27(4), páginas 521–544, 2001.
- STEDE, M. The hyperonym problem revisited: Conceptual and lexical hierarchies in language generation. En *INLG'2000 Proceedings of the First International Conference on Natural Language Generation*, páginas 93–99. 2000.
- STEEDMAN, M. *The syntactic process*. Cambridge, MA: MIT press, 2000.
- STENT, A., PRASAD, R. y WALKER, M. Trainable sentence planning for complex information presentations in spoken dialog systems. En *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, páginas 79–86. 2004.
- STOCK, O., ZANCANARO, M., BUSETTA, P., CALLAWAY, C., KRÜGER, A., KRUPPA, M., KUFLIK, T., NOT, E. y ROCCHI, C. Adaptive, intelligent presentation of information for the museum visitor in peach. *User Modeling and User-Adapted Interaction*, vol. 17(3), páginas 257–304, 2007.
- SUTSKEVER, I., MARTENS, J. y HINTON, G. E. Generating text with recurrent neural networks. En *ICML*. 2011.
- SUTSKEVER, I., VINYALS, O. y LE, Q. V. Sequence to sequence learning with neural networks. En *Advances in neural information processing systems*, páginas 3104–3112. 2014.
- TANAKA, H., KINOSHITA, A., KOBAYAKAWA, T., KUMANO, T. y KATO, N. Syntax-driven sentence revision for broadcast news summarization. En *Proceedings of the 2009 Workshop on Language Generation and Summarisation (UCNLG+ Sum 2009)*, páginas 39–47. 2009.
- TANG, J., YANG, Y., CARTON, S., ZHANG, M. y MEI, Q. Context-aware natural language generation with recurrent neural networks. *arXiv preprint arXiv:1611.09900*, 2016.
- THEUNE, M., KLABBERS, E., DE PIJPER, J.-R., KRAHMER, E. y ODIJK, J. From data to speech: a general approach. *Natural Language Engineering*, vol. 7(1), páginas 47–86, 2001.
- THOMASON, J., VENUGOPALAN, S., GUADARRAMA, S., SAENKO, K. y MOONEY, R. Integrating language and vision to generate natural language descriptions of videos in the wild. En *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, páginas 1218–1227. 2014.
- THOMPSON, K. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, vol. 11(6), páginas 419–422, 1968.
- TROOST, R., DORNER, S. y MOORE, K. Communicating presentation information in internet messages: The content-disposition header field. Informe Técnico RFC 2183, Internet Engineering Task Force (IETF), 1997.
- TURNER, R., SRIPADA, S., REITER, E. y DAVY, I. P. Selecting the content of textual descriptions of geographically located events in spatio-temporal weather data. En *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, páginas 75–88. Springer, 2007.
- VAN DEEMTER, K. *Not exactly: In praise of vagueness*. Oxford University Press, 2012.

- VASWANI, A., SHAZER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, Ł. y POLOSUKHIN, I. Attention is all you need. En *Advances in neural information processing systems*, páginas 5998–6008. 2017.
- VICENTE, M., BARROS, C., PEREGRINO, F. S., AGULLÓ, F. y LLORET, E. La generación de lenguaje natural: análisis del estado actual. *Computación y Sistemas*, vol. 19(4), páginas 721–756, 2015.
- VAN DER WAL, R., SHARMA, N., MELLISH, C., ROBINSON, A. y SIDDHARTHAN, A. The role of automated feedback in training and retaining biological recorders for citizen science. *Conservation Biology*, vol. 30(3), páginas 550–561, 2016.
- WALKER, M., RAMBOW, O. y ROGATI, M. Spot: A trainable sentence planner. En *Second Meeting of the North American Chapter of the Association for Computational Linguistics*. 2001.
- WANNER, L., BOSCH, H., BOUAYAD-AGHA, N., CASAMAYOR, G., ERTL, T., HILBRING, D., JOHANSSON, L., KARATZAS, K., KARPPINEN, A., KOMPATSIARIS, I. ET AL. Getting the environmental information across: from the web to the user. *Expert Systems*, vol. 32(3), páginas 405–432, 2015.
- WILLIAMS, S. y REITER, E. Generating basic skills reports for low-skilled readers. *Natural Language Engineering*, vol. 14(4), páginas 495–525, 2008.
- XU, K., BA, J., KIROS, R., CHO, K., COURVILLE, A., SALAKHUDINOV, R., ZEMEL, R. y BENGIO, Y. Show, attend and tell: Neural image caption generation with visual attention. En *International conference on machine learning*, páginas 2048–2057. PMLR, 2015.
- YU, J., REITER, E., HUNTER, J. y MELLISH, C. Choosing the content of textual summaries of large time-series data sets. *Natural Language Engineering*, vol. 13(1), páginas 25–49, 2007.

*-¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.*

*Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes*

*-Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.*

*Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes*

