# Trabajo Fin de Master: Búsqueda de Respuestas sobre la COVID-19

**Trabajo Fin de Master**

**Francisco Xavier Torres Peón**

Trabajo de investigación para el

Máster en Lenguajes y Sistemas Informáticos

Universidad Nacional de Educación a Distancia

Dirigido por el

**Prof. Dr. D. Anselmo Peñas Padilla**

Junio 2021

# Agradecimientos

Me gustaría mostrar mi más sincero agradecimiento a mi tutor el profesor Dr. D. Anselmo Peñas Padilla, así como al resto de profesores del Máster Universitario en Tecnologías del Lenguaje por compartir sus conocimientos y estar ahí siempre que lo he necesitado.

# Resumen

En este trabajo se pretende abordar uno de los desafíos del TAC-2020 consistente en la respuesta a preguntas epidemiológicas (EPIC-QA). El objetivo del desafío EPIC-QA es evaluar la capacidad de los sistemas para proporcionar respuestas oportunas a preguntas sobre la enfermedad COVID-19.

Este desafío esta dividido en dos tareas:

- La **Tarea A** consiste en dar respuesta a las preguntas a nivel de experto. Estas preguntas están dirigidas a las comunidades científicas y medicas.
- La **Tarea B** consiste en dar respuesta a las preguntas a nivel de consumidor. Esta preguntas están dirigidas al publico en general.

En este trabajo solo se va a abordar la Tarea A.

Para realizar este desafío se parte de una colección de artículos biomédicos publicados por el COVID-19 Open Research Dataset Challenge (CORD-19). Este conjunto abierto de datos de investigación fue creado por el Instituto Allen de IA en asociación con grupos de investigación líderes. La evaluación preliminar utiliza una instantánea de CORD-19 del 18 de noviembre de 2020, y consta de 129.069 artículos que se presentan como un único objeto JSON.

Resolver este desafío no es tarea fácil, ya que se pretende extraer información precisa sobre determinadas cuestiones y temas científicos de un conjunto grande de recursos sin etiquetar.

El desafío EPIC-QA requiere que los participantes recuperen sentencias en lugar de documentos o pasajes.

La recuperación de sentencias a gran escala es un proceso difícil para los sistemas neuronales previamente entrenados debido a la falta de contexto circundante. Asimismo, es un proceso computacionalmente costoso.

Se plantea un sistema de tres etapas en cascada. En la primera etapa se usa un sistema no neuronal para recuperar los *k* pasajes mas relevantes para una consulta determinada. Los pasajes mas relevantes se consiguen debido a que el sistema no neuronal puntúa cada resultado obtenido ($score_{RI}$) y en función de esa puntuación obtenemos la relevancia. En una segunda etapa se extraen las sentencias de los pasajes recuperados en la etapa anterior y se usa un sistema neuronal previamente entrenado que clasifica esas sentencias también asignándole una puntuación ($score_{QA}$). La tercera etapa es donde se reclasifican esas sentencias teniendo en cuenta la puntuación obtenida por el sistema no neuronal y la puntuación obtenida por el sistema neuronal.

# Abstract

This work aims to address one of the challenges of the TAC-2020 consisting of the answer to epidemiological questions (EPIC-QA). The goal of the EPIC-QA challenge is to assess the ability of systems to provide timely answers to questions about COVID-19 disease.

This challenge is divided into two tasks:

- **Task A** consists of answering the questions at the expert level. These questions are directed at the scientific and medical communities.
- **Task B** consists of answering the questions at the consumer level. These questions are directed at the general public.

In this work, only Task A will be addressed.

To carry out this challenge, we start from a collection of biomedical articles published by the COVID-19 Open Research Dataset Challenge (CORD-19). This open set of research data was created by the Allen Institute for AI in partnership with leading research groups. The preliminary assessment uses a CORD-19 snapshot from November 18, 2020, and consists of 129,069 articles that are rendered as a single JSON object.

Solving this challenge is not an easy task, as it is intended to extract precise information on certain scientific questions and topics from a large set of unlabeled resources.

The EPIC-QA challenge requires participants to retrieve sentences rather than documents or passages.

Large-scale sentence retrieval is a difficult process for previously trained neural systems due to the lack of surrounding context. It is also a computationally expensive process.

A three-stage cascade system is proposed. In the first stage, a non-neural system is used to retrieve the *k* most relevant passages for a given query. The most relevant passages are obtained because the non-neural system scores each result obtained ($score_{RI}$) and depending on that score we obtain the relevance. In a second stage, the sentences of the passages recovered in the previous stage are extracted and a previously trained neural system is used that classifies these sentences, also assigning them a score ($score_{QA}$). The third stage is where these sentences are reclassified taking into account the score obtained by the non-neural system and the score obtained by the neural system.

# Indice General

# Indice de Figuras

# Indice de Tablas

Capítulo 1

# Introducción

En el año 2019 se detectó una neumonía de origen desconocido en Wuhan, China, y se informó por primera vez a la Organización Mundial de la Salud (OMS) el 31 de Diciembre de 2019.

El 30 de Enero de 2020, el brote se había intensificado hasta el punto de que se declaró una emergencia de salud pública de preocupación internacional.

El 11 de Febrero de 2020, la OMS nombró oficialmente a la enfermedad como coronavirus de 2019 "COVID-19".

Para el 11 de Marzo de 2020, en 114 países, se reportaron más de 118,000 casos que resultaron en más de 4,291 muertes. La OMS caracterizó formalmente al COVID-19 como un pandemia.

En Marzo de 2020, en los Estados Unidos, varios estados y ciudades comenzaron a emitir ordenanzas de cuarentena obligatorias, así como orientación sobre "distanciamiento social" y cierres forzosos de gimnasios, bares y clubes nocturnos.

Hasta el 7 de Abril de 2020, 41 estados habían emitido directivas obligatorias de auto cuarentena, prohibiendo actividades no esenciales fuera del hogar.

## 1.1. Motivación

Durante este período, ha habido una rápida escalada en la investigación científica sobre COVID-19 y coronavirus relacionados, así como la respuesta del gobierno y la comunidad para prevenir o mantener el brote. Por ejemplo, la comunidad científica ha secuenciado el genoma del SARS-CoV-2, ha propuesto múltiples vacunas potenciales y ha explorado tratamientos basados en anticuerpos, antivirales y celulares.

En consecuencia, EPIC-QA en TAC 2020 tiene como objetivo ayudar a reducir la carga de la comunidad científica fomentando la investigación en el diseño de sistemas automáticos de respuesta a preguntas para respaldar la investigación científica.

La motivación de este trabajo reside en que, como la literatura científica sobre COVID19 crece a tal velocidad, los expertos biosanitarios necesitan herramientas para explorarla.

## 1.2. Objetivos y preguntas de investigación

El objetivo de este trabajo es realizar un sistema que permita responder preguntas sobre la COVID19. Para ello se parte de:

- un conjunto de 129.069 artículos biomédicos liberados para el CORD-19 (COVID-19 Open Research Dataset Challenge), y
- un conjunto de 45 preguntas formuladas por expertos.

Como resultado el sistema debe proporcionar una lista clasificada de respuestas a cada pregunta a nivel de experto. Las respuestas deben proporcionar información que sea útil para investigadores, científicos o médicos. Es resumen, se trata de resolver un problema de respuesta a preguntas (QA).

La arquitectura de la solución esta basada en la aplicación secuencial de un módulo IR (Recuperación de Información) y un módulo QA (Respuesta a preguntas).

El módulo IR esta basado en Elasticsearch. A partir de la colección de documentos de expertos se construirán dos índices: uno con los pasajes de los documentos y otro con las sentencias. Este módulo IR realizará búsquedas, en los dos índices anteriores, utilizando para la consulta los tres campos de cada pregunta: `question`, `query` y `background`. Este módulos nos van a aportar una puntuación, $score_{RI}$, que utilizaremos para la clasificación de resultados.

El módulo QA esta basado en Transformers. Para la tarea de Responder a Preguntas, se evaluarán los resultados de aplicar los distintos modelos pre-entrenados. Este módulos nos van a aportar una puntuación, $score_{QA}$, que utilizaremos para la clasificación de resultados.

Con la arquitectura propuesta de Recuperación de Información más Extracción de Respuestas (IR+QA), vamos a responder a las siguientes preguntas:

1. ¿Es mejor indexar y recuperar oraciones (sentencias) o párrafos (pasajes)?

2. A la hora de formular las consultas al sistema, ¿es mejor utilizar palabras clave, una narrativa o la pregunta a contestar?

3. ¿Qué modelo neuronal de extracción de respuestas se adapta mejor a nuestro dominio de aplicación (COVID-19)

4. ¿Cuál es la mejor combinación de evidencias entre el módulo de recuperación y el de extracción de respuestas para realizar un ranking de respuestas?

5. Dado que hay respuestas que provienen del mismo documento, ¿es adecuado fusionarlas en una única respuesta? ¿En qué escenarios?

6. ¿Qué efecto tiene considerar más o menos pasajes en la recuperación como paso previo a la extracción de respuestas?

## 1.3. Estructura del documento

Breve descripción de los capítulos del trabajo.

**Capítulo 1**. Introducción. Este capítulo introduce los principales motivos que han llevado a la realización de este trabajo, así como la problemática y el estado actual de la disciplina.

**Capítulo 2**. Preliminares. Este capítulo describe en mayor detalle la disciplina que nos ocupa, presentando su origen y su historia hasta el presente. Se muestran las técnicas actuales más utilizadas para resolver las tareas más relevantes del tema abordado, así como sus debilidades.

**Capítulo 3**. Arquitectura de la solución. En este capítulo se describe en profundidad la solución propuesta.

**Capítulo 4**. Marco de evaluación. Este capítulo describe la metodología utilizada para evaluar la propuesta realizada, se describen las colecciones de evaluación y líneas base.

**Capítulo 5**. Experimentación y resultados. Se presentan los resultados obtenidos al evaluar el método propuesto en diferentes tareas (consultas con `question`, `query` o `background`), sobre colecciones de evaluación de distintos dominios (pasajes y sentencias) y distintos modelos.

**Capítulo 6**. Conclusiones y trabajo futuro. Este capítulo recopila las diferentes conclusiones extraídas del trabajo realizado, y propone algunas líneas de trabajo futuro.

Capítulo 2

# Preliminares

El objetivo de este capítulo es el de dar una visión global de conceptos que forman parte de las bases de este trabajo. Se describe cada uno de ellos y se muestra su estado de desarrollo actual. Se empieza con una explicación general del concepto de Recuperación de Información (RI) y de Elasticsearch aplicado a la Recuperación de Información. Se continúa con una explicación del sistema de Búsqueda de Respuestas (Question-Answering, QA) y Transformers. Y se termina con una descripción del uso de Transformers aplicado a la Búsqueda de Respuestas.

## 2.1. Recuperación de información (RI)

La recuperación de información, llamada en inglés Information Retrieval (IR), es la ciencia de la búsqueda de información en documentos electrónicos y cualquier tipo de colección documental digital, encargada de la búsqueda dentro de estos mismos, búsqueda de metadatos que describan documentos, o también la búsqueda en bases de datos relacionales, ya sea a través de internet, una intranet, y como objetivo realiza la recuperación en textos, imágenes, sonido o datos de otras características, de manera pertinente y relevante (fuente Wikipedia). Por ejemplo, los motores de búsqueda Web son las aplicaciones RI más conocidas. Se cree que la recuperación de información es la forma dominante de acceso a la información.

### 2.1.1- Sistema de recuperación de información

Un sistema de RI es un sistema software que tiene la capacidad de representar, almacenar, organizar y acceder a elementos de información. Es necesario un conjunto de palabras clave para realizar la búsqueda. Estas palabras clave resumen la descripción de la información.

El sistema de RI ayuda a los usuarios a encontrar la información que necesitan, pero no devuelve explícitamente las respuestas a la pregunta. Notifica sobre la existencia y ubicación de documentos que pudieran contener la información requerida. La recuperación de información también amplía el apoyo a los usuarios para examinar o filtrar la recopilación de documentos o procesar un conjunto de documentos recuperados.

El sistema de recuperación de información también consta de dos componentes: el sistema de indexación y el sistema de consulta. El primer componente se encarga de analizar los documentos y de la creación de índices que luego permiten realizar consultas de búsqueda. El segundo componente es la interfaz visible del motor de búsqueda, es decir, la parte con la que interactúan los usuarios.

Un modelo de recuperación de información (RI) selecciona y clasifica el documento que el usuario ha solicitado en forma de consulta. Los documentos y las consultas se representan de manera similar, de modo que la selección y clasificación de documentos se pueden formalizar mediante una función de coincidencia que devuelve un valor de estado de recuperación (RSV) para cada documento de la colección. Muchos de los sistemas de recuperación de información representan el contenido del documento mediante un conjunto de descriptores, denominados términos, que pertenecen a un vocabulario V.

Para recuperar de forma eficaz los documentos relevantes, mediante estrategias de RI, tendremos que transformar previamente los documentos en una representación adecuada. Cada estrategia de recuperación incorpora un modelo específico para sus propósitos de representación de documentos. La Figura 2.1 muestra la relación de algunos modelos comunes. Los modelos están categorizados según dos dimensiones: la base matemática y las propiedades del modelo.

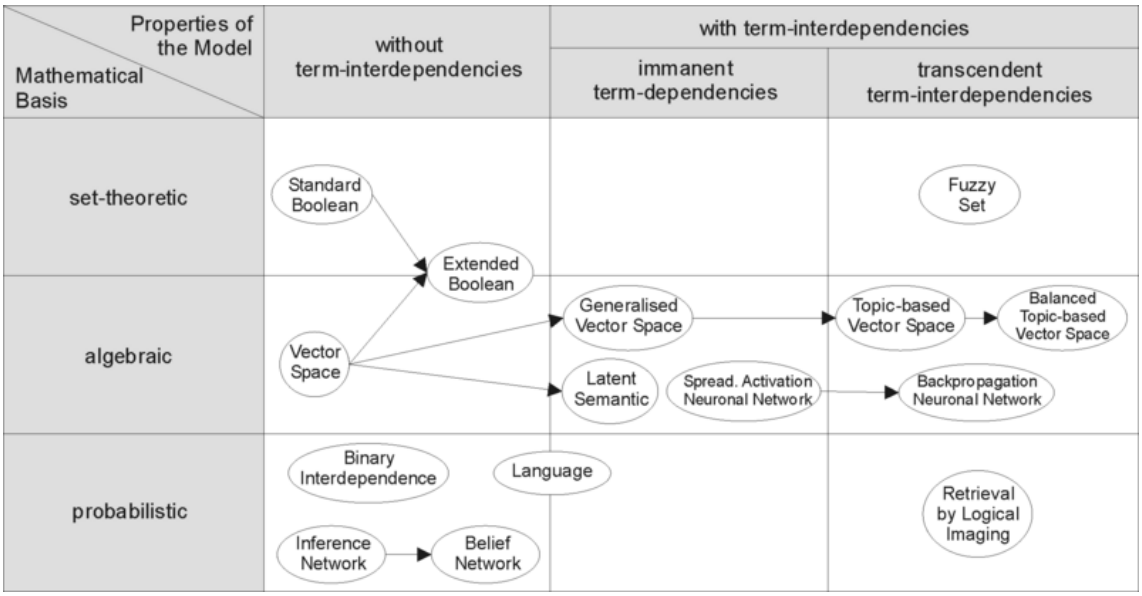



**Figura 2.1 - Modelos de Recuperación de Información (Wikipedia)**

### 2.1.2. Elasticsearch

Como sistema de RI se va a utilizar Elasticsearch. Elasticsearch es un motor de búsqueda y análisis orientado a documentos, distribuido y con capacidad de multitenencia con una interfaz web RESTful. Un índice es como una base de datos en una base de datos relacional. Cada registro en Elasticsearch es un documento JSON estructurado. En otras palabras, cada dato que se envía a Elasticsearch para su indexación es un documento JSON. Todos los campos de los documentos están indexados de forma predeterminada y estos campos indexados se pueden usar en una sola consulta.

Elasticsearch utiliza la biblioteca Apache Lucene para escribir y leer los datos del índice. De hecho, Apache Lucene es la base de Elasticsearch. Está desarrollado en Java y está publicado como código abierto bajo las condiciones de la licencia Apache.

### 2.1.3. Indexado en Elasticsearch

Cada documento enviado a Elasticsearch se almacena en Apache Lucene y la biblioteca almacena todos los datos en una estructura de datos llamada índice invertido. Un índice

invertido es una estructura de datos que se asigna a documentos y términos. Eso significa que un índice invertido tiene una lista de todas las palabras únicas que aparecen en cualquier documento. Además, tiene una lista de documentos en los que aparece cada palabra única recopilada. El índice invertido es un algoritmo de indexación básico utilizado por los motores de búsqueda. Diseñado con esta estructura de datos, el rendimiento de la búsqueda rápida de texto completo se realiza a bajo costo.

Todos los datos de Apache Lucene se almacenan en un índice invertido. Esta transformación es necesaria para que Elasticsearch responda correctamente a las solicitudes de búsqueda. El proceso de transformación de estos datos se llama análisis. Elasticsearch tiene un *módulo de análisis de índices*. El proceso de análisis consiste en tokenizar (dividir) un texto en términos individuales para ser usados en un índice invertido. Esto sucede dos veces:

- En el momento de la indexación
- A la hora de buscar

Un analizador tiene los siguientes componentes:

- *Filtro de caracteres*: (cero o mas) Cambia caracteres del texto o los elimina. Por ejemplo cambia & por “and”, o elimina etiquetas HTML etc.
- *Tokenizer*: (exactamente uno) Divide la cadena en otros términos utilizando espacios en blanco o signos de puntuación. Genera un conjunto de tokens que luego se usan para construir el indice invertido.
- *Token filters*: (cero o mas) Puede añadir, borrar o cambiar tokens. Por ejemplo cambiar de mayusculas a minúsculas, elimina términos como las stopwords (palabras vacías) o agrega sinónimos.

Lucene primero divide los campos de cada documento en palabras separadas (es decir, términos o tokens) y crea una lista de todos los términos y documentos únicos para almacenar en la estructura de datos de índice invertido. Por ejemplo:

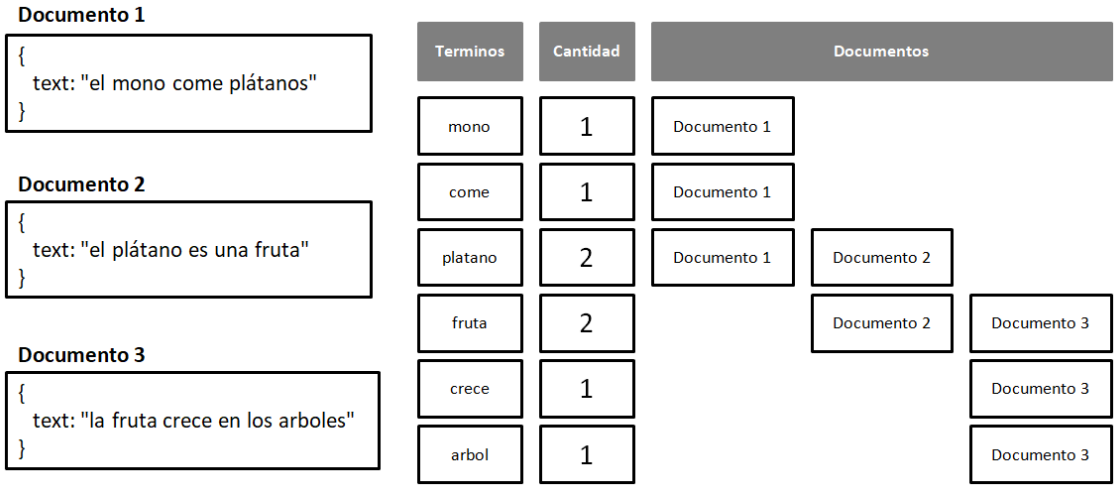



**Figura 2.2 – Ejemplo de Indice Invertido**

Se crea un índice con los diferentes términos que aparecen, indicando el documento en el que aparece cada término. Esta estructura permite una búsqueda eficiente y rápida, especialmente para consultas basadas en términos. La razón de esto es clara: enumerar los documentos por palabra es menos costoso que enumerar las palabras por documento en términos de recursos

del sistema. Además, un índice invertido incluye la posición de cada término dentro del documento y tiene un valor numérico que indica la incidencia de un término.

Este es el tipo de indexación que usan los buscadores, ya que permite una rápida búsqueda en documentos de texto. El índice, además de indicar el documento donde aparece un término de búsqueda dado, podría contener otra información, por ejemplo, la frecuencia de aparición del término, o el lugar del documento donde aparece.

### 2.1.4. Mapping en Elasticsearch

Los mappings son como esquemas de bases de datos que describen los campos o propiedades que pueden tener los documentos de ese tipo.

### 2.1.5. Búsqueda en Elasticsearch

Elasticsearch distingue entre dos tipos de búsquedas:

- Las búsquedas que devuelven una puntuación o **score**, que se llaman **queries**: Están pensadas para búsquedas sobre campos de tipo frase, ya que este tipo de consultas lo que hacen es medir cuánto se parece una frase buscada a la almacenada.
- Las búsquedas sin puntuación o filters: Estos filtros se utilizan para búsquedas binarias, en las que hay dos posibilidades, o se encuentra el valor o no se encuentra. Tienen la ventaja que son más rápidas y que se cachean en memoria. Este tipo de búsquedas se utilizan para campos que no sean de texto.

La similitud es una función de ranking utilizada en Recuperación de Información (RI) para la asignación de relevancia a los documento ante una búsqueda. Es una función que nos permite ordenar por relevancia los documentos que contienen las palabras que forman parte de la búsqueda.

El módulo de similitud de Elasticsearch será el encargado de calcular esta puntuación (score) para el tipo de búsquedas llamadas "queries".

La similitud define cómo se puntúan los documentos coincidentes. La similitud se realiza por campo, lo que significa que a través del mapeo se puede definir una similitud diferente para cada campo.

La similitud Okapi BM25 (que es la predeterminada) se basa en *TF/IDF* que tiene una normalización de *TF* incorporada y se supone que funciona mejor para campos cortos (como nombres). Dada una consulta *Q*, que contiene las palabras clave $q_1$, ..., $q_n$, el valor de relevancia asignado mediante la función BM25 para el documentos *D* será:

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})} \quad (1)$$

donde $f(q_i, D)$ es la frecuencia de aparición en el documento D de los términos que aparecen en la consulta Q, |D| es la longitud del documento D (en número de palabras), y *avgdl* es la longitud media de los documentos en la colección sobre la cual estamos realizando la búsqueda. $IDF(q_i)$ es el peso IDF (Inverse Document Frequency) de las palabras clave que aparecen en la consulta Q. Normalmente el *IDF* se calcula mediante la siguiente función:

$$IDF(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} \quad (2)$$

donde *N* es el número total de documentos en la colección, y $n(q_i)$ es el número de documentos que contienen la palabra clave $q_i$. $k_1$ y *b* son parámetros que permiten ajustar la función a las características concretas de la colección con la que estamos trabajando. Esta similitud tiene las siguientes opciones:

- `k1`: Controla la normalización de frecuencia de términos no lineales (saturación). El valor predeterminado es 1.2.
- `b`: controla hasta qué punto la longitud del documento normaliza los valores de TF. El valor predeterminado es 0,75.
- `discount_overlaps`: Determina si los tokens superpuestos (tokens con incremento de posición 0) se ignoran al calcular la norma. De forma predeterminada, esto es cierto, lo que significa que los tokens superpuestos no cuentan cuando se calculan las normas.

Elasticsearch dispone de un conjunto de similitudes pero se puede configurar una similitud personalizada: Similitud DFR (Divergence from Randomness), Similitud DFI (Divergence from Independence), Similitud IB (Information Base), Similitud de LM Dirichlet, Similitud de LM Jelinek Mercer y Similitud Scripted.

La similitud BM25, con valores de $k_1$ = 1.2 y *b* = 0.75, fue la que se utilizó en este trabajo.

## 2.2. Búsqueda de respuestas (QA)

La búsqueda de respuestas, llamada en inglés Question Answering (QA) es una rama de la inteligencia artificial dentro de los campos de procesamiento del lenguaje natural (NLP) y recuperación de información (RI). Es un tipo de recuperación de información. El sistema debería responder a preguntas planteadas en un lenguaje natural por los humanos.

Los sistemas de QA pueden construir respuestas a través de la consulta de una base de conocimientos (una base de datos estructurada de conocimientos) o una colección no estructurada de documentos en un lenguaje natural.

Los sistemas de QA son de dominio cerrado (respondiendo preguntas de un dominio específico) o de dominio abierto (basándose en ontologías generales y conocimiento generalizado).

Los sistemas de QA de dominio abierto toman preguntas en lenguaje natural y las transforman en una consulta estructurada. La extracción de palabras clave se utiliza para determinar el tipo de pregunta (quién, dónde, cuántos). Parte del etiquetado de voz y las técnicas de análisis sintáctico se pueden utilizar para determinar el tipo de respuesta (persona, ubicación, número). Luego, se utiliza un sistema de recuperación de información para encontrar los datos que contienen las palabras clave. Luego, se usa el análisis para traducir la respuesta en texto significativo.

Los primeros sistemas de QA se desarrollaron en torno a 1960 y básicamente eran interfaces de lenguaje natural para sistemas expertos centradas en dominios específicos. Los sistemas de

QA actuales utilizan documentos de texto como base de conocimiento y combinan diversas técnicas de procesamiento del lenguaje natural.

### 2.2.1. Sistema de QA

El esquema general de estos sistemas suele dividirse en tres grandes bloques, unidos de manera secuencial:

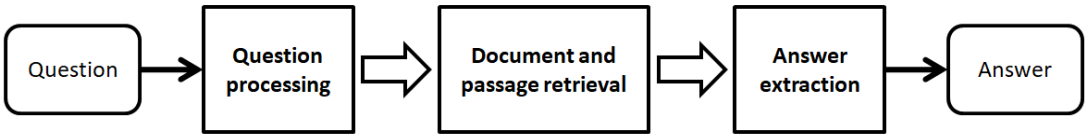



**Figura 2.3 – Esquema de un sistema de QA**

Las características principales de estos bloques son:

1. **Question processing**: Este módulo es el encargado, mediante la utilización de herramientas de NLP, de transformar la pregunta en una consulta que entienda el siguiente bloque. Además puede sacar información de la pregunta, como puede ser el "tipo de pregunta", que sea útil a la hora de elegir una respuesta.
2. **Document and passage retrieval**: Habitualmente, un sistema RI de recuperación de texto (documentos, párrafos…) que procesa la consulta que le pasa el bloque anterior y devuelve una colección de documentos o fragmentos de ellos (páginas, párrafos, frases...) relevantes, relacionada con la cadena introducida.
3. **Answer extraction**: En este bloque se trocea la información obtenida en el apartado anterior, se filtra lo que puede ser una respuesta y se evalúa, por diferentes métodos, la probabilidad de que sea la respuesta correcta, devolviendo la mejor valorada.

## 2.3. Transformers

Los modelos basados en Redes Neuronales Recurrentes (RNN) que hacían uso de embeddings (representaciones matriciales estáticas del texto, en las que cada palabra del vocabulario está codificada en un vector) se convirtieron en el estándar, al menos hasta finales del año 2017. Las RNN se utilizaban principalmente para las tareas de comprensión del lenguaje, modelado del lenguaje, la traducción automática y la respuesta a preguntas.

En diciembre de 2017, Vaswani et al. (Google Research y Google Brain) publicaron un artículo, "*Attention is All You Need*", donde presentaban la arquitectura del Transformer y la arquitectura secuencia a secuencia (*Seq2Seq*). Esta arquitectura de red neuronal novedosa se basaba en un mecanismo de auto atención que era particularmente adecuado para la comprensión del lenguaje. Este modelo tenía como principal innovación la sustitución de las capas recurrentes, como las LSTMs que se venían usando hasta ese momento en NLP (Natural Laguage Processing), por las denominadas capas de atención.

La mayoría de los modelos neuronales de traducción de secuencias tienen una estructura de codificador-decodificador. El codificador mapea una secuencia de entrada de representaciones de símbolos ($x_1$, ..., $x_n$) a una secuencia de representaciones continuas $z$ = ($z_1$, ..., $z_n$). Dado $z$, el decodificador genera una secuencia de salida ($y_1$, ..., $y_m$) de símbolos. En cada paso, el modelo

es auto-regresivo, utilizando los símbolos generados previamente como entrada adicional para generar el siguiente.

Tranformers sigue esta arquitectura general mediante el uso de capas apiladas de auto-atención y redes FFN, tanto para el codificador como para el decodificador.

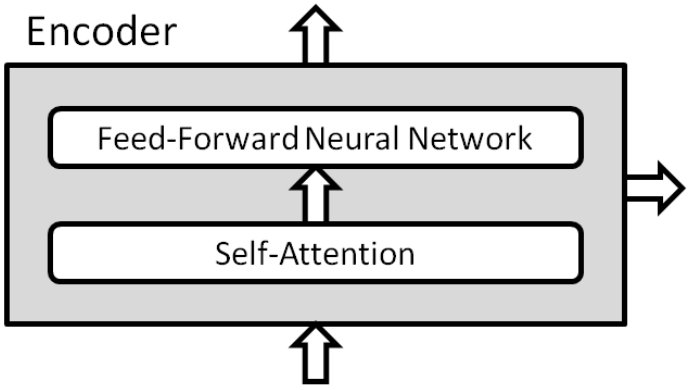



**Figura 2.4 - Arquitectura de capas apiladas de Self-Attention y FF Network**

El modelo Transformer original es una pila de 6 capas. La salida de la capa *l* es la entrada de la capa *l* + 1 hasta que se alcanza la predicción final. Hay una pila de codificadores de 6 capas a la izquierda y una pila de decodificadores de 6 capas a la derecha. Vamos a ver los componentes que constituyen este modelo.

### 2.3.1 Pila de Codificadores y Decodificadores

Codificador:

El codificador se compone de una pila de N = 6 capas idénticas. Cada capa tiene dos subcapas. La primera es un mecanismo de auto-atención de múltiples cabezales, y la segunda es una red FF simple, completamente conectada. Empleamos una conexión residual alrededor de cada una de las dos subcapas, seguida de la normalización de la capa. Es decir, la salida de cada subcapa es LayerNorm (x + Sublayer (x)), donde Sublayer (x) es la función implementada por la propia subcapa. Para facilitar estas conexiones residuales, todas las subcapas del modelo, así como las capas de embedding, producen salidas de dimensión $d_{model} = 512$.

Decodificador:

El decodificador también está compuesto por una pila de N = 6 capas idénticas. Además de las dos subcapas en cada capa del codificador, el decodificador inserta una tercera subcapa, que realiza una atención de múltiples cabezales (multi-head attention) sobre la salida de la pila del codificador. De manera similar al codificador, empleamos conexiones residuales alrededor de cada una de las subcapas, seguidas de la normalización de la capa. También modificamos la subcapa de auto-atención en la pila de decodificadores para evitar que las posiciones atiendan a las siguientes. Este enmascaramiento, combinado con el hecho de que los embeddings de salida estén compensados en una posición, asegura que las predicciones para la posición i puedan depender solo de las salidas conocidas en posiciones menores que i.

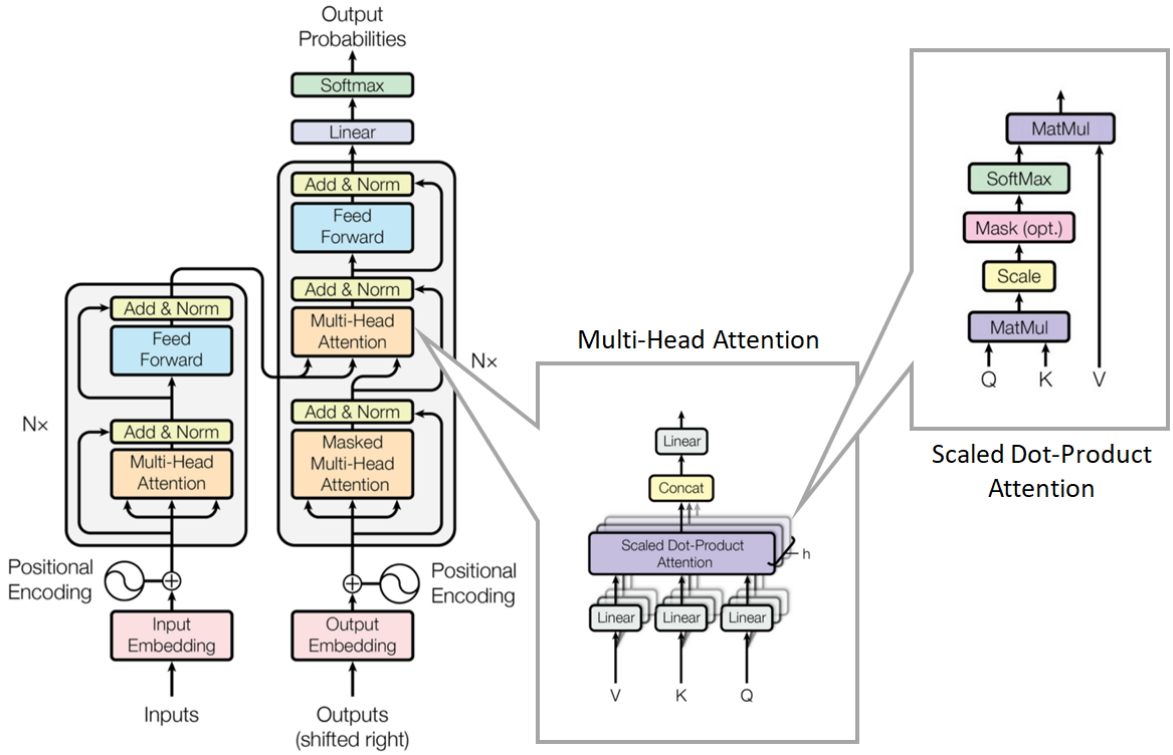



**Figura 2.5 - Arquitectura del modelo Transformers (Vaswani et al, 2017)**

La atención ha reemplazado a la recurrencia, que requiere un número creciente de operaciones a medida que aumenta la distancia entre dos palabras. El mecanismo de atención es una operación "palabra a palabra".

### 2.3.2 Atención

El mecanismo de atención encontrará cómo se relaciona cada palabra con todas las demás palabras en una secuencia, incluida la palabra que se analiza. Por ejemplo, en la sentencia: "El pájaro verde sobre la rama"

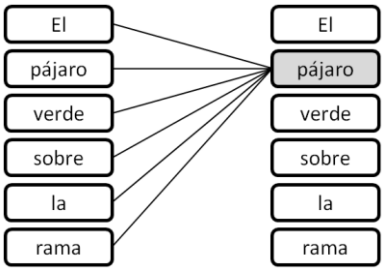
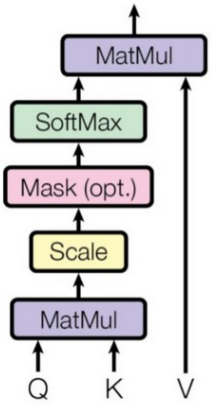



**Figura 2.6 - Atención para todas las palabras**

La atención calculará los productos escalares entre vectores de palabras y determinará las relaciones más fuertes de una palabra con todas las demás palabras, incluida ella misma (pájaro-pájaro).

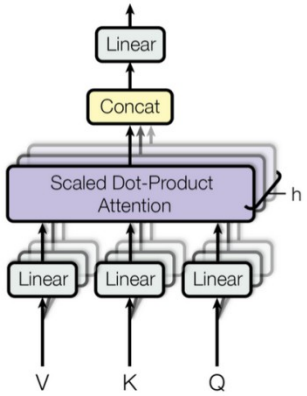
Para cada subcapa de atención, el modelo original de Transformer ejecuta ocho mecanismos de atención en paralelo para acelerar los cálculos. A este proceso se le denomina "Multi-Head Attention".




$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

donde *q* y *k* son vectores de dimensión $d_k$

**Figura 2.7 - Scaled Dot-Product Attention (Vaswani et al, 2017)**




$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\text{head}_1, \ldots, \text{head}_h]\mathbf{W}_0$$

$$\text{where head}_i = \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V\right)$$

**Figura 2.8 - Multi-Head Attention (Vaswani et al, 2017)**

### 2.3.3 Codificación posicional

La arquitectura de Transformer incluye otras innovaciones, como los embeddings posicionales, que permiten al algoritmo conocer la posición relativa de cada palabra del texto.

Dado que el modelo Transformers no contiene recurrencia ni convolución, para que el modelo haga uso del orden de la secuencia, debemos introducir alguna información sobre la posición relativa o absoluta de los tokens en la secuencia. Con este fin, agregamos "codificaciones posicionales" a los embeddings de entrada en la parte inferior de las pilas de codificadores y decodificadores. Las codificaciones posicionales tienen la misma dimensión $d_{model}$ que los embeddings, por lo que se pueden sumar. Hay muchas opciones de codificaciones posicionales, aprendidas y fijas.

Para este modelo se usan funciones seno y coseno de diferentes frecuencias:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}}) \quad (3)$$

donde *pos* es la posición, e *i* es la dimensión. Es decir, cada dimensión de la codificación posicional corresponde a una sinusoide. Las longitudes de onda forman una progresión geométrica de 2π a 10000 2π. Se eligen esta funciones porque permiten al modelo aprender fácilmente a atender por posiciones relativas, ya que para cualquier desplazamiento fijo *k*, $PE_{pos+k}$ se puede representar como una función lineal de $PE_{pos}$.

### 2.3.4 Embeddings y Softmax

De manera similar a otros modelos de transformación de secuencias, se usan embeddings aprendidos para convertir los tokens de entrada y los tokens de salida en vectores de dimensión $d_{model}$. También se usa la transformación lineal aprendida y la función *softmax* para convertir la salida del decodificador en probabilidades predichas del próximo token. En este modelo, se comparte la misma matriz de pesos entre las dos capas de embeddings y la transformación lineal pre-softmax. En las capas de embeddings, se multiplican estos pesos por *sqrt*($d_{model}$).

### 2.3.5 Redes Forward-Feed (FFN)

Además de las subcapas de atención, cada una de las capas del codificador y descodificador contiene una FFN completamente conectada, que se aplica a cada posición de forma separada e idéntica. Consiste en dos transformaciones lineales con una activación ReLU (Rectified Linear Unit) en el medio.

$$FFN(x) = max(0, xW_1 + b_1)\, W_2 + b_2 \quad (4)$$

Si bien las transformaciones lineales son las mismas en diferentes posiciones, utilizan diferentes parámetros de una capa a otra. Otra forma de describir esto es como dos convoluciones con tamaño de kernel 1. La dimensionalidad de entrada y salida es $d_{model} = 512$, y la capa interna tiene dimensionalidad $d_{ff} = 2048$.

### 2.3.6 Tipos de modelos disponibles en Transformers

Cada uno de los modelos de la librería Transformers se incluye en una de las siguientes categorías:

- *Modelos autorregresivos (Autoregressive models)*: están entrenados previamente en la tarea clásica de modelado de lenguaje: adivinar el siguiente token después de leer todos los anteriores. Corresponden al decodificador del modelo de Transformer original, y se usa una máscara encima de la oración completa para que el mecanismo de atención solo puedan ver lo que había antes en el texto, y no lo que sigue. Aunque esos modelos se pueden ajustar y lograr excelentes resultados en muchas tareas, la aplicación más natural es la generación de texto. Un ejemplo típico de tales modelos es GPT (Generative Pre-Training).
- *Modelos de codificación automática (Autoencoding models)*: se entrenan previamente corrompiendo los tokens de entrada de alguna manera y tratando de reconstruir la oración original. Corresponden al codificador del modelo de Transformer original en el sentido de que tienen acceso a las entradas completas sin ninguna máscara. Estos modelos suelen construir una representación bidireccional de toda la oración. Se pueden ajustar y lograr excelentes resultados en muchas tareas, como la generación de texto, pero su aplicación más natural es la clasificación de oraciones. Un ejemplo típico de tales modelos es BERT (Bidirectional Encoder Representations from Transformers). Otros ejemplo son RoBERTa (Robustly Optimized BERT Pretraining Approach), DistilBERT (distilled version of BERT), Longformer (Long-Document Transformer) y otros.
- *Modelos secuencia a secuencia (sequence-to-sequence models)*: utilizan tanto el codificador como el descodificador del Transformer original, ya sea para tareas de traducción o para transformar otras tareas en problemas secuencia a secuencia. Pueden ajustarse a muchas tareas, pero sus aplicaciones más naturales son la traducción, el resumen y la respuesta a preguntas (question-answering). Como ejemplos de este modelo tenemos el modelo de transformador original (solo para traducción), T5, MT5, BART, Pegasus y otros.
- *Modelos multimodales (Multimodal models)*: mezclan entradas de texto con otros tipos (por ejemplo, imágenes) y son más específicos para una tarea determinada. Como ejemplo esta MMBT (Supervised Multimodal Bitransformers).
- *Modelos basados en recuperación (Retrieval-based models)*: Algunos modelos utilizan la recuperación de documentos durante el pre-entrenamiento y la inferencia para

responder a preguntas (question-answering) de dominio abierto. Como ejemplos están DPR (Dense Passage Retrieval) y RAG (Retrieval-Augmented Generation).

Para este trabajo se han utilizado los modelos pre-entrenados de codificación automática (Autoencoding models). Vamos a describir algunos de los modelos utilizados,

### 2.3.7 BERT

En mayo de 2019, Devlin et al. (Google AI Language) presentaron un nuevo modelo de representación del lenguaje llamado BERT (Bidirectional Encoder Representations from Transformers). BERT es un modelo de embedding basado en el contexto.

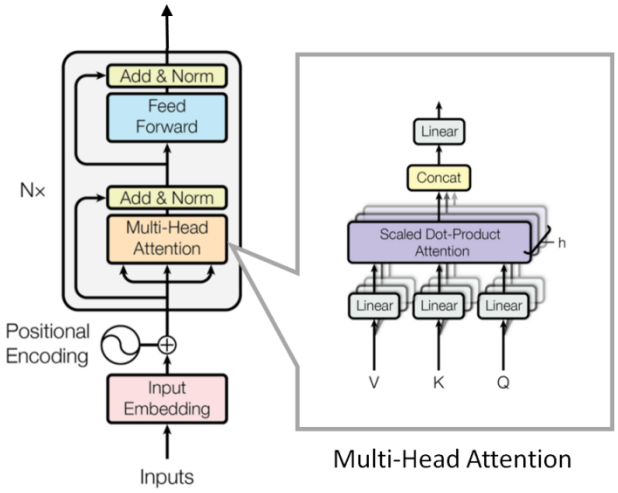



**Figura 2.9 - Arquitectura de BER (Vaswani et al, 2017)T**

Los modelos como word2vec o GloVe generan una representación de una sola palabra para cada palabra en el vocabulario (asignación fija, aunque esta palabra sea polisémica), mientras que BERT tiene en cuenta el contexto para cada aparición de una palabra determinada.

Como su nombre indica BERT se basa en Transformer, pero solo con el codificador (encoder). Le enviamos una sentencia a la entrada del codificador de Transformer y este nos devuelve como salida la representación de cada palabra en la sentencia. El codificador del Transformer es bidireccional ya que puede leer la sentencia en ambas direcciones. Es decir, tiene en cuenta las palabras a la derecha de una palabra dada, y las palabras a la izquierda. Por lo tanto, BERT es básicamente las representaciones del codificador bidireccional obtenidas de Transformers.

BERT se presento con dos configuración estándar:

- BERT-base (12-layer, 768-hidden, 12-heads, 110M parameters)
- BERT-large (24-layer, 1024-hidden, 16-heads, 340M parameters)

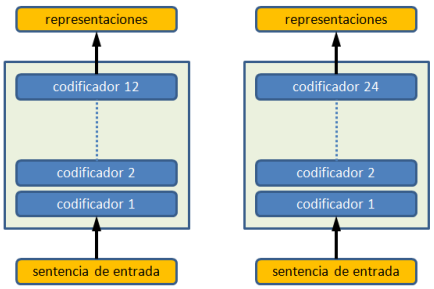



**Figura 2-10 - BERT-base y BERT-large**

### 2.3.8 RoBERTa

En julio de 2019, Yinhan Liu et al. publicaron un artículo "RoBERTa: A Robustly Optimized BERT Pretraining Approach" donde proponían el modelo RoBERTa que se basaba en el modelo BERT de Google.

Esta implementación es la misma que BertModel con un pequeño ajuste de embeddings, así como una configuración para modelos previamente entrenados por Roberta.

RoBERTa tiene la misma arquitectura que BERT, pero usa un BPE de nivel de byte como tokenizador (igual que GPT-2) y usa un esquema de pre-entrenamiento diferente.

RoBERTa no tiene `token_type_ids`, no es necesario que indique qué token pertenece a qué segmento. Simplemente separe sus segmentos con el token de separación `tokenizer.sep_token` (o `</s>`).

### 2.3.9 DistilBERT

En marzo de 2020, Victor SANH et al. publicaron un artículo " DistilBERT, a distilled version of BERT: smaller,faster, cheaper and lighter " donde proponían el modelo DistilBERTa que se basaba en el modelo BERT de Google.

DistilBERT es un modelo de Transformers pequeño, rápido, económico y ligero entrenado mediante destilación de la base BERT. Tiene un 40% menos de parámetros que bert-base-uncased, funciona un 60% más rápido y conserva más del 95% del rendimiento de BERT según lo medido en el punto de referencia de comprensión del lenguaje GLUE.

DistilBERT no tiene `token_type_ids`, no es necesario que indique qué token pertenece a qué segmento. Simplemente separa sus segmentos con el token de separación `tokenizer.sep_token` (o `[SEP]`).

DistilBERT no tiene opciones para seleccionar las posiciones de entrada (entrada `position_ids`).

## 2.4. Transformers aplicados a la extracción de respuestas

Una vez instalada la libreria Transformers, lo primero que debemos hacer es elegir un modelo

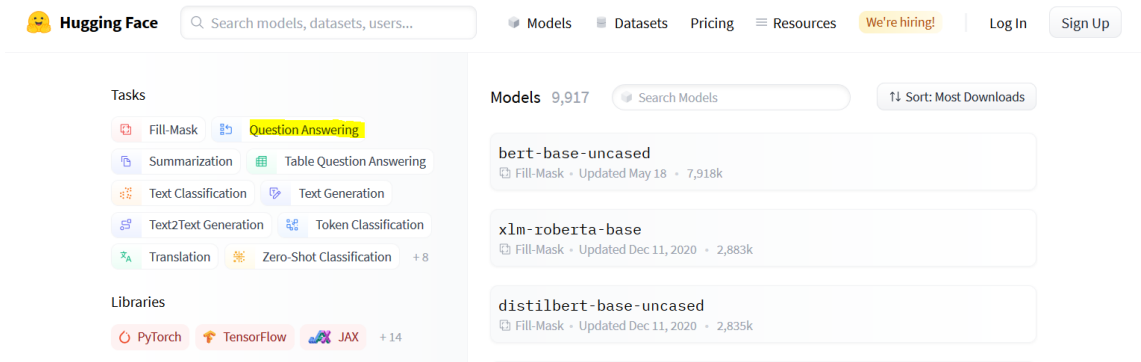



**Figura 2.11 - Pagina de modelos de Huggingface**

En la página de modelos de HuggingFace (`huggingface.co/models`) seleccionamos "Question-Answering" para filtrar los modelos entrenados específicamente para respuesta a preguntas. También podemos buscar modelos específicos.

Los modelos que se van a utilizar son modelos de codificacion automatica (Autoencoders models) pre-entrenados para responder a preguntas (question-answering):

- `ktrapeznikov/biobert_v1.1_pubmed_squad_v2`
- `deepset/roberta-base-squad2` (12-layer, 768-hidden, 12-heads, 125M parámetros El modelo RoBERTa usa la arquitectura BERT-base)
- `distilbert-base-uncased-distilled-squad` (6-layer, 768-hidden, 12-heads, 66M parámetros. El modelo DistilBERT es un modelo destilado del modelo BERT `bert-base-uncased`, con una capa lineal adicional.)
- `distilbert-base-cased-distilled-squad` (6-layer, 768-hidden, 12-heads, 65M parameters. El model DistilBERT model distilled from the BERT model `bert-base-cased` checkpoint, with an additional question answering layer)
- `deepset/bert-large-uncased-whole-word-masking-squad2` (24-layer, 1024-hidden, 16-heads, 336M parámetros. Entrenado con texto en Inglés en minusculas usando Whole-Word-Masking)
- `ktrapeznikov/scibert_scivocab_uncased_squad_v2` (12-layer, 768-hidden, 12-heads, 110M parameters. Es un modelo BERT entrenado con textos cientificos)

Basicamente, el proceso de respuesta a preguntas consta de tres pasos:

- Inicialización de modelo y tokenizador
- Tokenización de consultas
- Pipeline y predicción

1. Inicialización del modelo y tokenizador: Este es el primer paso. Una vez selecionado el modelo correspondiente, se importará Transformers y se inicializará el modelo y tokenizador. Se va a usar como ejemplo el modelo: `distilbert-base-uncased-distilled-squad`.

```
from transformers import AutoModelForQuestionAnswering
from transformers import AutoTokenizer

the_model = "distilbert-base-uncased-distilled-squad"

model = AutoModelForQuestionAnswering.from_pretrained(the_model)
tokenizer = AutoTokenizer.from_pretrained(the_model)
```

El modelo `distilbert-base-uncased-distilled-squad` ha sido pre-entrenado para QA con el conjunto de datos SQuAD 1.1 (lo indica `squad` que aparece al final). El modelo es `distilbert-base`, es decr, la versión base de BERT (no la large). Es también `uncased` que significa que considera iguales las letras minúsculas y mayúsculas.

2. Tokenización de datos de entrada: se ha inicializado el tokenizador y ahora hay que suministrarle texto para que lo convierta en *token IDs* que entienda DistilBERT. El tokenizador se encarga de convertir texto en datos compatibles con DistilBERT, llamados *token IDs*.

Lo que primero realiza el tokenizador es tomar una cadena y dividirla en tokens. Por ejemplo, para la siguiente cadena de texto:

```
texto: "What is the capital of Scotland?"
```

Obtendríamos los siguientes tokens:

```
token IDs: ["[CLS]", "What", "is", "the", "capital", "of", "Scotland", "?",
"[SEP]", "[PAD]", "[PAD]"]
```

Algunos de estos tokens, como `[CLS]` y `[PAD]`, son tokens especiales y son utilizados por DistilBERT para significar diferentes cosas, por ejemplo:

| Token | Significado | Token ID |
|---|---|---|
| [PAD] | El token de relleno nos permite mantener secuencias de la misma longitud (512 tokens para Bert) incluso cuando se introducen oraciones de diferentes tamaños | 0 |
| [UNK] | Se usa cuando una palabra es desconocida para BERT | 100 |
| [CLS] | Aparece al comienzo de cada secuencia | 101 |
| [SEP] | Indica un separador. Se usa para indicar el punto entre el contexto y la pregunta y aparece al final de las secuencias. | 102 |
| [MASK] | Se usa al enmascarar tokens, por ejemplo, en entrenamiento con modelado de lenguaje enmascarado (MLM, Masked Language Modelling) | 103 |

**Tabla 2.1 - Algunos de los Tokens que utiliza DistilBERT y su Token ID asociado**

Después de esta tokenización inicial, se convierte esta lista de strings (tokens) en una lista de números enteros (*Token ID*). Esto se hace usando un diccionario interno que contiene todos los tokens comprendidos por DistilBERT. Cada uno de estos tokens se asigna a un valor entero único. Son estos *Token IDs* los que se introducen en DistilBERT.

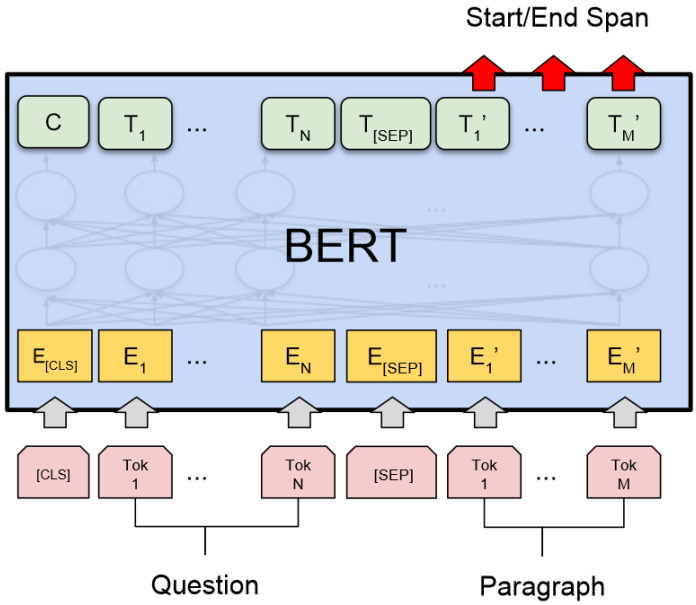



**Figura 2.12 - Arquitectura BERT para Question-Answering (Devlin et al, 2019)**

3. Pipeline y predicción: ahora ya se pueden hacer preguntas.

```
from transformers import pipeline
qa_pip = pipeline("question-answering", model=model, tokenizer=tokenizer)
```

Entonces se puede comenzar a hacer preguntas con solo especificar el *contexto* y la *pregunta*. Por ejemplo, (en contexto siguiente se ha extraído de Wikipedia, de la película Terminator):

```
context = """The Terminator is a 1984 science fiction film directed by James
Cameron. It stars Arnold Schwarzenegger as the Terminator, a cyborg assassin
sent back in time from 2029 to 1984 to kill Sarah Connor (Linda Hamilton),
whose son will one day save mankind from extinction by a hostile artificial
intelligence in a post-apocalyptic future. Michael Biehn plays Kyle Reese, a
soldier sent back in time to protect Sarah. The screenplay is credited to
Cameron and producer Gale Anne Hurd, while co-writer William Wisher Jr.
received a credit for additional dialogue."""
```

```
question = "Who is the director?"

resp = qa_pip(question=question, context=context)
print("\n\n", resp)
```

La respuesta será:

```
{'score': 0.9566303491592407, 'start': 58, 'end': 71, 'answer': 'James
Cameron'}
```

A la función `pipeline()` le estamos pasando la pregunta (`question`) y el contexto (`context`). Al introducirlos en un modelo de preguntas y respuestas, se espera un formato específico, que tendrá la forma siguiente:

```
[CLS] <context> [SEP] <question> [SEP] [PAD] [PAD] ... [PAD]
```

Entonces, en nuestro ejemplo, la entrada combinada se verá así:

```
"[CLS] The Terminator is a 1984 science fiction film directed by James
Cameron. It stars Arnold Schwarzenegger as the Terminator, a cyborg assassin
sent back in time from 2029 to 1984 to kill Sarah Connor (Linda Hamilton),
whose son will one day save mankind from extinction by a hostile artificial
intelligence in a post-apocalyptic future. Michael Biehn plays Kyle Reese, a
soldier sent back in time to protect Sarah. The screenplay is credited to
Cameron and producer Gale Anne Hurd, while co-writer William Wisher Jr.
received a credit for additional dialogue. [SEP] Who is the director? [SEP]
[PAD] [PAD] [PAD]"
```

Que luego se convertirá en un formato de *Token IDs*:

```
'inputs_ids': [[
        101,  1996,  2744, 23207,  2003,  1037,  3118,  2671,  4349,  2143,
       2856,  2011,  2508,  7232,  1012,  2009,  3340,  7779, 29058,  8625,
      13327,  2004,  1996,  2744, 23207,  1010,  1037, 22330, 11755, 12025,
       2741,  2067,  1999,  2051,  2013, 16798,  2683,  2000,  3118,  2000,
       3102,  4532,  6720,  1006,  8507,  5226,  1007,  1010,  3005,  2365,
       2097,  2028,  2154,  3828, 14938,  2013, 14446,  2011,  1037, 10420,
       7976,  4454,  1999,  1037,  2695,  1011, 27660,  2925,  1012,  2745,
      12170, 11106,  2078,  3248,  7648, 15883,  1010,  1037,  5268,  2741,
       2067,  1999,  2051,  2000,  4047,  4532,  1012,  1996,  9000,  2003,
       5827,  2000,  7232,  1998,  3135, 14554,  4776, 15876,  4103,  1010,
       2096,  2522,  1011,  3213,  2520,  4299,  2121,  3781,  1012,  2363,
       1037,  4923,  2005,  3176,  7982,  1012,   102,  2040,  2003,  1996,
       2472,  1029,   102]])
```

Y es esto lo que le pasará a DistilBERT. DistilBERT realizará los cálculos, nos devolverá los índices de los caracteres inicial y final que conforman la respuesta, es decir, el índice del carácter inicial y el índice del carácter final. Que en nuestro caso será:

```
{'score': 0.9566303491592407, 'start': 58, 'end': 71, 'answer': 'James
Cameron'}
```

donde

- **score** (float): es la probabilidad asociada a la respuesta.
- **start** (int): es el índice del carácter inicial de la respuesta.
- **end** (int): es el índice del carácter final de la respuesta.
- **answer** (str): es la respuesta a la pregunta.

En el siguiente texto aparecen marcados en negro los caracteres entre el 58 y 71 de la cadena de texto `context`:

```
context = """The Terminator is a 1984 science fiction film directed by James
Cameron. It stars Arnold Schwarzenegger as the Terminator, a cyborg assassin
sent back in time from 2029 to 1984 to kill Sarah Connor (Linda Hamilton),
whose son will one day save mankind from extinction by a hostile artificial
intelligence in a post-apocalyptic future. Michael Biehn plays Kyle Reese, a
soldier sent back in time to protect Sarah. The screenplay is credited to
Cameron and producer Gale Anne Hurd, while co-writer William Wisher Jr.
received a credit for additional dialogue."""
```

## 2.5. Estado del arte

Se han recopilado una serie de artículos actuales sobre este tema:

### 2.5.1. The University of Texas at Dallas HLTRI's Participation in EPIC-QA: Searching for Entailed Questions Revealing Novel Answer Nuggets

Weinzierl y Harabagiu, del HLTRI (Human Language Technology Research Institute) de la Universidad de Texas en Dallas, participaron en EPIC-QA con un sistema de recuperación de información neuronal multifase basado en la combinación de BM25, BERT y T5, así como la idea de considerar las relaciones de vinculación entre la pregunta original y las preguntas generadas automáticamente (a partir de las sentencias candidatas como respuesta). Además, debido a que también se consideraron las relaciones de vinculación entre todas las preguntas generadas, se pudo volver a clasificar las sentencias en función del número de nuevos *nuggets* que contenían, como lo indica el procesamiento de un gráfico de vinculación de preguntas.

El sistema se llama SEaRching for Entailed QUestions revealing NOVel nuggets of Answers (SER4EQUNOVA). La arquitectura del sistema es la siguiente

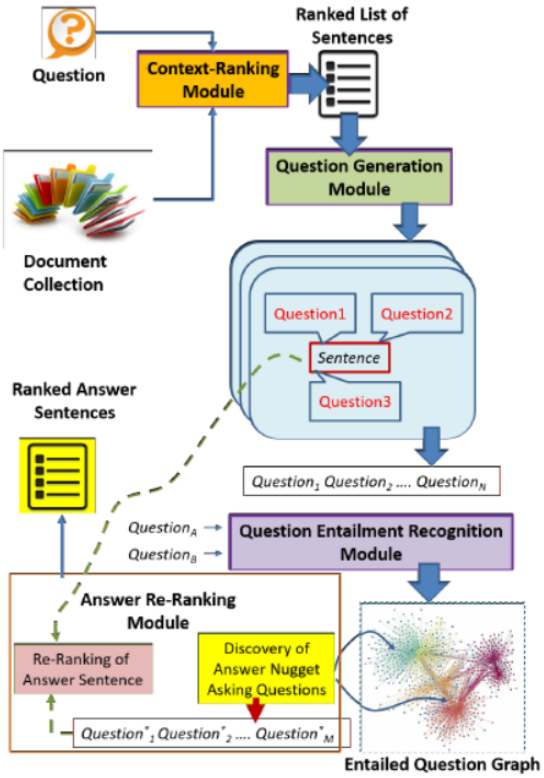



**Figura 2.13 - Arquitectura del sistema SEaRching for Entailed QUestions revealing NOVel nuggets of Answers (Weinzierl y Harabagiu).**

Dada una pregunta y la colección de documentos, el ***Módulo de Clasificación de Contexto*** recupera sentencias clasificadas de la colección de documentos que son candidatas para encontrar las respuestas a la pregunta. El ***Módulo de Generación de Preguntas*** genera preguntas para cada una de estas frases. Estos pares de preguntas se pasan a través del ***Módulo de Reconocimiento de Implicación de Preguntas***, para descubrir qué pregunta está textualmente implícita en otra pregunta. Estos pares de preguntas consisten en preguntas que se generaron a partir de las sentencias candidatas o en la pregunta original que se empareja con cualquiera de las preguntas generadas. Las relaciones de vinculación descubiertas permiten la generación del ***Gráfico de Preguntas de Vinculación*** (EQG), donde los nodos son preguntas y los vértices son las relaciones de vinculación descubiertas.

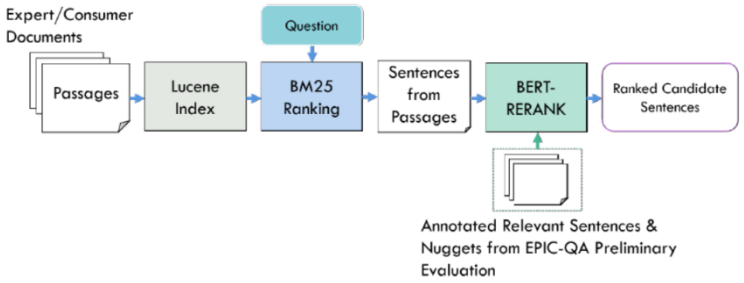



**Figura 2.14 - Módulo de clasificación de contexto de SER4EQUNOVA (Weinzierl y Harabagiu)**

El **Módulo de Clasificación de Contexto** tiene la tarea de buscar respuestas de candidatos relevantes para una pregunta determinada de la colección de documentos. El corpus se indexa con Lucene y se buscan pasajes relevantes (llamados "contextos") con BM25. A continuación, se consideran todas las sentencias dentro de cada contexto y se produce una puntuación de clasificación frente a cada pregunta utilizando el sistema BERT-RERANK. Se seleccionan las 1000 sentencias mas relevantes para cada pregunta de esta clasificación y se pasan al módulo de generación de preguntas.

El **Módulo de Generación de Preguntas** produce preguntas generadas sintéticamente para cada respuesta candidata y que es suministrada por el Módulo de Clasificación de Contexto. Las preguntas las genera el modelo docTTTT-Tquery entrenado con MS-MARCO.

El Gráfico de preguntas vinculadas (EQG) requiere un **Módulo de Reconocimiento de Preguntas Vinculadas** (RQE) para identificar las relaciones de vinculación de preguntas entre pares de preguntas. Se experimenta con dos sistemas RQE: BERTRQE y QBERT-RQE. BERT-RQE. Ambos modelos se ajustaron con el dataset de duplicación de preguntas de Quora1.

El **Módulo de Clasificación de Respuestas** tiene la tarea de volver a clasificar las respuestas proporcionadas por el Módulo de Clasificación de Contexto en función de los nuevos *nuggets* dentro de cada respuesta. Las preguntas generadas por el Módulo de Generación de Preguntas se utilizan para construir el Gráfico de preguntas vinculadas (EQG). El Módulo de Clasificación de Respuestas genera una lista clasificada de sentencias donde las sentencias más relevantes son aquellas que contienen mayor número de nuevos *nuggets*.

Los resultados obtenidos por este sistema para el conjunto de datos primario (Primary EPIC-QA dataset) y la Tarea A (Expert) son los siguientes:

| EXPERT | NDNS-Relaxed | NDNS-Partial | NDNS-Exact |
|---|---|---|---|
| Best | **0.371** | **0.370** | **0.421** |
| Median | 0.339 | 0.338 | 0.380 |
| CRM (RUN_2) | 0.364 | 0.363 | 0.413 |
| SER4EQUNOVA (RUN_3) | **0.371** | **0.370** | **0.421** |

### 2.5.2. Improving the Performance of Pre-trained Systems in Context Constrained Environments

Yastil Rughbeer, Anban Pillay y Edgar Jembere de la Universidad de KwaZulu-Natal proponen un framework en cascada que consta de dos etapas. La primera etapa utiliza sistemas no neuronales para recuperar las k primeras oraciones para una consulta determinada. Luego, un sistema previamente entrenado clasifica estas oraciones en orden descendente de relevancia. Su trabajo se centró en mejorar el rendimiento del sistema no neuronal. Esto lo logran asumiendo que recuperar pasajes en lugar de oraciones aumenta la probabilidad de recuperar oraciones con nuevas pepitas. Esta suposición se validó en el conjunto de datos preliminar después de que el sistema previamente entrenado experimentó una ganancia de rendimiento del 18%. La arquitectura del sistema es la siguiente:

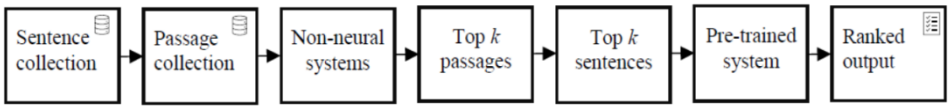



**Figura 2.15 - Arquitectura del sistema (Rughbeer et al.)**

En este trabajo se utilizaron varios sistemas no neuronales para la recuperación de información (BM25, ColBERT, DeepCT y docTTTTTquery). Los sistemas anteriores se entrenaron con el dataset MS MARCO.

Por último, como sistema neuronal previamente entrenado se usó la arquitectura BERT Large. La entrada a BERT se formó concatenando la consulta y la sentencia en una secuencia. BERT luego calculó la probabilidad de que la sentencia sea relevante para la pregunta, es decir, la puntuación de relevancia. Finalmente, las sentencias se ordenaron en orden descendente de relevancia.

Los resultados obtenidos por este sistema, para el conjunto de datos preliminar (Preliminary EPIC-QA dataset) y la Tarea A, son los siguientes:

| Data Pipeline | BERT Performance (NDNS-Exact) |
|---|---|
| Standard Pipeline | 0.1564 |
| Our Pipeline | 0.1855 |

### 2.5.3. IBM Submissions to7 EPIC-QA Open Retrieval Question Answering on COVID-19

Bhavani Iyer et al., investigadores del IBM Research AI y de la Universidad de Illinois realizaron tres presentaciones diferentes, cada una de las cuales sigue un proceso de tres etapas: recuperación de pasajes, búsqueda de respuestas y reclasificación

1. **IBM-1**: este sistema utiliza Anserini BM25 para la recuperación de pasajes, pero además reordena los pasajes recuperados utilizando un clasificador neuronal. Extrae intervalos de respuesta de estos pasajes reordenados utilizando el sistema de comprensión de lectura automática (MRC) de GAAMA. Un componente final utiliza los

intervalos de respuesta pronosticados en todos los pasajes para seleccionar y volver a clasificar la respuesta.

2. **IBM-2**: la única diferencia entre este sistema y el sistema IBM-1 es el módulo de recuperación de pasajes. Aquí se usa un conjunto de: (i) el módulo de recuperación de pasajes del sistema IBM-1, y (ii) el recuperador neuronal Dense Passage Retriever (DPR). El MRC y los módulos de reclasificación de respuestas siguen siendo los mismos.
3. **IBM-3**: Este sistema usa un tercer algoritmo de recuperación de pasajes, aquí se usa BM25 junto con un recuperador neuronal Dense Passage Retriever (DPR) adaptado al dominio COVID-19. El MRC y los módulos de reclasificación de respuestas siguen siendo los mismos que los otros dos sistemas.

En las tres ejecuciones, IR devuelve los 3000 pasajes principales, que luego asignamos a sus respectivos documentos de origen. Los 1000 documentos principales se ejecutan a través de MRC y la respuesta final se vuelve a clasificar.

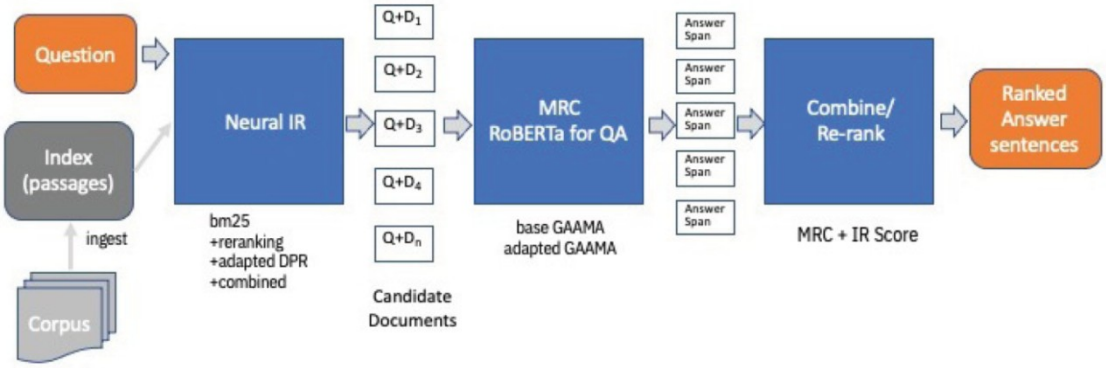



**Figura 2.16 - Open Retrieval Question Answering for EPIC-QA (Iyer et al.)**

Han utilizado el conjunto de datos preliminar (Preliminary EPIC-QA dataset), y los resultados obtenidos por los tres sistemas, para la Tarea A, son los siguientes:

| Run | Exact | Partial | Relaxed |
|---|---|---|---|
| Prelim Task A | | | |
| IBM-1 | 0.283 | **0.330** | 0.256 |
| IBM-2 | **0.293** | 0.259 | **0.259** |
| IBM-3 | 0.285 | 0.253 | 0.254 |

Capítulo 3

# Arquitectura de la solución

Nos encontramos ante un problema de big data y extracción de respuestas. Se parte de un conjunto de documentos muy grande: 129.069 documentos. Estos 129.069 documentos contienen 2.849.718 pasajes o 14.973.230 sentencias. El desafío EPIC-QA requiere que los participantes recuperen sentencias en lugar de documentos o pasajes.

```
QUESTION_ID    Q0    START_SENTENCE_ID:END_SENTENCE_ID    RANK    SCORE    RUN_NAME
```

**Figura 3.1 - Formato de los datos que se envían**

Pero el uso de sistemas pre-entrenados sobre la colección de sentencias requería una gran cantidad de cálculos. Por lo tanto, es necesario utilizar un sistema no neuronal que, para una consulta determinada, permita recuperar *m* sentencias. Posteriormente, el sistema pre-entrenado clasificará estas *m* sentencias en orden descendente de relevancia, y de estas sentencias se seleccionarán las *k* más relevantes.

Se utiliza un esquema basado en dos etapas: una de recuperación y otra de clasificación. El método de recuperación puede ser de dos formas diferentes: recuperación de sentencias y recuperación de pasajes.

Así, el esquema de dos etapas, con módulo de recuperación de sentencias, tendrá el siguiente aspecto:

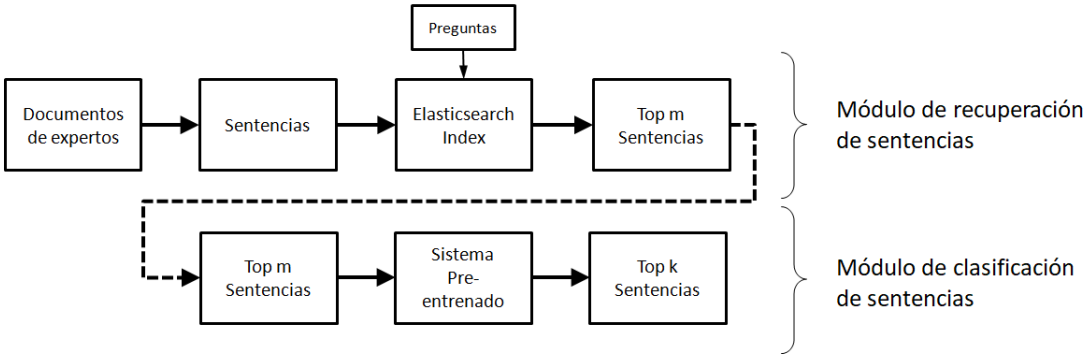



**Figura 3.2 – Arquitectura usando recuperación de sentencias**

Partiendo de la colección de documentos, se extraen las sentencias de todos los pasajes de los documentos y se crea un índice de sentencias en Elasticsearch. Posteriormente se recuperan m sentencias y, de estas m sentencias, con el sistema pre-entrenado, se clasifican las k sentencias más relevantes.

El esquema de dos etapas, con módulo de recuperación de pasajes, tendrá el siguiente aspecto:

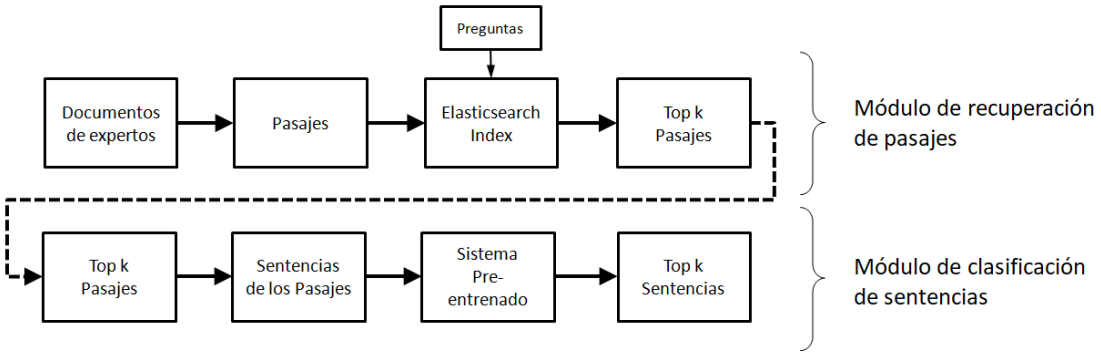



**Figura 3.3 - Arquitectura usando recuperación de pasajes**

Partiendo de la colección de documentos, se extraen los pasajes de todos los documentos y se crea un índice de pasajes en Elasticsearch, y se recuperan los *k* pasajes mas relevantes. Se recuerda que el numero de sentencias por pasaje está entre 1 y 15. De estos *k* pasajes se extraen las sentencias y, con el sistema pre-entrenado, clasificamos las *k* sentencias más relevantes.

Se va a usar una nueva arquitectura que añade a la anterior un módulo más de re-ranking de sentencias. Este nuevo módulo va a clasificar las sentencias asignándoles una nueva puntuación ($score_{NEW}$) teniendo en cuenta las puntuaciones obtenidas en los módulos de RI ($score_{RI}$) y QA ($score_{QA}$). Esta va a ser la arquitectura que se use en este trabajo.

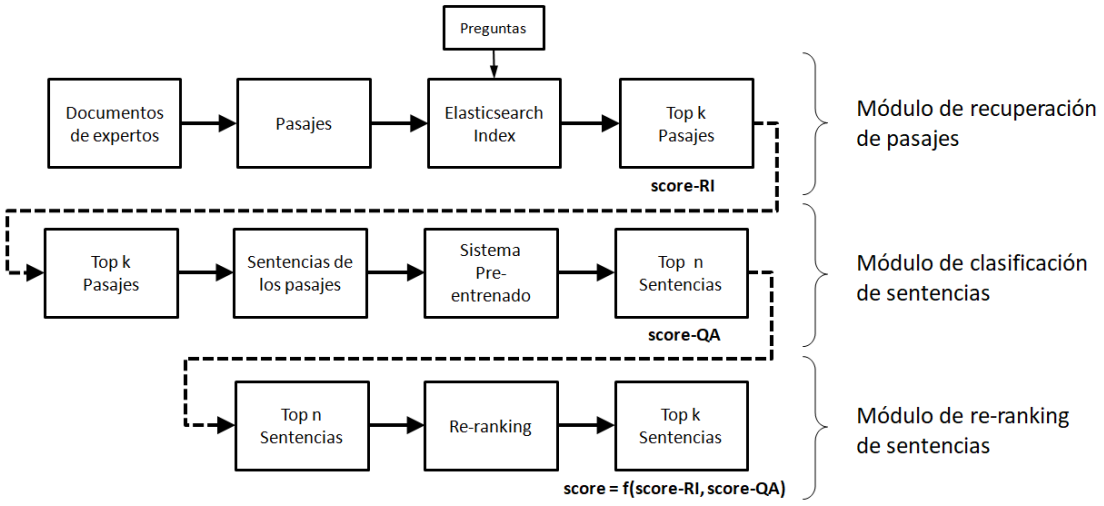



**Figura 3.4 - Arquitectura de tres etapas: recuperación de pasajes, clasificacion de sentencias y re-ranking de sentencias**

## 3.1. Indexación y recuperación de la información

Se ha partido de la colección de documentos de la ronda preliminar CORD-19 (972 MB), que está compuesta por artículos de investigación y que puede descargarse desde la URL:

`https://bionlp.nlm.nih.gov/epic_qa/data/epic_qa_cord_2020-06-19_v3.tar.gz`

Los documentos de esta colección se adhieren a una versión modificada del esquema JSON CORD-19 que se describe a continuación:

```
{
    "document_id": <str>,          # 40-character sha1 of the URL or document
    "metadata": {
        "title": <str>,            # Title of the document
        "url": <str>,              # URL of the page (for webpages)
        "authors": [...]           # Authors of the page (for CORD-19)
    },
    "contexts": [                  # List of context(s) in the document
        {
            "section": <str>,      # Name of the section (if any) containing the context
            "text": <str>,         # The full text (without markup) in the section
            "context id": <str>,   # Globally unique context identifier
            "sentences": [
                {
                    "start": 0,       # Inclusive character start offset of the sentence
                    "end": 27,        # Exclusive character end offset of the sentence
                    "sentence id": <str>,  # Globally unique identifier for the sentence
                },
                {                             # Second sentence in the context
                    "start": 28,
                    "end": 56,
                    "sentence id": <str>,
                },
                {...},                        # Third sentence in the context
                ...
            ]
        },
        {...},                                # Second context in the document
        ...
    ]
}
```

Los identificadores de contextos y sentencias se proporcionarán a los participantes como parte de la colección. En la colección CORD-19, los contextos corresponderán a los párrafos definidos por los autores de sus publicaciones. Los contextos que tienen más de 15 sentencias se segmentarán en bloques de aproximadamente 15 sentencias. Los contextos se segmentarán en sentencias, cada una asociada con una identificación única (`sentence_id`).

Los participantes deberán proporcionar las identificaciones iniciales y finales de las sentencias que constituyen cada una de sus respuestas. Para mantener la procedencia, cada respuesta también debe estar asociada con el ID del documento (`document_id`) y el ID de contexto (`context_id`) de los que se originó.

Por tanto se tienen 129.069 archivos con el anterior esquema JSON. A partir de esta colección de documentos crearemos dos índices en Elasticsearch: uno de pasajes y otro de sentencias.

### 3.1.1. Indexado por pasajes

Para crear el índice de pasajes usamos el script Python `indexing_passages.py`. Este script crea el índice `my-covid-map1` en Elasticsearch con el mapping siguiente:

```
mapping = {
   "mappings" : {
      "properties" : {
         "document id" : { "type": "text", "index": "false", "store": "false" },
         "context id" :  { "type": "text", "index": "false", "store": "false" },
         "text" :        { "type": "text", "index": "true"},
         "sentences": {
            "properties": {
               "sentence id": { "type": "text", "index": "false", "store": "false" },
               "start":       { "type": "integer", "store": "false" },
               "end":         { "type": "integer", "store": "false" }
            }
         }
      }
```

```
   }
}
```

Al terminar la ejecución del script tendremos un índice con 2.849.718 pasajes. En el índice solo se guarda la información de todos los pasajes (`contexts`) en cada documento, junto con el campo `"document_id"`. Solo se indexa el campo `"text"` que es sobre el que vamos a hacer las búsquedas.

### 3.1.2. Indexado por sentencias

Para crear el índice de sentencias usamos el script Python `indexing_sentences.py`. Este script crea el índice `my-covid-map2` en Elasticsearch con el mapping siguiente:

```
mapping = {
   "mappings" : {
      "properties" : {
         "sentence id" : { "type": "text", "index": "false", "store": "false" },
         "text" :        { "type": "text", "index": "true"},
      }
   }
}
```

Al terminar la ejecución del script tendremos un índice con 14.973.230 sentencias. En el índice solo se guarda la información de todos las sentencias (`sentences`) en cada documento. Solo se indexa el campo `"text"` que es sobre el que vamos a hacer las búsquedas.

### 3.1.3. Recuperación de información: pasajes

TREC-COVID ha preparado dos conjuntos de 45 preguntas: un conjunto para preguntas de nivel de experto (Tarea A) y otro para preguntas de nivel de consumidor (Tarea B). El conjunto de 45 preguntas de nivel de experto que vamos a utilizar se puede descargar de la siguiente URL:

`https://bionlp.nlm.nih.gov/epic_qa/data/expert_questions_prelim.json`

Las preguntas de esta tarea se proporcionarán utilizando el esquema JSON que se describe a continuación:

```
{
    "question_id": <str>,             # Globally unique identifier for the question
    "question": <str>,                # Text of the question
    "query": <str>,                   # Text of the query
    "background": <str>,              # Background narrative for the question
}
```

Para recuperar la información del índice `my-covid-map1` (que es el índice de pasajes) usamos el script Python `passages_retrieval.py`.

```
index_name = "my-covid-map1"
size_search = 1000
req_time = 10

query_body = {
   "query": {
      "simple_query_string": {
         "query": text_to_search,
         "fields": ["text"],
         "default_operator": "OR"
      }
   }
}

response = es.search(index=index_name, body=query_body, size=size_search,
request_timeout=req_time)
```

En la variable `text_to_search` copiaremos, cada uno de los tres campos `question`, `query` y `background`, para cada una de las 45 preguntas. Así obtendremos 1000 pasajes para cada una de las preguntas y cada uno de los campos anteriores.

El script Python `passages_retrieval.py`. nos crea dos tipos de ficheros:

- Ficheros JSON (`File_EQ001.json,…, File_EQ045.json`): guardan la respuesta dada por Elasticsearch a cada pregunta.
- Fichero TXT (`respMAP1*.txt`): es el fichero que se usa con el script de evaluación `epic_eval.py`. , y que permite evaluar el modelo de recuperación de información únicamente.

Por ejemplo, el fichero `File_EQ001.json` que guarda la consulta hecha a Elasticsearch usando el campo `"question"` de la pregunta EQ0001 tendrá el siguiente aspecto:

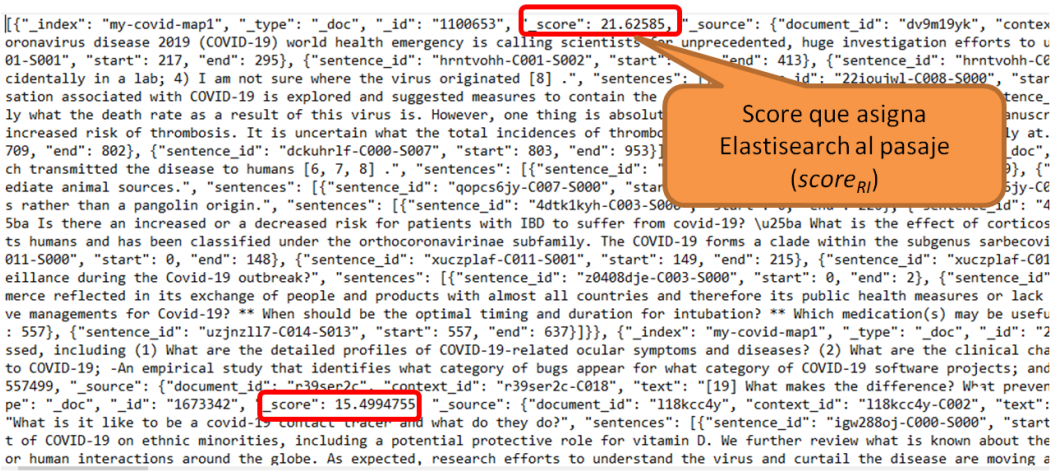



**Figura 3.5 - Ejemplo del fichero File_EQ001.json**

Elasticsearch puntúa los documentos que se recuperan mediante una consulta. Entonces en la recuperación de información (RI), con Elasticsearch, obtendremos una colección de pasajes o sentencias junto con su puntuación (***score**$_{RI}$*).

El fichero `respMAP1QTN.txt` que contiene los 1000 pasajes devueltos por Elasticsearch usando el campo `"question"` tendrá el siguiente aspecto:

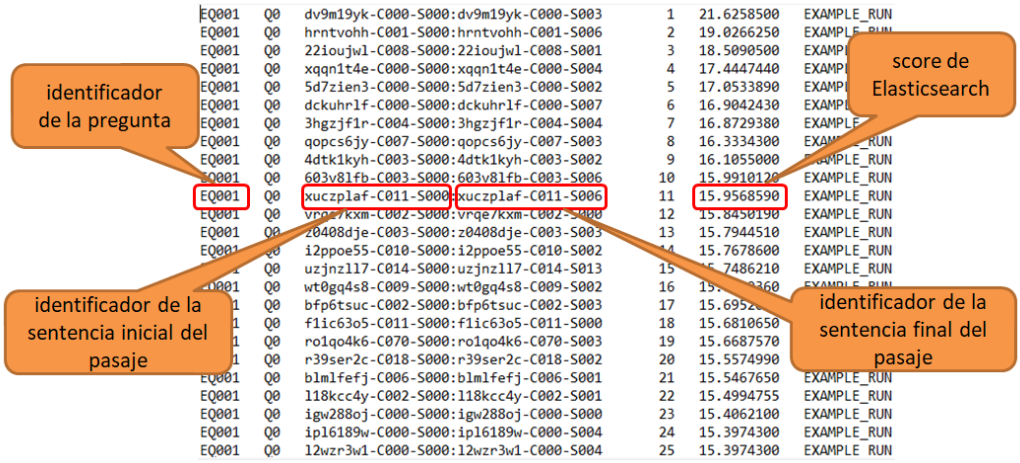



**Figura 3.6 - Ejemplo del fichero respMAP1QTN.txt**

El siguiente gráfico nos muestra un esquema del proceso de indexado y recuperación de información para los 1000 pasajes.

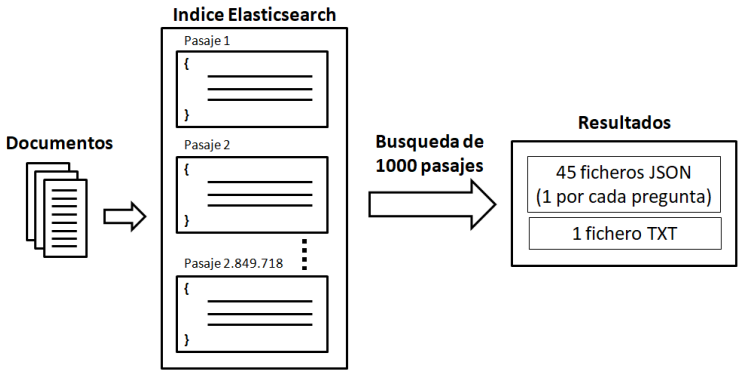



**Figura 3.7 - Esquema del proceso de recuperación de pasajes**

Todo este proceso se repetirá para los campos `question`, `query` y `background` de cada pregunta.

Después de este proceso tendremos tres ficheros para evaluar: `respMAP1QTN.txt` (para el campo `question`), `respMAP1QRY.txt` (para el campo `query`) y `respMAP1BCK.txt` (para el campo `background`).

### 3.1.4. Recuperación de información: sentencias

Para recuperar la información del índice `my-covid-map2` (indexado por sentencias) usamos el script Python `sentences_retrieval.py`.

```
index_name = "my-covid-map2"
size_search = 1000
req_time = 10

query_body = {
   "query": {
      "simple_query_string": {
         "query": text_to_search,
         "fields": ["text"],
         "default_operator": "OR"
      }
   }
}

response = es.search(index=index_name, body=query_body, size=size_search,
request_timeout=req_time)
```

En la variable `text_to_search` copiaremos, para cada una de las 45 preguntas. Así obtendremos 1000 sentencias para cada una de las preguntas y cada uno de los campos anteriores.

El script Python `sentences_retrieval.py`. nos crea dos tipos de ficheros:

- Ficheros JSON (`File_EQ001.json,…, File_EQ045.json`): guardan la respuesta dada por Elasticsearch a cada pregunta.
- Fichero TXT (`respMAP2*.txt`): es el fichero que se usa con el script de evaluación `epic_eval.py`.

Por ejemplo, el fichero `File_EQ002.json` que guarda la consulta hecha a Elasticsearch usando el campo `"question"` de la pregunta EQ0002 tendrá el siguiente aspecto:

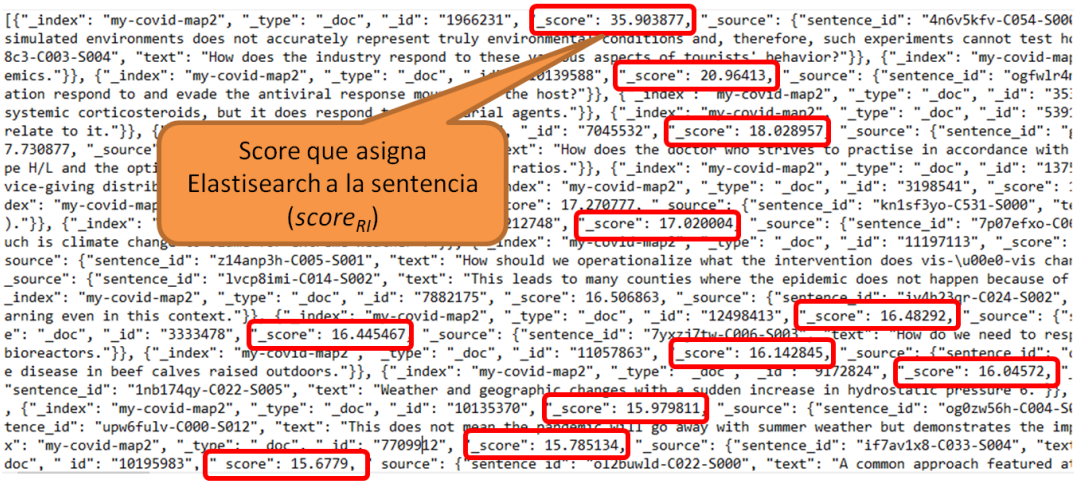



**Figura 3.8 - Ejemplo del fichero File_EQ002.json**

Elasticsearch puntúa las sentencias que se recuperan mediante una consulta. Entonces en la recuperación de información (RI), con Elasticsearch, obtendremos una colección de pasajes o sentencias junto con su puntuación (***score**$_{RI}$*).

Por ejemplo, el fichero `respMAP2QTN.txt` que contiene las 1000 sentencias devueltas por Elasticsearch usando el campo `"question"` tendrá el siguiente aspecto:

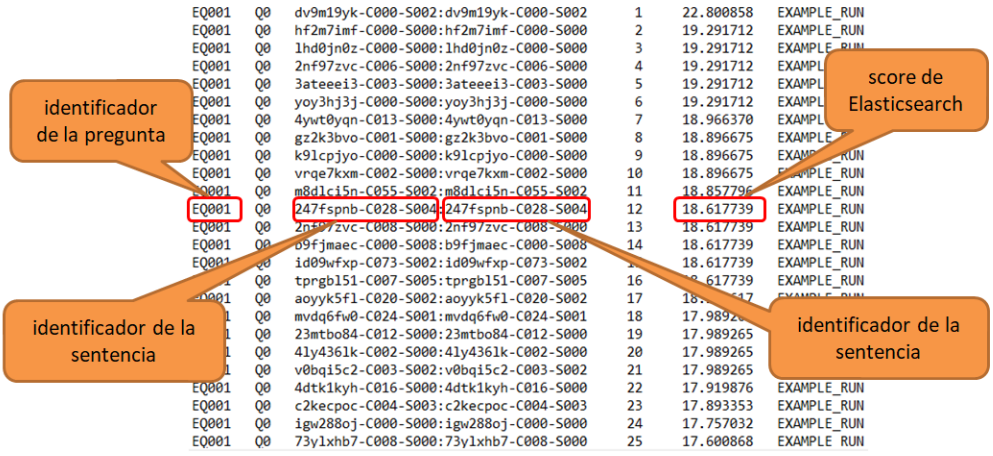



**Figura 3.9 - Ejemplo del fichero respMAP2QTN.txt**

Para el caso de recuperación de sentencias, en el fichero `respMAP2QTN.txt` que se evalúa, se observa que los campos identificador de sentencia inicial e identificador de sentencia final son iguales. En el caso de pasajes eran distintos ya que estos podían contener hasta 15 sentencias (En el caso de que el pasaje contuviera 1 sentencia también serian iguales).

Esto mismo ocurre con los otros dos ficheros, `respMAP2QRY.txt` y `respMAP2BCK.txt`, que son los que se evalúan con el script Python `epic_eval.py`.

El siguiente gráfico nos muestra un esquema del proceso de indexado y recuperación de información para las 1000 sentencias.

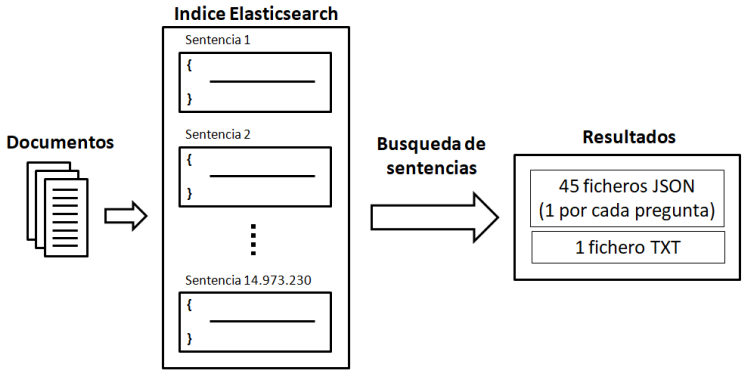



**Figura 3.10 - Esquema del proceso de recuperación de sentencias**

Todo este proceso se repetirá para los campos `question`, `query` y `background` de cada pregunta.

Después de este proceso tendremos tres ficheros para evaluar: `respMAP2QTN.txt` (para el campo `question`), `respMAP2QRY.txt` (para el campo `query`) y `respMAP2BCK.txt` (para el campo `background`).

## 3.2. Extracción de respuestas

En escenarios que contienen una gran cantidad de datos (14.9M de sentencias, 2.8M de pasajes) el uso únicamente de un sistema de respuesta a preguntas (QA) requiere una gran cantidad de cálculos y por tanto el tiempo de computación es muy alto. De ahí que la mejor solución sea la combinación de ambos sistemas: RI + QA.

El sistema de recuperación de información (RI) usado será el de recuperar k pasajes (del índice de pasajes), ya que los resultados obtenidos son mas relevantes que cuando se recuperan sentencias. De los k pasajes obtendremos n sentencias. Posteriormente usaremos un sistema de respuesta a preguntas para obtener un nuevo "score" de cada una de ellas y así obtener las 1000 sentencias mas relevantes.

El proceso de extracción de respuestas nos va a permitir asociarle a cada sentencia una nueva puntuación ($\boldsymbol{score_{QA}}$), y en función de esta obtener las sentencias más relevantes.

Para la Respuesta a Preguntas (QA) se usa la librería Transformers. Transformers proporciona arquitecturas de uso general (BERT, RoBERTa, DistilBert, GPT-2, XLM, XLNet ...) para la comprensión del lenguaje natural (NLU) y la generación del lenguaje natural (NLG) con más de 32 modelos previamente entrenados en más de 100 idiomas y una interoperabilidad profunda entre Jax, PyTorch y TensorFlow.

Los modelos disponibles permiten muchas configuraciones diferentes y una gran versatilidad de uso para tareas como respuesta a preguntas, clasificación de secuencias, reconocimiento de entidades nombradas y más.

Usaremos la función `pipelines()` de la librería Transformers con los parámetros:

- `task = "question-answering"`
- `model = <modelo a utilizar>`
- `tokenizer = <tokenizer a utilizar>`
- `framework = "pt"` ("`pt`" para PyTorch y "`tf`" para TensorFlow)

- `device = 0` (0 para GPU y -1 para CPU)

En el siguiente código Python se muestra el uso de la función `pipelines()`:

```
def qNa():
    the_model = "ktrapeznikov/biobert_v1.1_pubmed_squad_v2"
    model = AutoModelForQuestionAnswering.from_pretrained(the_model)
    tokenizer = AutoTokenizer.from_pretrained(the_model)
    qa_pip = pipeline("question-answering", model=model, tokenizer=tokenizer,
framework="pt", device=0)
    return qa_pip


def update_phrase_list(question_text, phrases):
    nlp_qa = qNa()
    for phrase in phrases:
        phrase_text = phrase['text']
        try:
            result = nlp_qa(question=question_text, context=phrase_text)
            scr = result['score']
        except:
            scr = 0
        phrase['score2'] = scr
```

Se evalúan diversos modelos (elegidos para la tarea "question-answering"):

- ktrapeznikov/biobert_v1.1_pubmed_squad_v2
- deepset/roberta-base-squad2
- distilbert-base-uncased-distilled-squad
- distilbert-base-cased-distilled-squad
- deepset/bert-large-uncased-whole-word-masking-squad2
- ktrapeznikov/scibert_scivocab_uncased_squad_v2

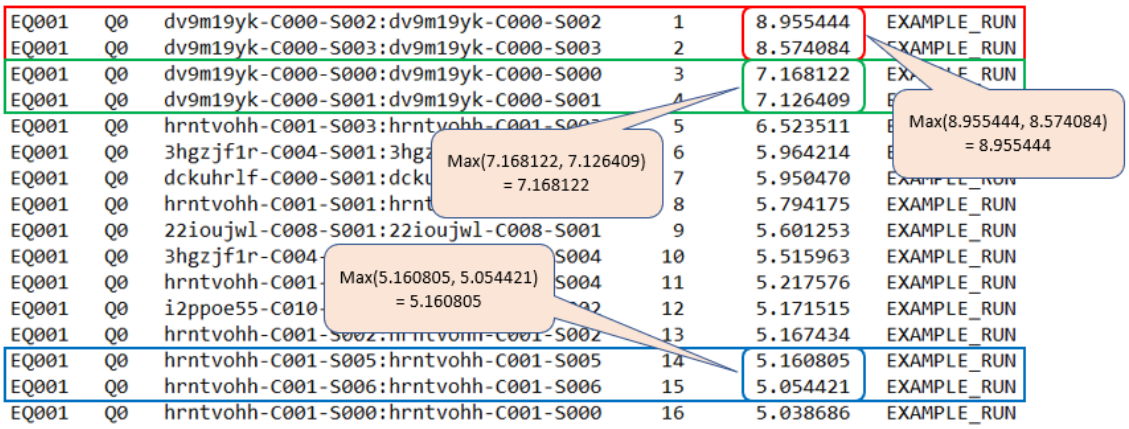
Así pues, para la tarea "*question-answering*", la función `pipelines()` de la librería Transformers nos devuelve un resultado que consiste en un diccionario con las siguientes claves:

- *score (float)*: la probabilidad asociada a la respuesta.
- *start (int)*: el índice de inicio de caracteres de la respuesta (en la versión tokenizada de la entrada).
- *end (int)*: el índice de final de carácter de la respuesta (en la versión tokenizada de la entrada).
- *answer (str)*: la respuesta a la pregunta.

Para cada para par (*context*, *question*), donde *context* es una sentencia obtenida en la etapa RI y *question* es cada una de las 45 preguntas del EPIC-QA, se obtiene con Transformers una nueva puntuación ($\boldsymbol{score_{QA}}$) que representa la probabilidad asociada a la respuesta. Por lo tanto, toma valores entre 0 y 1.

## 3.3. Post-procesamiento

Como paso final, se ha realizaso un post-procesamiento de los resultados que consiste en unir sentencias adyacentes que conforman la respuesta. Es decir, supongamos que se ha generado el fichero TXT que será evaluado por el script Python `epic_eval.py`:




| | | | | | |
|---|---|---|---|---|---|
| EQ001 | Q0 | dv9m19yk-C000-S002:dv9m19yk-C000-S002 | 1 | 8.955444 | EXAMPLE_RUN |
| EQ001 | Q0 | dv9m19yk-C000-S003:dv9m19yk-C000-S003 | 2 | 8.574084 | EXAMPLE_RUN |
| EQ001 | Q0 | dv9m19yk-C000-S000:dv9m19yk-C000-S000 | 3 | 7.168122 | EXAMPLE_RUN |
| EQ001 | Q0 | dv9m19yk-C000-S001:dv9m19yk-C000-S001 | 4 | 7.126409 | EXAMPLE_RUN |
| EQ001 | Q0 | hrntvohh-C001-S003:hrntvohh-C001-S003 | 5 | 6.523511 | EXAMPLE_RUN |
| EQ001 | Q0 | 3hgzjf1r-C004-S001:3hgzjf1r-C004-S001 | 6 | 5.964214 | EXAMPLE_RUN |
| EQ001 | Q0 | dckuhrlf-C000-S001:dckuhrlf-C000-S001 | 7 | 5.950470 | EXAMPLE_RUN |
| EQ001 | Q0 | hrntvohh-C001-S001:hrntvohh-C001-S001 | 8 | 5.794175 | EXAMPLE_RUN |
| EQ001 | Q0 | 22ioujwl-C008-S001:22ioujwl-C008-S001 | 9 | 5.601253 | EXAMPLE_RUN |
| EQ001 | Q0 | 3hgzjf1r-C004-S004:3hgzjf1r-C004-S004 | 10 | 5.515963 | EXAMPLE_RUN |
| EQ001 | Q0 | hrntvohh-C001-S004:hrntvohh-C001-S004 | 11 | 5.217576 | EXAMPLE_RUN |
| EQ001 | Q0 | i2ppoe55-C010-S002:i2ppoe55-C010-S002 | 12 | 5.171515 | EXAMPLE_RUN |
| EQ001 | Q0 | hrntvohh-C001-S002:hrntvohh-C001-S002 | 13 | 5.167434 | EXAMPLE_RUN |
| EQ001 | Q0 | hrntvohh-C001-S005:hrntvohh-C001-S005 | 14 | 5.160805 | EXAMPLE_RUN |
| EQ001 | Q0 | hrntvohh-C001-S006:hrntvohh-C001-S006 | 15 | 5.054421 | EXAMPLE_RUN |
| EQ001 | Q0 | hrntvohh-C001-S000:hrntvohh-C001-S000 | 16 | 5.038686 | EXAMPLE_RUN |

**Figura 3.11 - Ejemplo del fichero respMAP1QTN.txt**

Las respuestas de las líneas 1-2 se pueden unir. Lo mismo pasa con las respuestas de las líneas 3-4 y 14-15. Como valor de "score" de las dos líneas unidas pondríamos el valor máximo de los scores de ambas líneas. El resultado sería el siguiente:

| | | | | | |
|---|---|---|---|---|---|
| EQ001 | Q0 | dv9m19yk-C000-S002:dv9m19yk-C000-S003 | 1 | 8.955444 | EXAMPLE_RUN |
| EQ001 | Q0 | dv9m19yk-C000-S000:dv9m19yk-C000-S001 | 2 | 7.168122 | EXAMPLE_RUN |
| EQ001 | Q0 | hrntvohh-C001-S003:hrntvohh-C001-S003 | 3 | 6.523511 | EXAMPLE_RUN |
| EQ001 | Q0 | 3hgzjf1r-C004-S001:3hgzjf1r-C004-S001 | 4 | 5.964214 | EXAMPLE_RUN |
| EQ001 | Q0 | dckuhrlf-C000-S001:dckuhrlf-C000-S001 | 5 | 5.950470 | EXAMPLE_RUN |
| EQ001 | Q0 | hrntvohh-C001-S001:hrntvohh-C001-S001 | 6 | 5.794175 | EXAMPLE_RUN |
| EQ001 | Q0 | 22ioujwl-C008-S001:22ioujwl-C008-S001 | 7 | 5.601253 | EXAMPLE_RUN |
| EQ001 | Q0 | 3hgzjf1r-C004-S004:3hgzjf1r-C004-S004 | 8 | 5.515963 | EXAMPLE_RUN |
| EQ001 | Q0 | hrntvohh-C001-S004:hrntvohh-C001-S004 | 9 | 5.217576 | EXAMPLE_RUN |
| EQ001 | Q0 | i2ppoe55-C010-S002:i2ppoe55-C010-S002 | 10 | 5.171515 | EXAMPLE_RUN |
| EQ001 | Q0 | hrntvohh-C001-S002:hrntvohh-C001-S002 | 11 | 5.167434 | EXAMPLE_RUN |
| EQ001 | Q0 | hrntvohh-C001-S005:hrntvohh-C001-S006 | 12 | 5.160805 | EXAMPLE_RUN |
| EQ001 | Q0 | hrntvohh-C001-S000:hrntvohh-C001-S000 | 13 | 5.038686 | EXAMPLE_RUN |
| EQ001 | Q0 | xuczplaf-C011-S004:xuczplaf-C011-S004 | 14 | 4.986574 | EXAMPLE_RUN |
| EQ001 | Q0 | xqqn1t4e-C000-S002:xqqn1t4e-C000-S002 | 15 | 4.847782 | EXAMPLE_RUN |
| EQ001 | Q0 | vrqe7kxm-C002-S000:vrqe7kxm-C002-S000 | 16 | 4.803590 | EXAMPLE_RUN |

**Figura 3.12 - Ejemplo del fichero respMAP1QTN.txt con merge de sentencias**

También se ha probado a calcular el valor medio (AVG) en vez del valor máximo (MAX) pero los resultados obtenidos eran los mismos, y el valor máximo era mas fácil de calcular. Al evaluar, con el script Python `epic_eval.py`, este fichero TXT con las sentencias unidas se obtiene una mejora para las métricas NDNS-Partial y NDNS-Relaxed y un empeoramiento para la métrica NDNS-Exact.

## 3.4. Combinación de evidencias y ranking de respuestas

Las etapas secuenciales de Recuperación de Información (RI) y Respuesta a Preguntas (QA) son eventos mutuamente excluyentes. Teniendo en cuenta las dos puntuaciones obtenidas (ver Figura 3.4):

- $\boldsymbol{score_{RI}}$ : que es la puntuación que obtenemos, al recuperar la información (RI), de Elasticsearch, y
- $\boldsymbol{score_{QA}}$ : que es la puntuación de obtenemos, al responder a preguntas (QA), con la función `pipeline()` de la librería Transformers.

De las dos puntuaciones anteriores obtenemos una nueva puntuación que será combinación lineal de ellas:

$$score_{NEW} = k_1 * score_{RI} + k_2 * score_{QA} \quad (1)$$

Debido a que $score_{QA}$ toma valores en el intervalo cerrado [0, 1], escalaremos $score_{RI}$ para que también tome valores en [0, 1].

Los valores de las constantes $k_1$ y $k_2$ se determinan experimentalmente, asignándoles valores entre 0 y 10.

| $k_1$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $k_2$ | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Tabla 3.1 - Valores de k1 y k2**

Así se determinará que valores de $k_1$ y $k_2$ hacen que se consigan mejores resultados.

Este nuevo valor de $score_{NEW}$ será el que se use en el fichero TXT que se evalúa con el script Python `epic_eval.py`.

Capítulo 4

# Marco de evaluación

Este capítulo describe la metodología utilizada para evaluar el sistema propuesto, a la vez que se describen las métricas de evaluación y las colecciones de evaluación de distintos dominios.

## 4.1. Metodología de evaluación

Para evaluar el sistema se usará el script de evaluación, que está escrito en Python, y se encuentra disponible para su descarga desde la URL:

`https://bionlp.nlm.nih.gov/epic_qa/data/epic_eval.zip`

El script `epic_eval.py` se usa de la siguiente forma:

```
epic eval.py [-h] [--task {expert,consumer,all}]
                  [--metrics {NDNS-Partial,NDNS-Relaxed,NDNS-Exact,NDCG-Exact,NDCG}]
                  judgment_file submission_file [ideal_ranking_file]
```

Los dos argumentos obligatorios son: `judgment_file, submission_file.`

El fichero `judgment_file` contiene la lista de respuestas generadas por humanos y anotaciones de respuestas a nivel de sentencia para las 21 preguntas evaluadas en la Tarea A y las 18 preguntas evaluadas en la Tarea B. El fichero `judgment_file` se puede descargar desde la siguiente URL:

`https://bionlp.nlm.nih.gov/epic_qa/data/prelim_judgments_corrected.json.gz`

El fichero `submission_file` es el fichero que genera nuestro sistema y queremos testear.

Debido a que solo se ha contemplado el responder a las preguntas planteadas en la Tarea A se usa también el argumento `-- tast expert`.

En resumen, se ejecuta el script Python con los siguientes argumentos:

```
python epic_eval.py --task expert corrected_judgments.json RESPUESTAS.TXT
```

## 4.2. Métricas de evaluación y escenario asociado a cada métrica

La métrica de evaluación primaria favorecerá la exploración del panorama de respuestas (o *nuggets*) en la colección, alentando a los sistemas a proporcionar una lista diversa de respuestas. Específicamente, se clasificarán las respuestas usando una versión modificada de Ganancia Acumulada Descontada Normalizada (Normalized Discounted Cumulative Gain, NDCG) donde:

1. una respuesta se considera relevante si y solo si describe un *nugget* que no se ha incluido en ninguna de las respuestas en los rangos anteriores en el lista y
2. las respuestas se penalizan en función de su longitud (en sentencias).

La métrica de evaluación se basa en una forma modificada de Ganancia Acumulada Descontada Normalizada (Normalized Discounted Cumulative Gain, NDCG) llamada Puntuación de Novedad de Descuento Normalizada (Normalized Discount Novelty Score, NDNS). NDNS utiliza la siguiente puntuación de novedad (*NS*):

$$NS(a) = \frac{n_a(n_a + 1)}{n_a + f_a} \tag{1}$$

Donde $n_a$ es el número de nuevos *nuggets* en la respuesta *a* y $f_a$ es el factor de sentencia. Un *nugget* se considera nuevo si no ha estado presente en una respuesta recuperada anteriormente en la lista clasificada. El factor de sentencia $f_a$ es diferente para cada una de las tres métricas consideradas:

- **NDNS Exact**: las respuestas deben ser breves y no deben contener sentencias con *nuggets* suministrados en respuestas anteriores. Aquí $f_a$ es el número de sentencias en la respuesta.

$$f_a = s_a = s_0 + s_s + s_n \tag{2}$$

  Donde $s_a$ es el número de sentencias en la respuesta *a*, $s_0$ es el número de sentencias sin *nuggets*, $s_s$ es el número de sentencias con *nuggets* ya vistos y $s_n$ es el número de sentencias con nuevos *nuggets*.

- **NDNS Relaxed**: las respuestas deberían solamente contener sentencias con nuevos *nuggets*.

$$f_a = s_0 + s_s + min(s_n, 1) \tag{3}$$

- **NDNS Partial**: las respuestas no deberían contener sentencias sin *nuggets*.

$$f_a = s_0 + min(s_n, 1) \tag{4}$$

Al igual que en NDCG se calcula la puntuación de novedad acumulada NS de las respuestas recuperadas hasta el rango $l$ y descontamos la puntuación utilizando un factor de reducción logarítmico:

$$DNS(a_1, \dots, a_l) = \sum_{r=1}^{l} \frac{NS(a_r)}{log_2(r+1)} \tag{5}$$

Finalmente, normalizamos el NDNS de ranking $\boldsymbol{a} = a_1, \cdots, a_l$ por el NDNS del ranking óptimo o ideal de posibles respuestas que podrían haberse recuperado para esa pregunta:

$$NDNS(\boldsymbol{a}) = \frac{NDNS(\boldsymbol{a})}{NDNS(\widehat{\boldsymbol{a}})} \tag{6}$$

donde $\hat{a}$ es la clasificación óptima de respuestas que se podrían haber encontrado en la colección de documentos para la pregunta dada. En la evaluación, se usa una búsqueda de haz con un ancho de 10 para determinar la clasificación ideal de respuestas.

Cuando ejecutamos el script `epic_eval.py` nos calcula los tres parámetros, NDNS-Partial, NDNS-Relaxed y NDNS-Exact, para cada pregunta y al final la media:

```
EXAMPLE_RUN     EQ045   NDNS-Partial    0.11745228929961338
EXAMPLE_RUN     EQ045   NDNS-Relaxed    0.11745228929961338
EXAMPLE_RUN     EQ045   NDNS-Exact      0.12380780719084086
EXAMPLE_RUN     MEAN    NDNS-Partial    0.21782302856797536
EXAMPLE_RUN     MEAN    NDNS-Relaxed    0.21901773414354705
EXAMPLE_RUN     MEAN    NDNS-Exact      0.2447873666666533
```

**Figura 4.1 - Ejemplo de respuesta del script `epic_eval.py`**

## 4.3. Colecciones de evaluación

Para este trabajo se parte de la colección de la ronda preliminar de CORD-19. Consiste en una colección de artículos de biomédicos liberada para el CORD-19 (COVID-19 Open Research Dataset Challenge). Dicha colección (972MB) puede descargarse en la URL siguiente:

`https://bionlp.nlm.nih.gov/epic_qa/data/epic_qa_cord_2020-06-19_v3.tar.gz`

y que consta de 129.069 documentos JSON. Estos documentos están estructurados en contextos que corresponden a pasajes (párrafos) definidos por los autores de sus publicaciones. Los contextos que tienen más de 15 sentencias se segmentarán en bloques de aproximadamente 15 sentencias. Estos documentos JSON contienen los identificadores de documento, de contexto y de sentencia (`document_id`, `context_id`, `sentence_id`), y estos datos se suministran a los participantes.

## 4.4. Líneas base

Los resultados obtenidos con el sistema de recuperación de información (RI) constituyen la línea base de la que se ha partido. De los dos índices de los que partíamos (pasajes y sentencias) solo vamos a utilizar el índice de pasajes y de los tres campos de cada pregunta (`"question"`," `query`" y `"background"`) solo vamos a considerar la consulta por el campo `"question"`.

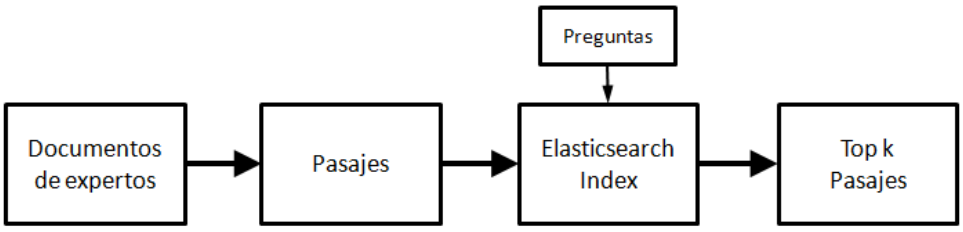



**Figura 4.2 - Modelo de recuperación de información (RI) que constituye la linea base**

## 4.5. Evaluación de $k_1$ y $k_2$

El módulo de RI nos proporciona una puntuación ($score_{RI}$), al igual que el módulo de QA ($score_{QA}$), que usamos para clasificar los resultados. La puntuación del módulo de RI, $score_{RI}$, debemos escalarla para que se ajuste al intervalo cerrado [0, 1]. En este caso la puntuación del módulo QA, ya se encuentra en el intervalo cerrado [0, 1], y por lo tanto no es preciso escalarla.

Estos dos módulos se aplican secuencialmente, y nos van a permitir calcular una nueva puntuación ($score_{NEW}$) que resulta de la combinación lineal de las puntuaciones del modulo RI y QA:

$$score_{NEW} = k_1 score_{RI} + k_2 score_{QA} \quad (7)$$

$k_1$ y $k_2$ son los pesos que asignamos a la puntuación de cada modulo (RI y QA). $k_1$ y $k_2$ deberían tomar valores entre 0 y 1, es decir, [0, 0.1, 0.2, ..., 1]. Sin embargo, les asignamos valores entre 0 y 10. Los resultados son los mismos ya que el script Python `epic_eval.py` escala las puntuaciones.

Utilizaremos este nuevo valor, $score_{NEW}$, para puntuar las sentencias, después evaluaremos el resultado y obtendremos las métricas *NDNS-Exact*, *NDND-Partial* y *NDNS-Relaxed*. Representaremos gráficamente estas métricas frente a valores de $k_1$. Y de estas graficas obtendremos los valores de $k_1$ que maximizan los valores de las métricas. Es decir, que valores de $k_1$ son más adecuados para una determinada métrica.

Capítulo 5

# Experimentación y resultados

Este capítulo analiza y discute en profundidad los resultados obtenidos en la evaluación presentada en el capítulo anterior. Se analizan los datos obtenidos de los dos índices (pasajes y sentencias), así como las consultas realizadas usando diferentes campos: `query`, `question` y `background`. También se analizan los distintos tipos de modelos usados en el sistema neuronal.

## 5.1. Indexación de sentencias o pasajes

Con el sistema de recuperación de información de índices y pasajes se han realizado las consultas por los campos "`query`", "`question`" y "`background`" y los resultados son los siguientes:

| Indexación | Consulta | NDNS-Partial | NDNS-Relaxed | NDNS-Exact |
|---|---|---|---|---|
| Pasajes | query | 0,1101 | 0,0978 | 0,0893 |
| | question | 0,2337 | 0,2171 | 0,1955 |
| | background | 0,1336 | 0,1248 | 0,1060 |
| Sentencias | query | 0,0722 | 0,0726 | 0,0803 |
| | question | 0,1408 | 0,1415 | 0,1534 |
| | background | 0,0741 | 0,0745 | 0,0828 |

**Tabla 5.1 - Resultados obtenidos del sistema de recuperacion de informacion de indices y pasajes usando los campos: query, question y background**

Se observa en la tabla que, el sistema de recuperación de información, obtienen mejores resultados cuando se recuperan pasajes (0.2337, 0.2171, 0.1955) que cuando se recuperan sentencias (0.1408, 0.1415, 0.1534). Esto es así con independencia de la métrica elegida.

## 5.2. Consulta con *question*, *query* o *background*

También en la tabla anterior se observa que, el sistema de recuperación de información de pasajes y sentencias, obtiene mejores resultados cuando en la consulta se utiliza el campo "`question`".: (0.2337, 0.2171, 0.1955). Igual que antes, esto es independiente de la métrica usada.

## 5.3. Eficacia de los diferentes modelos neuronales

Se analiza cada modelo neuronal partiendo del sistema de recuperación de información de pasajes mediante el campo "question".

### 5.3.1. Modelo: deepset/roberta-base-squad2

El modelo que usa es `roberta-base` ajustado para QA, evaluado en SQuAD 2.0 con el script de evaluación oficial.

Este modelo alcalza una puntuación:

- "F1": 83.00449234495325
- "Exact": 79.97136359807968

Los resultados obtenidos con este modelo para los distintos valores de $k_1$ son los siguientes:

| valor de $k_1$ | NDNS-Partial | NDNS-Relaxed | NDNS-Exact |
|---|---|---|---|
| 0 | 0,07977127862 | 0,08007141501 | 0,08912237897 |
| 1 | 0,085020108 | 0,085358838 | 0,094657732 |
| 2 | 0,09396631 | 0,094328344 | 0,105043978 |
| 3 | 0,111733168 | 0,112176564 | 0,125030111 |
| 4 | 0,125665681 | 0,126231051 | 0,140619927 |
| 5 | 0,153649374 | 0,154504416 | 0,17248901 |
| 6 | 0,186867657 | 0,187815927 | 0,208856786 |
| 7 | 0,193911939 | 0,19493275 | 0,216766275 |
| 8 | 0,202752414 | 0,203836466 | 0,227235774 |
| 9 | 0,204585747 | 0,20573104 | 0,229365596 |
| 10 | 0,1560223492 | 0,1567413794 | 0,177062277 |

**Tabla 5.2 - Resultados de las tres métricas obtenidos del sistema RI + QA + Re-ranking para los distintos valores de $k_1$ del modelo roberta-base-squad2**

De la tabla anterior se deduce que los resultados mas relevantes se obtienen cuando $k_1$ = 9, y esto con independencia de la métrica.

Si representamos gráficamente los resultados de la tabla anterior se obtiene la gráfica siguiente:

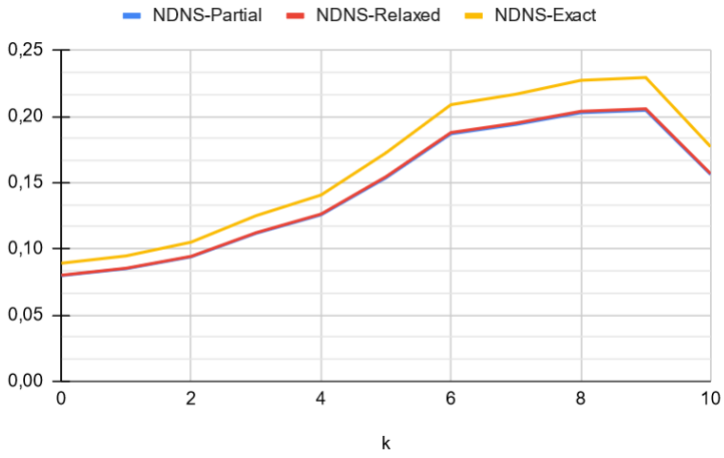



**Figura 5.1 - Gráfica de las tres métricas frente al valor de $k_1$ del modelo roberta-base-squad2**

### 5.3.2. Modelo: ktrapeznikov/biobert_v1.1_pubmed_squad_v2

El modelo que usa es `monolog /biobert_v1.1_pubmed` ajustado en SQuAD 2.0 con el script de evaluación oficial

Este modelo alcalza una puntuación:

- "F1": 79.37043950121722

- "Exact": 75.97068980038743

Los resultados obtenidos con este modelo para los distintos valores de $k_1$ son los siguientes:

| valor de $k_1$ | NDNS-Partial | NDNS-Relaxed | NDNS-Exact |
|---|---|---|---|
| 0 | 0,1034897491 | 0,1041257932 | 0,1175144207 |
| 1 | 0,118901929 | 0,119576532 | 0,134506389 |
| 2 | 0,1299775791 | 0,130686397 | 0,146964449 |
| 3 | 0,1483932914 | 0,149346941 | 0,168276129 |
| 4 | 0,1654971991 | 0,166581161 | 0,187862382 |
| 5 | 0,1841015485 | 0,185275997 | 0,208977937 |
| 6 | 0,2078393839 | 0,209123156 | 0,235424493 |
| 7 | 0,2248794752 | 0,226252788 | 0,253167562 |
| 8 | 0,2253357973 | 0,226803561 | 0,25377557 |
| 9 | 0,220155462 | 0,221562281 | 0,2477915318 |
| 10 | 0,1560223492 | 0,1567413794 | 0,177062277 |

**Tabla 5.3 - Resultados de las tres métricas obtenidos del sistema RI + QA + Re-ranking para los distintos valores de k1 del modelo biobert_v1.1_pubmed_squad_v2**

De la tabla anterior se deduce que los resultados mas relevantes se obtienen cuando $k_1 = 8$, y esto con independencia de la métrica.

Si representamos gráficamente los resultados de la tabla anterior se obtiene la gráfica siguiente:

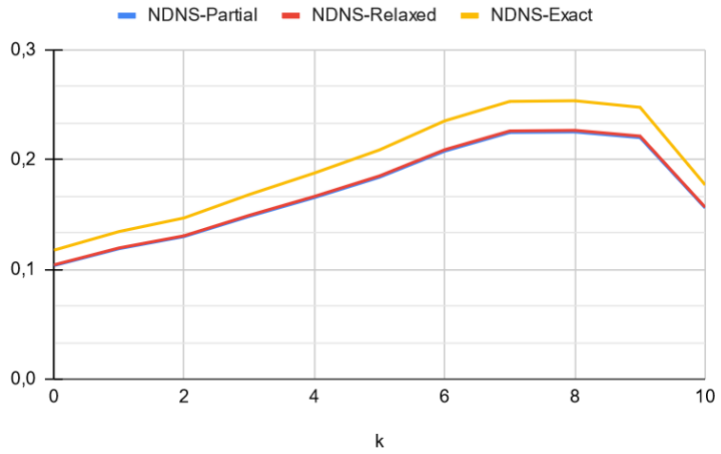



**Figura 5.2 - Gráfica de las tres métricas frente al valor de k1 del modelo biobert_v1.1_pubmed_squad_v2**

### 5.3.3. Modelo: distilbert-base-uncased-distilled-squad

El modelo que usa es `distilbert-base-uncased` ajustado en SQuAD v1.1.

Este modelo alcalza una puntuación:

- "F1": 86,9 (a modo de comparación, la versión BERT bert-base-uncased alcanza una puntuación F1 de 88,5).

Los resultados obtenidos con este modelo para los distintos valores de $k_1$ son los siguientes:

| valor de $k_1$ | NDNS-Partial | NDNS-Relaxed | NDNS-Exact |
|---|---|---|---|
| 0 | 0,07112449766 | 0,07156687875 | 0,08016953159 |
| 1 | 0,091282304 | 0,091789707 | 0,102403481 |
| 2 | 0,1035592274 | 0,1041361907 | 0,1165203812 |
| 3 | 0,121645907 | 0,122418951 | 0,136649249 |
| 4 | 0,143773478 | 0,144616324 | 0,1612605 |

| | | | |
|---|---|---|---|
| 5 | 0,172202361 | 0,17319727 | 0,192950355 |
| 6 | 0,188992288 | 0,190121359 | 0,211703409 |
| 7 | 0,197538049 | 0,198738727 | 0,221961482 |
| 8 | 0,202815873 | 0,204033193 | 0,227676213 |
| 9 | 0,201750135 | 0,202998761 | 0,226884565 |
| 10 | 0,1560223492 | 0,1567413794 | 0,177062277 |

**Tabla 5.4 - Resultados de las tres métricas obtenidos del sistema RI + QA + Re-ranking para los distintos valores de k1 del modelo distilbert-base-uncased-distilled-squad**

De la tabla anterior se deduce que los resultados mas relevantes se obtienen cuando $k_1$ = 8, y esto con independencia de la métrica.

Si representamos gráficamente los resultados de la tabla anterior se obtiene la gráfica siguiente:

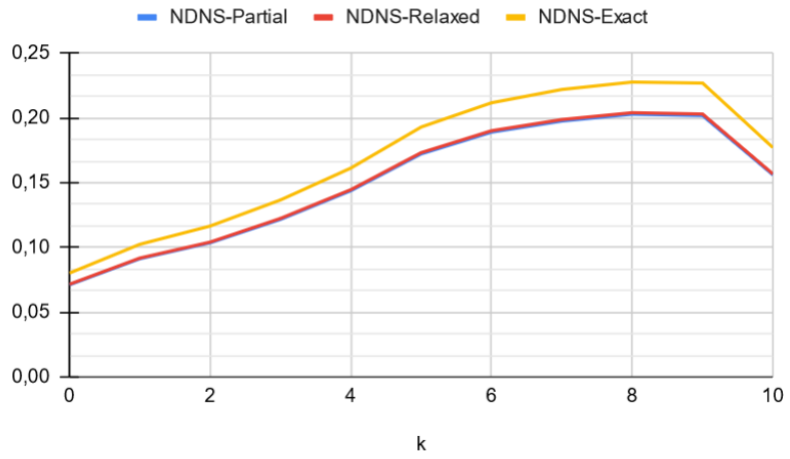



**Figura 5.3 - Gráfica de las tres métricas frente al valor de k1 del modelo distilbert-base-uncased-distilled-squad**

### 5.3.4. Modelo: distilbert-base-cased-distilled-squad

El modelo que usa es `distilbert-base-cased` ajustado en SQuAD v1.1.

Este modelo alcalza una puntuación:

- "F1": 87.1 (a modo de comparación, la versión BERT bert-base-cased alcanza una puntuación F1 de 88.7).

Los resultados obtenidos con este modelo para los distintos valores de $k_1$ son los siguientes:

| valor de $k_1$ | NDNS-Partial | NDNS-Relaxed | NDNS-Exact |
|---|---|---|---|
| 0 | 0,07937960035 | 0,07983651036 | 0,08907302908 |
| 1 | 0,1094459986 | 0,1099386216 | 0,121893269 |
| 2 | 0,1268116979 | 0,1275329071 | 0,140957195 |
| 3 | 0,1340460388 | 0,1348006169 | 0,149380696 |
| 4 | 0,1446735332 | 0,1455408686 | 0,1622477206 |
| 5 | 0,1646857 | 0,1657052523 | 0,1858711302 |
| 6 | 0,1975902289 | 0,1986902359 | 0,2202757185 |
| 7 | 0,2037395348 | 0,2049839433 | 0,2282705182 |
| 8 | 0,211452652 | 0,2128456005 | 0,23783896 |
| 9 | 0,2138581836 | 0,2152585395 | 0,2407056261 |
| 10 | 0,1560223492 | 0,1567413794 | 0,177062277 |

**Tabla 5.5 - Resultados de las tres métricas obtenidos del sistema RI + QA + Re-ranking para los distintos valores de k1 del modelo distilbert-base-cased-distilled-squad**

De la tabla anterior se deduce que los resultados mas relevantes se obtienen cuando $k_1$ = 9, y esto con independencia de la métrica.

Si representamos gráficamente los resultados de la tabla anterior se obtiene la gráfica siguiente:

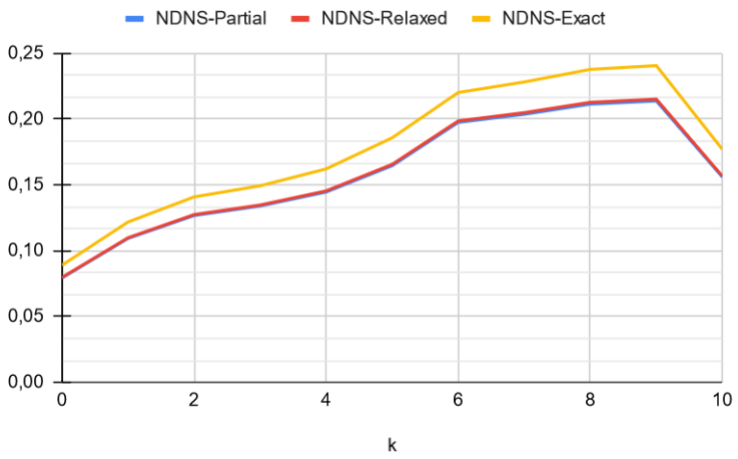



**Figura 5.4 - Gráfica de las tres métricas frente al valor de k1 del modelo distilbert-base-cased-distilled-squad**

### 5.3.5. Modelo: deepset/bert-large-uncased-whole-word-masking-squad2

Los resultados obtenidos con este modelo para los distintos valores de $k_1$ son los siguientes:

| valor de $k_1$ | NDNS-Partial | NDNS-Relaxed | NDNS-Exact |
|---|---|---|---|
| 0 | 0,1156343947 | 0,1163172877 | 0,1309430741 |
| 1 | 0,129271314 | 0,130004955 | 0,146387759 |
| 2 | 0,14049727 | 0,14127769 | 0,159696333 |
| 3 | 0,155535374 | 0,156393216 | 0,176750069 |
| 4 | 0,170812009 | 0,171769753 | 0,194109285 |
| 5 | 0,204312611 | 0,205429709 | 0,229938485 |
| 6 | 0,220344091 | 0,221577196 | 0,247485227 |
| 7 | 0,225413222 | 0,226578751 | 0,252213137 |
| 8 | 0,217823029 | 0,219017734 | 0,244787367 |
| 9 | 0,211575226 | 0,212768013 | 0,237648846 |
| 10 | 0,1560223492 | 0,1567413794 | 0,177062277 |

**Tabla 5.6 - Resultados de las tres métricas obtenidos del sistema RI + QA + Re-ranking para los distintos valores de k1 del modelo bert-base-uncased-whole-word-masked-squad2**

De la tabla anterior se deduce que los resultados mas relevantes se obtienen cuando $k_1$ = 7, y esto con independencia de la métrica.

Si representamos gráficamente los resultados de la tabla anterior se obtiene la gráfica siguiente:

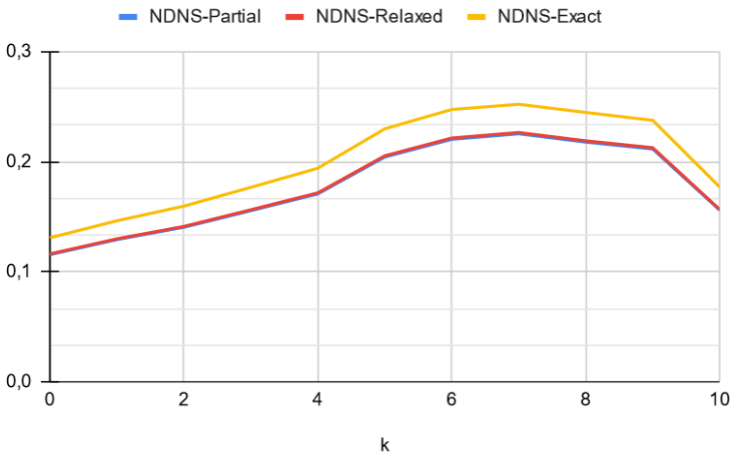



**Figura 5.5 - Gráfica de las tres métricas frente al valor de k1 del modelo bert-base-uncased-whole-word-masked-squad2**

### 5.3.6. Modelo: ktrapeznikov/scibert_scivocab_uncased_squad_v2

El modelo que usa es `allenai/scibert_scivocab_uncased` ajustado en SQuAD 2.0 con el script de evaluación oficial.

Este modelo alcalza una puntuación:

- F1: 78.47735207283013
- Exact: 75.07790785816559

Los resultados obtenidos con este modelo para los distintos valores de $k_1$ son los siguientes:

| valor de $k_1$ | NDNS-Partial | NDNS-Relaxed | NDNS-Exact |
|---|---|---|---|
| 0 | 0,1082511629 | 0,1087841003 | 0,121444918 |
| 1 | 0,1164706353 | 0,1170118658 | 0,1308268026 |
| 2 | 0,1257808985 | 0,1264008624 | 0,1418268824 |
| 3 | 0,1391043357 | 0,1397962238 | 0,1574111468 |
| 4 | 0,1526316253 | 0,1534130219 | 0,1734847577 |
| 5 | 0,1738842321 | 0,1747601064 | 0,1965457485 |
| 6 | 0,1956581337 | 0,1967116341 | 0,2206169812 |
| 7 | 0,2126743316 | 0,2138521563 | 0,2397638917 |
| 8 | 0,2127998966 | 0,213961606 | 0,2403699762 |
| 9 | 0,2084561151 | 0,2095957086 | 0,2349205587 |
| 10 | 0,1560223492 | 0,1567413794 | 0,177062277 |

**Tabla 5.7 - Resultados de las tres métricas obtenidos del sistema RI + QA + Re-ranking para los distintos valores de k1 del modelo scibert_scivocab_uncased_squad_v2**

De la tabla anterior se deduce que los resultados mas relevantes se obtienen cuando $k_1 = 8$, y esto con independencia de la métrica.

Si representamos gráficamente los resultados de la tabla anterior se obtiene la gráfica siguiente:

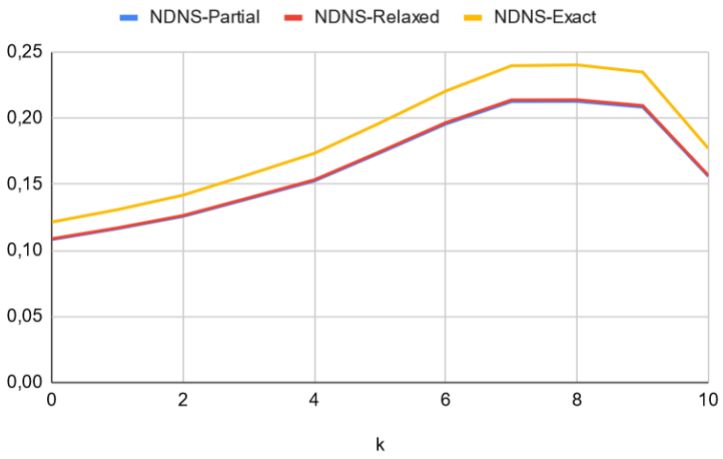



**Figura 5.6 - Gráfica de las tres métricas frente al valor de k1 del modelo scibert_scivocab_uncased_squad_v2**

### 5.3.7. Comparación de todos los modelos

Las siguientes tablas muestran los resultados obtenidos de todos los modelos para cada una de las métricas. También se muestran los mismos resultados en forma gráfica.

Cuando $k_1$ = 0 entonces toda la puntuación es debida a al sistema de recuperación de información (RI). cuando $k_1$ = 10 entonces toda la puntuación es debida al sistema de respuesta a preguntas (QA)

Para la métrica NDNS-Exact los datos son los siguientes:

| NDNS-Exact | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $k_1$ | biobert_squad_v2 | roberta-base-squad2 | distilbert-base-uncased | distibert-base-cased | bert-large-uncased | scibert_uncased_squad_v2 | biobert_squad_v2_sentence_merge | IR passage-to-sentence baseline | IR sentence baseline | IR passage baseline |
| 0 | 0,117514 | 0,089122 | 0,080170 | 0,089073 | 0,130943 | 0,121445 | 0,117538 | 0,232333 | 0,1534 | 0,1955 |
| 1 | 0,134506 | 0,094658 | 0,102403 | 0,121893 | 0,146388 | 0,130827 | 0,134511 | 0,232333 | 0,1534 | 0,1955 |
| 2 | 0,146964 | 0,105044 | 0,116520 | 0,140957 | 0,159696 | 0,141827 | 0,146610 | 0,232333 | 0,1534 | 0,1955 |
| 3 | 0,168276 | 0,125030 | 0,136649 | 0,149381 | 0,176750 | 0,157411 | 0,168118 | 0,232333 | 0,1534 | 0,1955 |
| 4 | 0,187862 | 0,140620 | 0,161261 | 0,162248 | 0,194109 | 0,173485 | 0,186150 | 0,232333 | 0,1534 | 0,1955 |
| 5 | 0,208978 | 0,172489 | 0,192950 | 0,185871 | 0,229938 | 0,196546 | 0,207698 | 0,232333 | 0,1534 | 0,1955 |
| 6 | 0,235424 | 0,208857 | 0,211703 | 0,220276 | 0,247485 | 0,220617 | 0,230410 | 0,232333 | 0,1534 | 0,1955 |
| 7 | 0,253168 | 0,216766 | 0,221961 | 0,228271 | 0,252213 | 0,239764 | 0,251340 | 0,232333 | 0,1534 | 0,1955 |
| 8 | 0,253776 | 0,227236 | 0,227676 | 0,237839 | 0,244787 | 0,240370 | 0,246914 | 0,232333 | 0,1534 | 0,1955 |
| 9 | 0,247792 | 0,229366 | 0,226885 | 0,240706 | 0,237649 | 0,234921 | 0,236467 | 0,232333 | 0,1534 | 0,1955 |
| 10 | 0,177062 | 0,177062 | 0,177062 | 0,177062 | 0,177062 | 0,177062 | 0,139330 | 0,232333 | 0,1534 | 0,1955 |

**Tabla 5.8 – Tabla de todos los modelos para la métrica NDNS-Exact y distintos valores de $k_1$**

De la tabla anterior, para la métrica NDNS-Exact, los resultados más relevantes se obtienen para el modelo biobert_squad_v2 y $k_1$ = 8. Para esta métrica se observa que el hacer un post-procesamiento (merge) de las sentencias empeorará los resultados.

También se observa que, para esta métrica, solo con el módulo de recuperación de información (pasajes) se obtienen mejores resultados que usando RI + QA. Esto ocurre para los modelos `roberta-base-squad2` y `distilbert-base-uncased`.

Si representamos gráficamente los resultados de la tabla anterior se obtiene la gráfica siguiente:

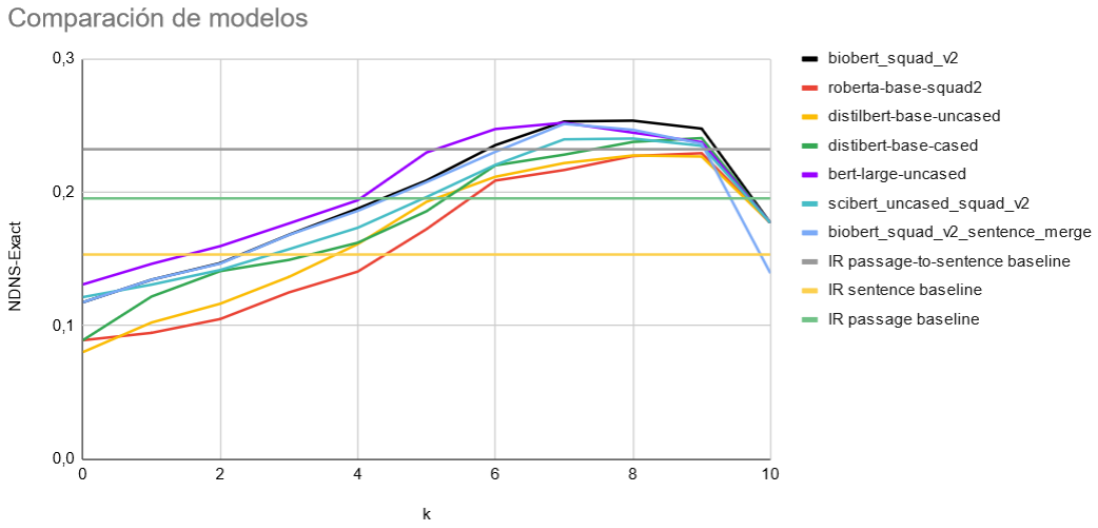



**Figura 5.7 – Gráfica de todos los modelos para la métrica NDNS-Exact y distintos valores de $k_1$**

Para la métrica NDNS-Partial los datos son los siguientes:

| NDNS-Partial | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $k_1$ | biobert_squad_v2 | roberta-base-squad2 | distilbert-base-uncased | distibert-base-cased | bert-large-uncased | scibert_uncased_squad_v2 | biobert_squad_v2_sentence_merge | IR passage-to-sentence baseline | IR sentence baseline | IR passage baseline |
| 0 | 0,103490 | 0,079771 | 0,071124 | 0,079380 | 0,115634 | 0,108251 | 0,103509 | 0,206768 | 0,1408 | 0,2337 |
| 1 | 0,118902 | 0,085020 | 0,091282 | 0,109446 | 0,129271 | 0,116471 | 0,111891 | 0,206768 | 0,1408 | 0,2337 |
| 2 | 0,129978 | 0,093966 | 0,103559 | 0,126812 | 0,140497 | 0,125781 | 0,130012 | 0,206768 | 0,1408 | 0,2337 |
| 3 | 0,148393 | 0,111733 | 0,121646 | 0,134046 | 0,155535 | 0,139104 | 0,148226 | 0,206768 | 0,1408 | 0,2337 |
| 4 | 0,165497 | 0,125666 | 0,143773 | 0,144674 | 0,170812 | 0,152632 | 0,166645 | 0,206768 | 0,1408 | 0,2337 |
| 5 | 0,184102 | 0,153649 | 0,172202 | 0,164686 | 0,204313 | 0,173884 | 0,185488 | 0,206768 | 0,1408 | 0,2337 |
| 6 | 0,207839 | 0,186868 | 0,188992 | 0,197590 | 0,220344 | 0,195658 | 0,210876 | 0,206768 | 0,1408 | 0,2337 |
| 7 | 0,224879 | 0,193912 | 0,197538 | 0,203740 | 0,225413 | 0,212674 | 0,230951 | 0,206768 | 0,1408 | 0,2337 |
| 8 | 0,225336 | 0,202752 | 0,202816 | 0,211453 | 0,217823 | 0,212800 | 0,231121 | 0,206768 | 0,1408 | 0,2337 |
| 9 | 0,220155 | 0,204586 | 0,201750 | 0,213858 | 0,211575 | 0,208456 | 0,225709 | 0,206768 | 0,1408 | 0,2337 |
| 10 | 0,156022 | 0,156022 | 0,156022 | 0,156022 | 0,156022 | 0,156022 | 0,172856 | 0,206768 | 0,1408 | 0,2337 |

**Tabla 5.9 – Tabla de todos los modelos para la métrica NDNS-Partial y distintos valores de $k_1$**

De la tabla anterior, para la métrica NDNS-Partial, los resultados más relevantes se obtienen para el modelo biobert_squad_v2_sentence_merge y $k_1$ = 8. Para esta métrica se observa que el hacer un post-procesamiento (merge) de las sentencias mejorará los resultados.

También se observa que, para esta métrica, solo con el módulo de recuperación de información (pasajes) se obtienen mejores resultados que usando RI + QA.

Si representamos gráficamente los resultados de la tabla anterior se obtiene la gráfica siguiente:

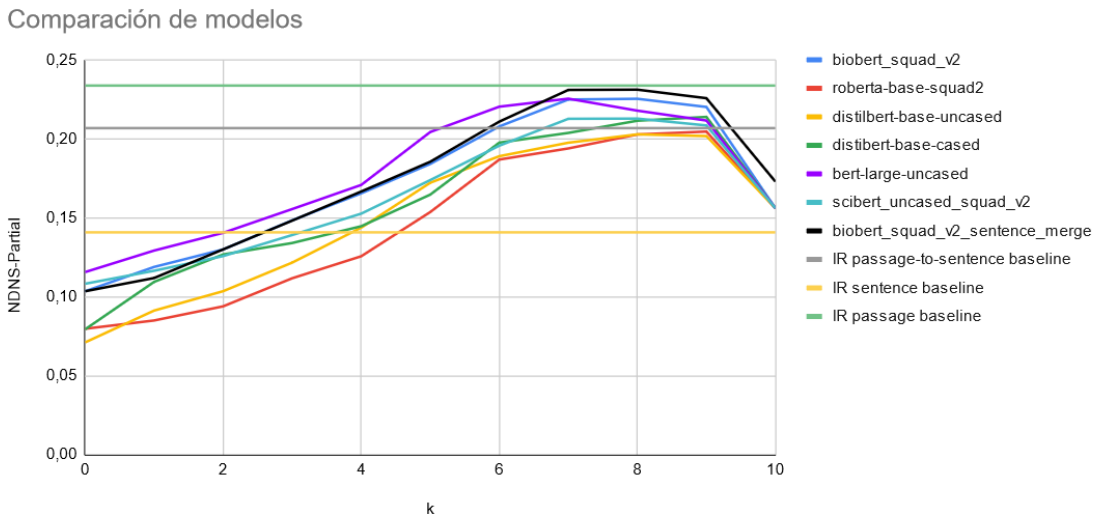



**Figura 5.8 – Gráfica de todos los modelos para la métrica NDNS-Partial y distintos valores de $k_1$**

Para la métrica NDNS-Relaxed los datos son los siguientes:

| NDNS-Relaxed | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $k_1$ | biobert_squad_v2 | roberta-base-squad2 | distilbert-base-uncased | distibert-base-cased | bert-large-uncased | scibert_uncased_squad_v2 | biobert_squad_v2_sentence_merge | IR passage-to-sentence baseline | IR sentence baseline | IR passage baseline |
| 0 | 0,104126 | 0,080071 | 0,071567 | 0,079837 | 0,116317 | 0,108784 | 0,104145 | 0,207923 | 0,1415 | 0,2171 |
| 1 | 0,119577 | 0,085359 | 0,091790 | 0,109939 | 0,130005 | 0,117012 | 0,119580 | 0,207923 | 0,1415 | 0,2171 |
| 2 | 0,130686 | 0,094328 | 0,104136 | 0,127533 | 0,141278 | 0,126401 | 0,130721 | 0,207923 | 0,1415 | 0,2171 |
| 3 | 0,149347 | 0,112177 | 0,122418 | 0,134801 | 0,156393 | 0,139796 | 0,149180 | 0,207923 | 0,1415 | 0,2171 |
| 4 | 0,166581 | 0,126231 | 0,144616 | 0,145541 | 0,171770 | 0,153413 | 0,167384 | 0,207923 | 0,1415 | 0,2171 |
| 5 | 0,185276 | 0,154504 | 0,173197 | 0,165705 | 0,205430 | 0,174760 | 0,186737 | 0,207923 | 0,1415 | 0,2171 |
| 6 | 0,209123 | 0,187816 | 0,190121 | 0,198690 | 0,221577 | 0,196712 | 0,211328 | 0,207923 | 0,1415 | 0,2171 |
| 7 | 0,226253 | 0,194933 | 0,198739 | 0,204984 | 0,226579 | 0,213852 | 0,231176 | 0,207923 | 0,1415 | 0,2171 |
| 8 | 0,226804 | 0,203836 | 0,204033 | 0,212846 | 0,219018 | 0,213962 | 0,230771 | 0,207923 | 0,1415 | 0,2171 |
| 9 | 0,221562 | 0,205731 | 0,202999 | 0,215259 | 0,212768 | 0,209596 | 0,225493 | 0,207923 | 0,1415 | 0,2171 |
| 10 | 0,156741 | 0,156741 | 0,156741 | 0,156741 | 0,156741 | 0,156741 | 0,156638 | 0,207923 | 0,1415 | 0,2171 |

**Tabla 5.10 – Tabla de todos los modelos para la métrica NDNS-Relaxed y distintos valores de $k_1$**

De la tabla anterior, para la métrica NDNS-Relaxed, los resultados más relevantes se obtienen para el modelo biobert_squad_v2_sentence_merge y $k_1$ = 7. Para esta métrica se observa que el hacer un post-procesamiento (merge) de las sentencias mejorará los resultados.

También se observa que, para esta métrica, solo con el módulo de recuperación de información (pasajes) se obtienen mejores resultados que usando RI + QA.

Si representamos gráficamente los resultados de la tabla anterior se obtiene la gráfica siguiente:

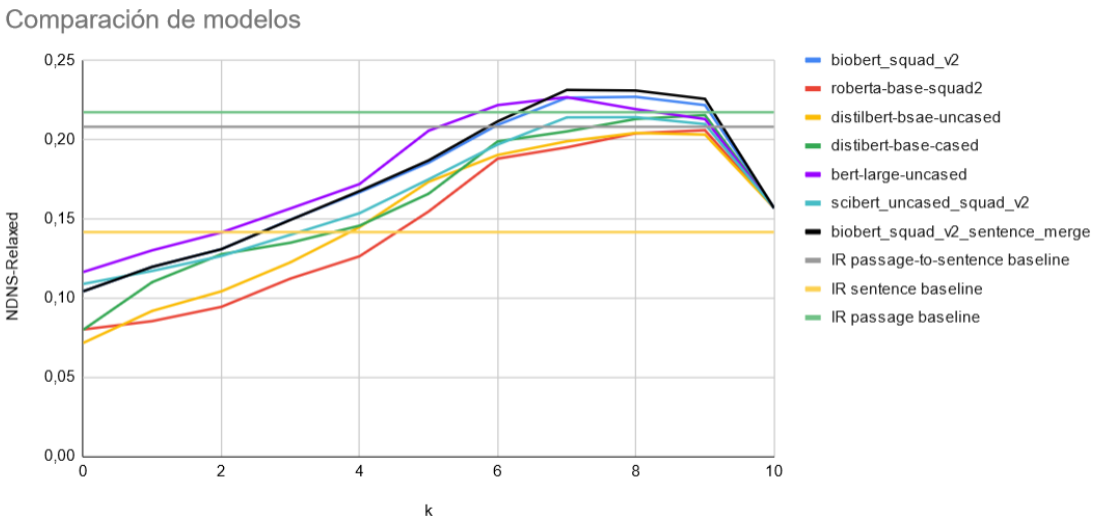



**Figura 5.9 – Gráfica de todos los modelos para la métrica NDNS-Relaxed y distintos valores de $k_1$**

## 5.4. Efecto del post-procesamiento adaptado a cada escenario

Para probar la mejoras del post-procesamiento se usa el modelo: `ktrapeznikov/biobert_v1.1_pubmed_squad_v2`. Los resultados obtenidos con y sin post-procesamiento (merge) se pueden ver en la siguiente tabla:

| | Modelo + MERGE | | | Modelo | | |
|---|---|---|---|---|---|---|
| **valor de $k_1$** | **NDNS-Partial** | **NDNS-Relaxed** | **NDNS-Exact** | **NDNS-Partial** | **NDNS-Relaxed** | **NDNS-Exact** |
| 0 | 0,1035090416 | 0,1041452085 | 0,1175378803 | 0,1034897491 | 0,1041257932 | 0,1175144207 |
| 1 | 0,1118905454 | 0,1195801141 | 0,1345111005 | 0,118901929 | 0,119576532 | 0,134506389 |
| 2 | 0,1300119198 | 0,13072117 | 0,1466101519 | 0,1299775791 | 0,130686397 | 0,146964449 |
| 3 | 0,1482262614 | 0,1491799893 | 0,168117503 | 0,1483932914 | 0,149346941 | 0,168276129 |
| 4 | 0,166644845 | 0,1673843688 | 0,1861500952 | 0,1654971991 | 0,166581161 | 0,187862382 |
| 5 | 0,1854875565 | 0,1867369994 | 0,2076984571 | 0,1841015485 | 0,185275997 | 0,208977937 |
| 6 | 0,2108760971 | 0,211328181 | 0,2304099268 | 0,2078393839 | 0,209123156 | 0,235424493 |
| 7 | 0,2309508628 | 0,2311763867 | 0,2513395356 | 0,2248794752 | 0,226252788 | 0,253167562 |
| 8 | 0,2311209878 | 0,2307711303 | 0,2469143759 | 0,2253357973 | 0,226803561 | 0,25377557 |
| 9 | 0,2257085822 | 0,2254928532 | 0,2364665284 | 0,220155462 | 0,221562281 | 0,2477915318 |
| 10 | 0,1728562111 | 0,1566378341 | 0,1393303061 | 0,1560223492 | 0,1567413794 | 0,177062277 |

**Tabla 5.11 - Resultados de las tres métricas obtenidos del sistema RI + QA + Re-ranking para los distintos valores de k1 del modelo biobert_v1.1_pubmed_squad_v2 con y sin post-procesamiento (merge)**

Para la métrica NDNS-Partial, se representan los datos obtenidos por el sistema con y sin post-procesamiento

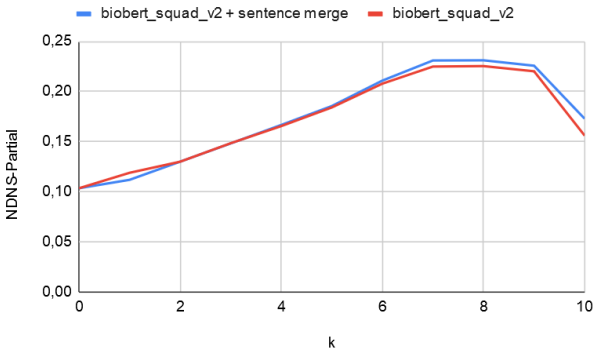


**Figura 5.10 – Gráficas de la métrica NDNS-Partial de los Modelos con y sin Post-procesamiento (Merge) para los distintos valores de $k_1$**

De la gráfica y la tabla, se puede concluir que el valor de la métrica NDNS-Partial es mayor cuando se realiza el post-procesamiento y que este valor se alcanza cuando $k_1$ es igual a 8.

Para la métrica NDNS-Relaxed, se representan los datos obtenidos por el sistema con y sin post-procesamiento

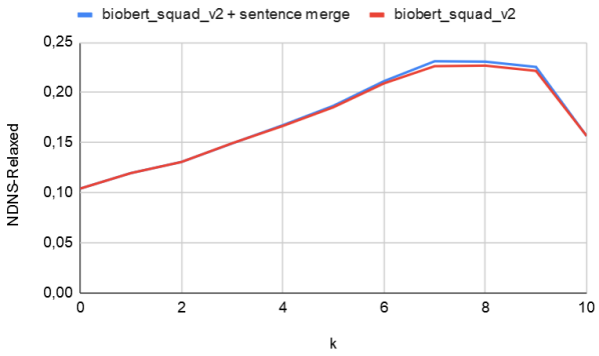



**Figura 5.11 – Gráficas de la métrica NDNS-Relaxed de los Modelos con y sin Post-procesamiento (Merge) para los distintos valores de $k_1$**

De la gráfica y la tabla, se puede concluir que el valor de la métrica NDNS-Relaxed es mayor cuando se realiza el post-procesamiento y que este valor se alcanza cuando $k_1$ es igual a 8.

Por último para la métrica NDNS-Exact, se representan los datos obtenidos por el sistema con y sin post-procesamiento

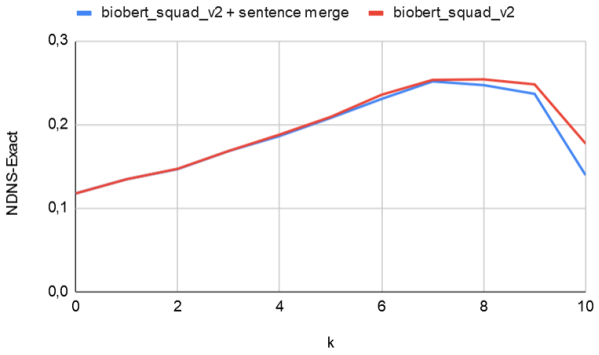



**Figura 5.12 – Gráficas de la métrica NDNS-Exact de los Modelos con y sin Post-procesamiento (Merge) para los distintos valores de $k_1$**

De la gráfica y la tabla, se puede concluir que el valor de la métrica NDNS-Exact es menor cuando se realiza el post-procesamiento y que este valor se alcanza cuando $k_1$ es igual a 7.

## 5.5. Efecto del numero de pasajes recuperados en el proceso de RI

Para ver si el número de pasajes recuperado influía significativamente en los resultados se ha repetido la consulta para recuperar 5000 y 10000 pasajes. En el sistema de recuperación de información (RI) se utilizo el campo "`question`", y en el sistema de respuesta a preguntas (QA) se utilizó el modelo "`ktrapeznikov/biobert_v1.1_pubmed_squad_v2`".

Se observa que la mejora no es significativa y el cálculo computacional es muy grande. Los resultados obtenidos para las métricas al recuperar 1000, 5000 y 10000 pasajes fueron los siguientes:

Los datos obtenidos para la métrica **NDNS-Partial** nos muestran que los valores más relevantes se obtienen cuando $k_1 = 8$, aunque estos valores mejoran poco al aumentar el número de pasajes recuperados. Aun así, se obtiene un resultado mejor cuando se utiliza solo el módulo de RI para recuperar 1000 pasajes.

| NDNS-Partial | | | | |
|---|---|---|---|---|
| $k_1$ | 1000 pasajes | 5000 pasajes | 10000 pasajes | 1000 IR passage baseline |
| 0 | 0,1034897491 | 0,06152057429 | 0,04831546802 | 0,2337 |
| 1 | 0,1189019292 | 0,07719582795 | 0,06075599515 | 0,2337 |
| 2 | 0,1299775791 | 0,09084088132 | 0,07824442416 | 0,2337 |
| 3 | 0,1483932914 | 0,1090881989 | 0,09830956546 | 0,2337 |
| 4 | 0,1654971991 | 0,1362822297 | 0,1266431418 | 0,2337 |
| 5 | 0,1841015485 | 0,1598509128 | 0,1527373682 | 0,2337 |
| 6 | 0,2078393839 | 0,1882877016 | 0,1850491972 | 0,2337 |
| 7 | 0,2248794752 | 0,214994804 | 0,2144174885 | 0,2337 |
| 8 | 0,2253357973 | 0,2277198734 | 0,2279825259 | 0,2337 |
| 9 | 0,220155462 | 0,2229339368 | 0,224053889 | 0,2337 |
| 10 | 0,1560223492 | 0,1559298031 | 0,1559298031 | 0,2337 |

**Tabla 5.12 – NDNS-Partial para diferentes cantidades de pasajes recuperados y los distintos valores de $k_1$**

De los datos de la tabla anterior en obtenemos la siguiente gráfica:

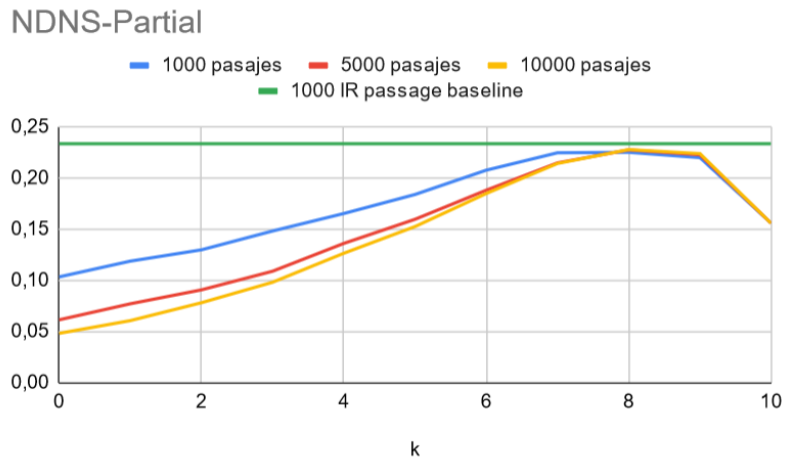



**Figura 5.13 – Gráfica de la métrica NDNS-Partial para diferentes cantidades de pasajes recuperados y distintos valores de $k_1$**

Los datos obtenidos para la métrica **NDNS-Relaxed** nos muestran que los valores más relevantes se obtienen cuando $k_1 = 8$, y que estos valores mejoran poco al aumentar el número de pasajes recuperados. En este caso, el resultado mejor se obtiene cuando se utiliza el módulo de RI+QA para recuperar 10000 pasajes.

| NDNS-Relaxed | | | | |
|---|---|---|---|---|
| $k_1$ | 1000 pasajes | 5000 pasajes | 10000 pasajes | 1000 IR passage baseline |
| 0 | 0,1041257932 | 0,06173565955 | 0,04849290782 | 0,2171 |
| 1 | 0,1195765318 | 0,07748532321 | 0,06096315928 | 0,2171 |
| 2 | 0,1306863971 | 0,09122748676 | 0,07858106625 | 0,2171 |
| 3 | 0,1493469412 | 0,1095882779 | 0,09877005719 | 0,2171 |
| 4 | 0,166581161 | 0,1371247893 | 0,1274613968 | 0,2171 |

| | | | | |
|---|---|---|---|---|
| 5 | 0,1852759971 | 0,1608601536 | 0,1536823048 | 0,2171 |
| 6 | 0,2091231556 | 0,1894219182 | 0,1861804365 | 0,2171 |
| 7 | 0,2262527877 | 0,2163052659 | 0,2157363653 | 0,2171 |
| 8 | 0,2268035615 | 0,2292090343 | 0,2294738849 | 0,2171 |
| 9 | 0,221562281 | 0,2243651032 | 0,2254868608 | 0,2171 |
| 10 | 0,1567413794 | 0,1566456747 | 0,1566456747 | 0,2171 |

**Tabla 5.13 – NDNS-Relaxed para diferentes cantidades de pasajes recuperados y los distintos valores de $k_1$**

De los datos de la tabla anterior en obtenemos la siguiente gráfica:

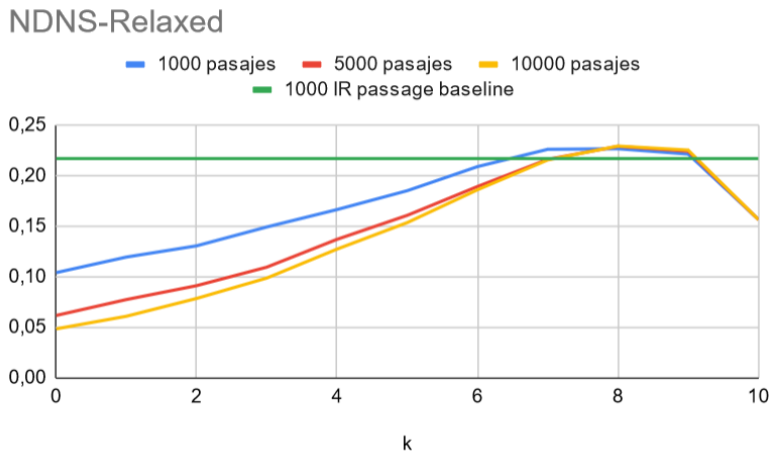



**Figura 5.14 – Gráfica de la métrica NDNS-Relaxed para diferentes cantidades de pasajes recuperados y distintos valores de $k_1$**

Los datos obtenidos para la métrica **NDNS-Exact** nos muestran que los valores más relevantes se obtienen cuando $k_1 = 8$, y que estos valores mejoran un poco al aumentar el número de pasajes recuperados. En este caso, el resultado mejor se obtiene cuando se utiliza el módulo de RI+QA para recuperar 10000 pasajes.

| NDNS-Exact | | | | |
|---|---|---|---|---|
| **$k_1$** | **1000 pasajes** | **5000 pasajes** | **10000 pasajes** | **1000 IR passage baseline** |
| 0 | 0,1175144207 | 0,0689716405 | 0,05411606825 | 0,232333 |
| 1 | 0,1345063887 | 0,08647255081 | 0,0679592183 | 0,232333 |
| 2 | 0,1469644487 | 0,1020976596 | 0,08781900586 | 0,232333 |
| 3 | 0,1682761292 | 0,1229544261 | 0,110751189 | 0,232333 |
| 4 | 0,1878623817 | 0,1550650608 | 0,1439084583 | 0,232333 |
| 5 | 0,2089779367 | 0,1815586428 | 0,1737711749 | 0,232333 |
| 6 | 0,2354244931 | 0,2128351211 | 0,209346239 | 0,232333 |
| 7 | 0,2531675615 | 0,2432684701 | 0,2427197996 | 0,232333 |
| 8 | 0,2537755704 | 0,2566450708 | 0,2569722111 | 0,232333 |
| 9 | 0,2477915318 | 0,2510056583 | 0,2522553635 | 0,232333 |
| 10 | 0,177062277 | 0,1769474419 | 0,1769474419 | 0,232333 |

**Tabla 5.14 – NDNS-Exact para diferentes cantidades de pasajes recuperados y los distintos valores de $k_1$**

De los datos de la tabla anterior en obtenemos la siguiente gráfica:

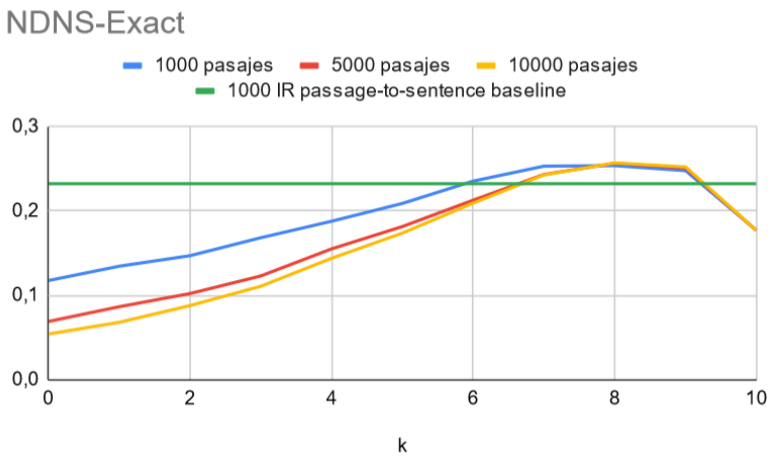



**Figura 5.15 – Gráfica de la métrica NDNS-Exact para diferentes cantidades de pasajes recuperados y distintos valores de $k_1$**

Capítulo 6

# Conclusiones y trabajo futuro

Este capítulo recopila las diferentes conclusiones extraídas del trabajo realizado, y propone algunas líneas de trabajo futuro.

## 6.1. Conclusiones

De los resultados de los experimentos que se han realizado se ha extraído las siguientes conclusiones:

**Pregunta 1** - ¿Es mejor indexar y recuperar oraciones (sentencias) o párrafos (pasajes)?.

Después de evaluar los resultados de las tres métricas (ver Tabla 5.1) se concluye que es mejor recuperar pasajes a sentencias. Posiblemente esto sea así debido a que cuando recuperamos las sentencias se pierde el contexto.

**Pregunta 2** - A la hora de formular las consultas al sistema, ¿es mejor utilizar palabras clave, una narrativa o la pregunta a contestar?

También se ha observado (ver Tabla 5.1) que cuando utilizamos el campo "`question`" de las 45 preguntas, en la recuperación de información, se obtienen mejores resultados que cuando se utilizan los campos "`query`" o "`background`".

**Pregunta 3** - ¿Qué modelo neuronal de extracción de respuestas se adapta mejor a nuestro dominio de aplicación (COVID-19)

Se han analizado los datos (ver Tabla 5.8, Tabla 5.9 y Tabla 5.10), para las distintas métricas y modelos pre-entrenados para la tarea de extracción de respuestas (*question-answering*), y el modelo `ktrapeznikov/biobert_v1.1_pubmed_squad_v2` es el que mostró mejores resultados de todos los que se probaron, para todas las métricas y valor de $k_1 = 8$.

**Pregunta 4** - ¿Cuál es la mejor combinación de evidencias entre el módulo de recuperación y el de extracción de respuestas para realizar un ranking de respuestas?

Analizando los datos (ver Tabla 5.8, Tabla 5.9 y Tabla 5.10), se observa que los mejores resultados, para todas las métricas, se obtienen para $k_1 = 8$ y para el modelo `biobert_v1.1_pubmed_squad_v2`.

Si ahora nos fijamos en el resto de modelos, los mejores resultados para las distintas métricas, se obtienen para los siguientes valores de $k_1$:

| **NDNS-Exact y NDNS-Relaxed** | |
|---|---|
| $k_1$ | **Modelo** |
| 7 | `bert-large-uncased-whole-word-masking-squad2` |
| 8 | `biobert_v1.1_pubmed_squad_v2, distilbert-base-uncased-distilled-squad, scibert_scivocab_uncased_squad_v2` |
| 9 | `roberta-base-squad2 / distilbert-base-cased-distilled-squad` |
| **NDNS-Partial** | |
| $k_1$ | **Modelo** |
| 8 | `biobert_v1.1_pubmed_squad_v2, distilbert-base-uncased-distilled-squad, scibert_scivocab_uncased_squad_v2, bert-large-uncased-whole-word-masking-squad2` |
| 9 | `roberta-base-squad2 / distilbert-base-cased-distilled-squad` |

**Tabla 6.1 - Valores de $k_1$ y modelos que hacen máximo los valores de las distintas métricas.**

En la tabla anterior, vemos que el valor de $k_1$ varia entre 7 y 9, y esto nos indica que el modelo RI tiene mayor peso que el modelo QA. Y la mayor puntuación se obtiene cuando usamos el modelo `biobert_v1.1_pubmed_squad_v2` y las sentencias se puntúan usando la ecuación:

$$score_{NEW} = k_1 * score_{RI} + k_2 * score_{QA} \qquad \text{con } k_1 = 8 \text{ y } k_2 = 2$$

**Pregunta 5** - Dado que hay respuestas que provienen del mismo documento, ¿es adecuado fusionarlas en una única respuesta? ¿En qué escenarios?

Para las tres métricas tenemos:

- **NDNS-Relaxed**: Las respuestas deberían contener sentencias con nuevos *nuggets*.
- **NDNS-Partial**: Las respuestas no deberían contener sentencias sin *nuggets*.
- **NDNS-Exact**: Las respuestas deberían ser cortas (usar las menos sentencias posibles).

Y de los datos obtenidos (ver Tabla 5.11), donde se compara el modelo RI+QA con el modelo RI+QA+merge, se observó que con el fusionado (merge) de sentencias mejoraban las métricas NDNS-Partial y NDNS-Relaxed, y empeoraba la métrica NDNS-Exact. Esto coincide con lo esperado.

**Pregunta 6** - ¿Qué efecto tiene considerar más o menos pasajes en la recuperación como paso previo a la extracción de respuestas?

Del análisis de los datos (ver Tabla 5.12, Tabla 5.13 y Tabla 5.14) se observó que los resultados mejoran al aumentar el número de pasajes recuperados por el sistema RI+QA, y que estos resultados se obtienen para $k_1$ =8 (con independencia de la métrica utilizada). La mejora entre recuperar 1000, 5000 y 10000 es pequeña, sin embargo la diferencia en el tiempo computacional es muy alto. Solo cuando recuperamos 1000 pasajes con el sistema RI (baseline) los resultados de la métrica NDNS-Partial son mejores con independencia del número de pasajes recuperados por el sistema RI+QA.

## 6.2. Trabajo futuro

No se prevé realizar ningún trabajo futuro para mejorar el sistema, pero sería interesante la evaluación de otros sistemas no neuronales para la recuperación de información (RI) como BM25, ColBERT, DeepCT y otros.

También, se podrían tener en cuenta la aparición de nuevos modelos pre-entrenados en la tarea de respuesta a preguntas (*question-answering*) sobre el tema del COVID, como:

- Modelo: `lordtt13/COVID-SciBERT`

o la utilización de modelos que se podrían entrenar con los nuevos datasets que han aparecido sobre COVID, como:

- Dataset: `covid_qa_deepset` (COVID-QA es un dataset para QA que consta de 2,019 pares de preguntas/respuestas anotados por expertos biomédicos voluntarios en artículos científicos relacionados con COVID-19. Un total de 147 artículos científicos del conjunto de datos CORD-19 fueron anotados por 15 expertos.)

# Bibliografía


Abhishek Andhavarapu. 2017. Learning Elasticsearch: Structured and unstructured data using distributed real-time search and analytics. Packt Publishing.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. 2017. Attention Is All You Need. arXiv preprint arXiv:1907.11692.

Betty van Aken, Benjamin Winter, Alexander Löser, and Felix A. Gers. 2019. How does bert answer questions?. Proceedings of the 28th ACM International Conference on Information and Knowledge Management.

Bharvi Dixit, Rafał Kuć, Marek Rogoziński, Saurabh Chhajed. 2017. Elasticsearch : A complete guide. Packt Publishing.

Bruce Croft, Donald Metzler, Trevor Strohman. 2009. Search Engines Information Retrieval in Practice. Pearson.

Chen Qu, Liu Yang, Minghui Qiu, W. Bruce Croft, Yongfeng Zhang, Mohit Iyyer. 2019. BERT with History Answer Embedding for Conversational Question Answering. arXiv preprint arXiv:1905.05412v2.

Christopher D. Manning. Prabhakar Raghavan, Hinrich Schutze. 2008. Introduction to Information Retrieval. Cambridge University Press.

David A. Grossman, Ophir Frieder. 2004. Information Retrieval: Algorithms and Heuristics, 2nd Edition. Springer.

Deepak Ravichandran and Eduard Hovy . 2002. Learning Surface Text Patterns for a Question Answering System. Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, July 2002, pp. 41-47.

Dongfang Xu, Manoj Gopale, Jiacheng Zhang, Kris Brown, Edmon Begoli, and Steven Bethard. 2020. Unified Medical Language System resources improve sieve-based generation and Bidirectional Encoder Representations from Transformers (BERT)–based ranking for concept normalization. Journal of the American Medical Informatics Association, Volume 27, Issue 10, October 2020, Pages 1510–1519.

Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wen-tau Yih, Yejin Choi, Percy Liang, Luke Zettlemoyer. 2018. QuAC : Question Answering in Context. arXiv preprint arXiv:1808.07036.

Gerald Kowalski. 2011. Information Retrieval Architecture and Algorithms. Springer.

Hillary Ngai, Yoona Park, John Chen, Mahboobeh Parsapoor (Mah Parsa). 2021. Transformer-Based Models for Question Answering on COVID19. arXiv preprint arXiv:2101.11432.

Iz Beltagy, Kyle Lo, Arman Cohan. 2019. SciBERT: A Pretrained Language Model for Scientific Text. arXiv preprint arXiv:1903.10676.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, Jaewoo Kang. 2019. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. Bioinformatics, Volume 36, Issue 4, 15 February 2020, Pages 1234–1240.

Peilin Yang, Hui Fang, Jimmy Lin. 2018. Anserini: Reproducible Ranking Baselines Using Lucene. Journal of Data and Information QualityVolume 10Issue 4November 2018 Article No.: 16pp 1–20.

Ricardo Baeza-Yates, Berthier Ribeiro-Neto. 2010. Modern Information Retrieval: The Concepts and Technology Behind Search. Addison Wesley.

Richard J Cooper and Stefan M Ruger. 2000. A Simple Question Answering System. TREC, 2000 - Citeseer.

Victor Sanh, Lysandre Debut, Julien Chaumond, Thomas Wolf. 2020. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv preprint arXiv:1907.11692.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. arXiv preprint arXiv:1909.11942.