Recuperación de información

Cálamo Educación S.L.
María Teresa Tijeras Pascual
Cálamo Educación S.L.
María Teresa Tijeras Pascual

Indice

Recuperación de información

El objetivo de esta unidad es presentar los conceptos, fundamentos y técnicas más importantes de la tarea de recuperación de información, como parte de las tareas de procesamiento del lenguaje natural dedicadas a la extracción de información.

0. Objetivos de la unidad

El objetivo de esta unidad es presentar los conceptos, fundamentos y técnicas más importantes de la tarea de recuperación de información, como parte de las tareas de procesamiento del lenguaje natural dedicadas a la extracción de información. La recuperación de información es la base de la sociedad actual de las Tecnologías de Información y Comunicaciones (TIC), y ayuda a gestionar toda la información digital ya que su objetivo es organizar grandes colecciones de textos con el fin de devolver la lista de documentos más relevantes respecto a una consulta realizada por el usuario.

El **apartado 1** presenta los conceptos más importantes en recuperación de información.

El **apartado 2** se centra en presentar los modelos de recuperación de información existentes, analizando sus fundamentos y describiendo sus principales características y ventajas.

En el **apartado 3** se analiza el concepto de similitud semántica y su relación con el escenario de búsqueda semántica.

En el **apartado 4** se describen las métricas de evaluación que se utilizan típicamente en la tarea de recuperación de información.

Por último, el **apartado 5** presenta las tareas habituales del lingüista en este tipo de proyectos.

1. Fundamentos y conceptos

La recuperación de información (en inglés, *information retrieval*, IR) es un campo de la informática cuyo objetivo es la representación, almacenamiento y acceso a la información. En otras palabras, la recuperación de información se ocupa de la organización y recuperación de información en grandes bases de datos.

En general, estos sistemas se utilizan para ayudar a gestionar grandes volúmenes de información: gestión documental, bibliotecas digitales, archivos de documentos, motores de búsqueda en Internet, etc.

Recuperación de información
Fuente de la imagen: TREC Logo

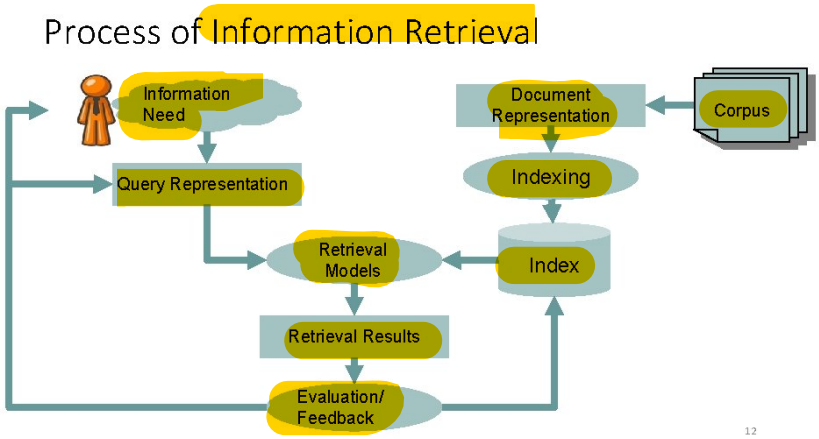
El campo de la recuperación de información siempre ha estado estrechamente relacionado con el del procesamiento del lenguaje natural. Nació en los años cincuenta, periodo en el que el desarrollo de la informática y de la literatura científica condujeron a la elaboración de nuevos enfoques para la caracterización de documentos. Fue en esta etapa cuando comenzó a considerarse el campo de trabajo de la recuperación de información como parte de un área mayor que incluía también el procesamiento del lenguaje natural.

El colapso sufrido en el campo de la traducción automática en los años setenta puso en entredicho la validez y la viabilidad de las teorías de la lingüística computacional y los trabajos de investigación en recuperación de información se alejaron del procesamiento del lenguaje natural, abrazando la orientación hacia métodos puramente estadísticos (independientes de la lingüística).

Ha sido hace pocos años cuando se han restablecido las relaciones entre ambos campos. Como cabía esperar, los métodos puramente estadísticos no alcanzaban los resultados deseados y esto condujo a la introducción de técnicas de procesamiento del lenguaje natural como componentes de los sistemas de recuperación de información. Muchos de los sistemas actuales hacen uso ya de herramientas propias del procesamiento del lenguaje natural como diccionarios y bases léxicas, segmentadores de textos, analizadores sintácticos y semánticos, reconocimiento de entidades, etc.

Formalmente, la recuperación de información es el proceso por el que se representa y almacena una gran colección de datos con vistas a poder utilizarla para encontrar información relevante como respuesta a las necesidades de información de un usuario, expresadas como consultas al sistema.

Este proceso incluye varias etapas básicas: la representación del contenido de los documentos, la representación de la necesidad de información del usuario (consulta), y la comparación entre las dos representaciones para encontrar la información más pertinente respecto a la consulta y devolverla al usuario. Estos procesos se muestran en la figura siguiente.



Proceso de recuperación de información

El proceso de representación de los documentos (textos) se denomina **indexación**. Consiste en procesar cada uno de los documentos mediante cualquier técnica que sea capaz de extraer su contenido semántico y permita su representación eficiente con el fin de poder compararla rápidamente, incluso cuando haya un gran volumen de información.

La indexación es un proceso *off-line* (sin conexión), es decir, que se realiza una única vez, de forma no interactiva y sin intervención por parte del usuario.

El objetivo de la indexación es conseguir una representación lo más breve posible, con poca redundancia, relacionada semánticamente con el documento y manteniendo la especificidad (es decir, manteniendo las características propias que permiten distinguir un documento de otro).

La opción trivial, pero ampliamente utilizada para representar textos, consiste en la asignación por expertos de los términos de indexación a partir de un conjunto finito de ellos, la llamada indexación manual por vocabulario controlado. En contraposición, se tiene la indexación automática con vocabulario controlado, en la que los términos se escogen automáticamente de entre los existentes en un conjunto finito. Otro método es la indexación libre, en la que son los propios autores de los textos los que asignan los términos de indexación.

Por último, hoy en día contamos con la **indexación a través del lenguaje natural**, en la que los términos de indexación los elige el ordenador a partir de los textos. Este es el centro de atención en las investigaciones actuales en recuperación de información. El método más empleado en este tipo de indexación es tomar como índices las palabras más importantes del texto.

La dificultad de este enfoque es que, si indexamos un texto basándonos en palabras del lenguaje natural, podemos obtener decenas de miles de características, un número excesivo, y, además, con una redundancia considerable, en el sentido de que habrá sinónimos o casi sinónimos (si dos palabras son sinónimas podríamos pensar que deberían estar en los mismos documentos, y añadirlas como características).

Otro efecto perjudicial es la ambigüedad en el significado (palabras o expresiones lingüísticas con varios significados). Esto se puede ver como la disyunción entre dos o más conceptos inconexos o no relacionados. Se debe evitar la ambigüedad para las representaciones de los textos, puesto que es de suponer que es poco probable que todos los significados sean de interés en un texto determinado.

Hoy en día, se puede intuir que, de nuevo, como en otras tareas de procesamiento del lenguaje natural, típicamente se utiliza una representación del texto como vector (con *bag-of-words* o *embeddings*, estudiados en anteriores unidades). Los términos correspondientes a cada dimensión del vector del documento se extraen automáticamente a partir de las palabras presentes en el texto mediante técnicas de procesamiento del lenguaje natural (excluyendo en el vector las palabras sin contenido semántico *(stopwords)*, formas derivadas, formas que coinciden normalizando su capitalización, etc.).

El proceso de **búsqueda** consiste primero en generar la representación de la consulta del usuario (que expresa sus necesidades de búsqueda), con la misma estrategia que se utilizó para obtener la representación de los documentos. A continuación, se compara esta representación de la consulta con todas y cada una de las representaciones de cada documento, calculando una puntuación de relevancia (score). Finalmente, se devuelven los resultados al usuario ordenados por relevancia decreciente.

2. Modelos de recuperación de información

Un modelo de recuperación de información queda definido cuando se fijan todos los detalles de cómo se obtienen las representaciones de los documentos y la consulta, la estrategia para evaluar la relevancia de un documento respecto a una consulta y los métodos para establecer la importancia (orden) de los documentos de salida.

Existen diferentes modelos; los más importantes son el modelo booleano, el modelo de espacio de vectores y el modelo probabilístico.

Los sistemas de búsqueda de patrones de texto fueron los primeros que aparecieron, y, más que sistemas, sería más propio llamarlos herramientas. Las consultas se realizan a través de cadenas de texto o expresiones regulares. Hoy en día se emplean para buscar en colecciones pequeñas, tales como archivos en un ordenador personal. Ejemplos característicos son la familia de herramientas “grep” en el entorno UNIX, o la herramienta de búsqueda incluida en entornos Windows.

Los sistemas de búsquedas booleanas sucedieron a los anteriores y permiten buscar en colecciones considerablemente más grandes. En estos sistemas los documentos están representados por un conjunto de términos de indexación y las consultas se componen de términos unidos por operadores lógicos booleanos (AND, OR, NOT). Dada una consulta, se obtienen todos los documentos que cumplen la expresión dada.

Para mejorar el rendimiento de recuperación, se incorpora algún tipo de información sobre la distribución estadística de los términos, por ejemplo, la frecuencia con que estos aparecen en un documento dado, en la colección de documentos o en un subconjunto de documentos considerados relevantes para la consulta. Con esto surgieron nuevos modelos, como los sistemas estadísticos (por ejemplo, el modelo de espacio de vectores) y los modelos probabilísticos.

2.1. Modelo booleano

El modelo booleano es el modelo de recuperación de información más sencillo y fue el empleado en los primeros sistemas.

En el modelo booleano se representa cada documento por la lista de términos (tokens) que contiene, eliminando *stopwords*, símbolos y caracteres especiales, y normalizando la capitalización (por ejemplo, pasando todo a minúsculas). Así, un documento sería:

documento = {término1, término2, término3… términoN}

La consulta de usuario consiste en una expresión escrita que incluye los términos que se quieren encontrar combinados con los operadores de la lógica booleana AND, OR y NOT.

quiero documentos con las palabras … y … o … pero no …

Por ejemplo, la consulta siguiente expresa que el usuario desea obtener documentos que contengan el términoA y además el términoB o el términoC.

términoA AND (términoB OR términoC)

El sistema buscaría entre todos los documentos aquellos que cumplan esa expresión booleana y los devolvería como resultado.

El significado de los operadores es bien conocido:

"t1 AND t2"

Dos términos unidos por el operador AND expresa que se quieren encontrar los documentos en los que aparezcan simultáneamente los términos t1 y t2.

"t1 OR t2"

Dos términos unidos por el operador OR expresa que se quieren encontrar los documentos en los que aparezcan o solamente el término t1 o solamente el término t2 o ambos términos simultáneamente.

"NOT t1"

Un término precedido por el operador unario NOT expresa que se quieren encontrar los documentos en los que no aparezca el término t1.

Un sistema con este modelo sustituye cada término presente en la consulta por el conjunto de documentos en los que aparece dicho término y luego realiza las operaciones de conjuntos correspondientes a cada operador:

"t1 AND t2"

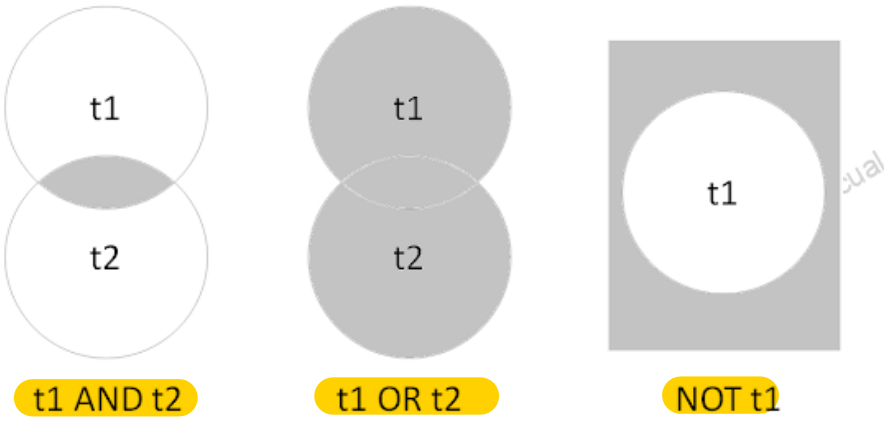
Calcula la intersección de los conjuntos de documentos correspondientes a t1 y t2.

"t1 OR t2"

Calcula la unión de los conjuntos de documentos correspondientes a t1 y t2.

“NOT t1”
Calcula la diferencia entre el conjunto de todos los documentos de la colección y el conjunto de documentos correspondiente a t1.

La figura siguiente ilustra el conjunto de resultados:



Operaciones de conjuntos

La eficacia de recuperación de este modelo es muy baja a no ser que se empleen consultas muy elaboradas y complicadas. Además, a veces los resultados pueden parecer poco intuitivos (por ejemplo, una consulta con cuatro claves AND, no encontraría documentos que tuvieran tres de ellas). Así, parece necesario relajar el significado de estos operadores.

Otro problema es que en el proceso de la indexación hay que decidir si un término es característico del documento que se está procesando, pero este modelo tan sencillo no permite recoger la incertidumbre de esta decisión. Es decir, no es capaz de ordenar la lista de documentos devueltos, ya que un documento o bien cumple la consulta (es *relevante*) o bien no la cumple (es *irrelevante*), pero no existe ningún cálculo de la relevancia del documento respecto a la consulta.

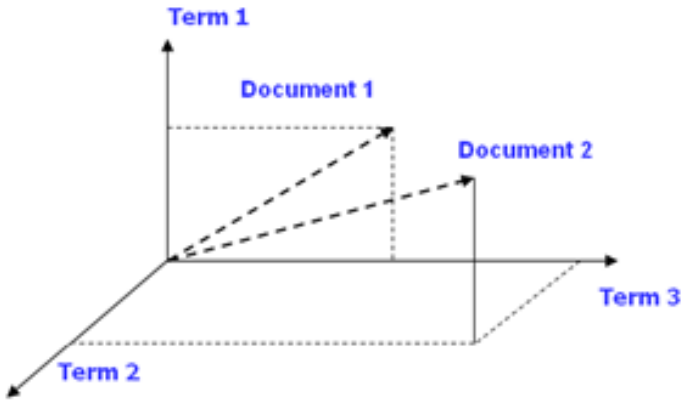
2.2. Modelo de espacio vectorial

El modelo de espacio vectorial (o modelo de espacio de vectores, en inglés, *vector space model*) fue definido por Gerard Salton en los años setenta y es el modelo habitual de representación de textos. Consiste en que cada documento de la colección está representado por un vector n-dimensional (n es el número de términos de indexación elegidos para la colección) en el que cada componente representa el peso del término asociado a esa dimensión en el documento (es decir, la importancia de dicha palabra en el texto).

$$\mathbf{v}_d = [w_{1,d}, w_{2,d}, \ldots, w_{N,d}]^T$$

Representación de cada texto como vector

Las consultas de usuario se representan de igual forma.



Representación de los documentos en el espacio de vectores

Para calcular el peso de cada palabra en un documento, existen diferentes técnicas. La técnica más sencilla es utilizar la frecuencia del término (TF, *term frequency*), que es el número de veces que aparece el término en el documento.

Cuantas más veces aparezca una palabra en el texto, más importante será en dicho texto. Es importante excluir del análisis las palabras sin significado *(stopwords)*, en idiomas flexivos considerar una única forma lematizada y no las diferentes variantes de una palabra ("cantar" en vez de "cantamos", "cantaba"…), convertir a una misma capitalización ("PEDRO" es igual que "Pedro" para la recuperación), así como normalizar de alguna manera los valores de frecuencia para compensar la longitud del documento (en documentos más largos, es previsible que aparezca más veces cada término).

Además, puede resultar útil añadir peso adicional a los contenidos que aparezcan en según qué partes de la estructura del documento, por ejemplo, asignar pesos mayores para los términos que aparecen en el título o en el resumen frente a los términos que aparecen solo en el cuerpo del texto.

Otra técnica habitual para calcular el peso de cada término es utilizar la fórmula TF·IDF, donde TF es la frecuencia del documento multiplicada por IDF, que es la frecuencia inversa de documento *(inverse document frecuency)*, una medida que considera el número de documentos diferentes en los que aparece el término. Así, si un término aparece en muchos documentos, dicho término será menos importante para diferenciar entre ellos, es decir, su valor de IDF y, por tanto, su peso en el vector serán menores.

$$w_i = tf_i * log\left(\frac{D}{df_i}\right)$$

Esquema de peso basado en TF-IDF

Por último, como ya se estudió en anteriores unidades, para representar cada documento se emplean también los vectores de *embeddings*, independientes o no del contexto, calculados con los diferentes algoritmos o técnicas existentes.

Para calcular la similitud entre un documento y la consulta, se emplea una operación que compara sus vectores. Se podría utilizar la distancia euclídea entre vectores (0 es la menor distancia, luego la mayor relevancia), pero típicamente se emplea la llamada fórmula del coseno, o producto escalar, entre ambos vectores (vector de cada documento y vector de la consulta), que va entre 0 y 1, donde 1 es la mayor similitud, luego la mayor relevancia.

$$Sim(Q, D_i) = \frac{\sum_i w_{Q,j} w_{i,j}}{\sqrt{\sum_j w_{Q,j}^2} \sqrt{\sum_i w_{i,j}^2}}$$

Similitud con la fórmula del coseno

Los documentos se ordenan según el valor de esta similitud.

La figura siguiente muestra un ejemplo con tres documentos. El documento D2 es finalmente el que presenta más similitud con la consulta, luego es el más relevante.

Query, Q: "gold silver truck"
D_1: "Shipment of gold damaged in a fire"
D_2: "Delivery of silver arrived in a silver truck"
D_3: "Shipment of gold arrived in a truck"
D = 3; IDF = log(D/df_i)

	Counts, tf_i							Weights, $w_i = tf_i * IDF_i$			
Terms	Q	D_1	D_2	D_3	df_i	D/df_i	IDF_i	Q	D_1	D_2	D_3
a	0	1	1	1	3	3/3 = 1	0	0	0	0	0
arrived	0	0	1	1	2	3/2 = 1.5	0.1761	0	0	0.1761	0.1761
damaged	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
delivery	0	0	1	0	1	3/1 = 3	0.4771	0	0	0.4771	0
fire	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
gold	1	1	0	1	2	3/2 = 1.5	0.1761	0.1761	0.1761	0	0.1761
in	0	1	1	1	3	3/3 = 1	0	0	0	0	0
of	0	1	1	1	3	3/3 = 1	0	0	0	0	0
silver	1	0	2	0	1	3/1 = 3	0.4771	0.4771	0	0.9542	0
shipment	0	1	0	1	2	3/2 = 1.5	0.1761	0	0.1761	0	0.1761
truck	1	0	1	1	2	3/2 = 1.5	0.1761	0.1761	0	0.1761	0.1761

Ejemplo de modelo de espacio de vectores (pesos)

$$Sim\,\theta_{D_1} = \frac{0.0310}{0.5382 * 0.7192} = 0.0801$$

$$Sim\,\theta_{D_2} = \frac{0.4862}{0.5382 * 1.0955} = 0.8246$$

$$Sim\,\theta_{D_3} = \frac{0.0620}{0.5382 * 0.3522} = 0.3271$$

Ejemplo de modelo de espacio de vectores (similitudes)

La gran ventaja de este modelo reside en su simplicidad y facilidad de comprensión.

2.3. Modelo probabilístico

El modelo de espacio de vectores anterior asume que los vectores de términos distribuidos en el espacio son ortogonales y no considera las relaciones existentes entre los términos.

Los modelos probabilísticos, desarrollados en los setenta y ochenta en los trabajos de Robertson y Sparck Jones, se basan en la asignación de pesos a los términos de indexación de un documento respecto a la totalidad de términos de indexación de toda la colección. La idea del modelo es, dada una consulta y un documento, estimar la probabilidad de que el usuario encuentre el documento con esa consulta. El modelo analiza la distribución de los documentos dentro de la colección y se basa en la premisa de que los términos que aparecen en documentos relevantes previamente recuperados para una consulta dada deberían tener mayor peso.

Un algoritmo de este tipo es BM25 (BM es el acrónimo de *best matching*), también conocido como Okapi BM25 (Okapi fue el primer sistema donde se empleó), que utiliza *bag-of-words* y TF·IDF.

Hoy en día la mayoría de sistemas de recuperación de información, aunque también soportan el modelo clásico TF·IDF, por defecto se basan en BM25 como, por ejemplo, los sistemas *opensource* Solr, Lucene y el más moderno Elasticsearch.

Para saber más sobre *Okapi BM25*, puedes pinchar en este enlace.

3. Similitud semántica

3.1. Conceptos

En muchos escenarios, la tarea básica consiste en, de una forma u otra, encontrar la similitud o el grado de parecido entre dos textos. Por ejemplo, en recuperación de información, sirve para medir la relevancia de cada texto con respecto a la consulta de usuario, o para encontrar los textos más parecidos entre sí en un conjunto de textos (sistemas de *clustering*).

Para calcular la similitud textual *(textual similarity)* entre dos textos, se puede hacer a nivel más superficial (similitud léxica, *lexical similarity*), considerando solo las palabras que contienen ambos textos, o bien analizando su significado (similitud semántica) que, como resulta evidente, es más complejo, aunque ofrece mejores resultados.

Por ejemplo, consideremos las dos frases siguientes:

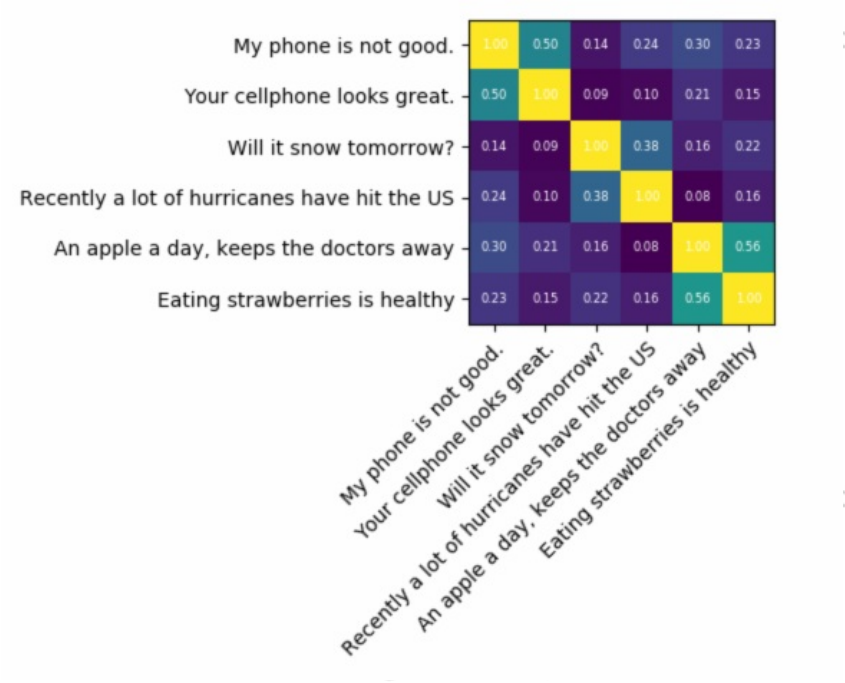
El gato se comió al ratón

El ratón se comió la comida del gato

Su similitud léxica es muy alta, ya que de las 4 palabras que no son *stopwords* (“gato”, “comió”, “ratón” y “comida”), coinciden 3 de ellas (todas salvo “comida”). Sin embargo, su similitud a nivel semántico es muy baja ya que el significado real de las palabras en el contexto de la frase es totalmente distinto.

Para abordar el cálculo de la similitud semántica, la idea es representar los textos como vectores de características y comparar los textos midiendo la distancia entre estas características. Y hay múltiples formas de calcular las características que capturan la semántica de los documentos y múltiples algoritmos para capturar la estructura de dependencia de los documentos para centrarse en su significado.

De todas ellas, como ya se estudió en la unidad 1, la técnica que mejores resultados alcanza hoy en día es la representación del texto como *embeddings*, bien independientes del contexto (modelo Word2Vec o similares) o bien dependientes del contexto (modelos basados en técnicas de *transformers* como BERT). Para medir la distancia entre vectores típicamente se utiliza la distancia euclídea, la fórmula del coseno o la similitud de Jaccard.



Ejemplo de similitud entre frases utilizando distancia euclídea entre vectores de embeddings *con BERT*

3.2. Búsqueda semántica

La búsqueda semántica trata de mejorar la precisión de la búsqueda mediante la comprensión del contenido de la consulta. A diferencia de los motores de búsqueda tradicionales, que solo encuentran documentos basándose en coincidencias léxicas, la búsqueda semántica también encuentra sinónimos.

La idea que subyace a la búsqueda semántica es representar todos los elementos del corpus, ya sean textos completos, párrafos o frases, en un espacio de vectores vectorial, utilizando vectores de *embeddings*. En el momento de la búsqueda, se procesa la consulta de igual manera y se buscan los vectores de *embeddings* más cercanos en el corpus, que son los que tendrán mayor solapamiento semántico.

Hay dos escenarios típicos de búsqueda semántica:

Búsqueda semántica simétrica

En la búsqueda semántica simétrica, la consulta y las entradas del corpus tienen aproximadamente la misma longitud y el mismo contenido.

Por ejemplo, la consulta podría ser "¿Cómo puedo aprender Python en línea?" para encontrar textos como "¿Cómo se puede aprender Python en la web?".

Una aplicación es la búsqueda de textos similares a un texto dado: identificación de textos duplicados, sistemas de *clustering* de textos, etc.

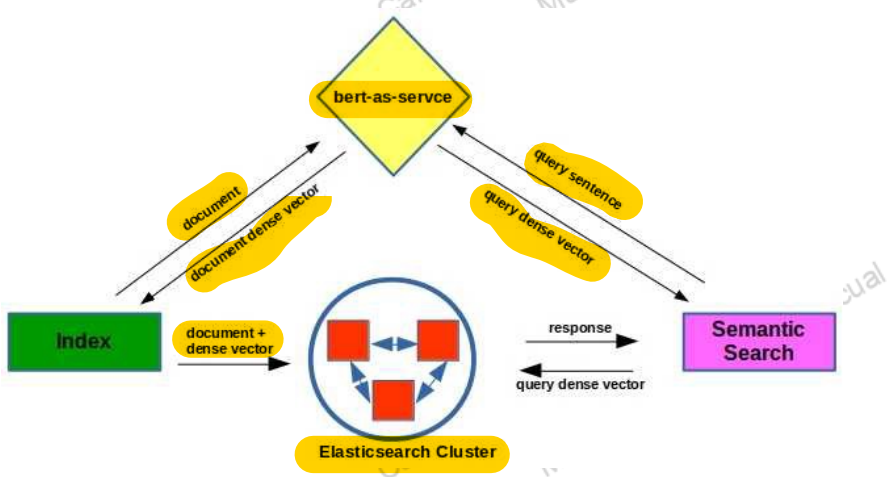
Búsqueda semántica asimétrica

En el caso de la búsqueda semántica asimétrica, normalmente se tiene una consulta corta (como una pregunta o algunas palabras clave) y se quiere encontrar un texto más largo que responda a la consulta. Es el caso habitual de los sistemas de recuperación de información.

Un ejemplo sería la consulta "¿Qué es Python?" para encontrar el texto "Python es un lenguaje de programación interpretado, de alto nivel y de propósito general".

Los sistemas de recuperación de información modernos, como Apache Solr y Elasticsearch, incluyen funcionalidad de búsqueda basada en vectores de *embeddings* (Elasticsearch de forma nativa, Solr con *plugins*). Optimizan el cálculo de los vectores en el momento de la indexación de los documentos, su representación dentro de los índices del sistema (en una estructura que denominan vectores densos -dense vector) y, sobre todo, el cálculo de distancias entre vectores en el momento de la búsqueda.

Sin embargo, estas soluciones solo funcionan suficientemente rápido con colecciones de documentos no muy grandes. Para optimizar, se utilizan servicios externos dedicados al cálculo de vectores de *embeddings* (como bert-as-service).

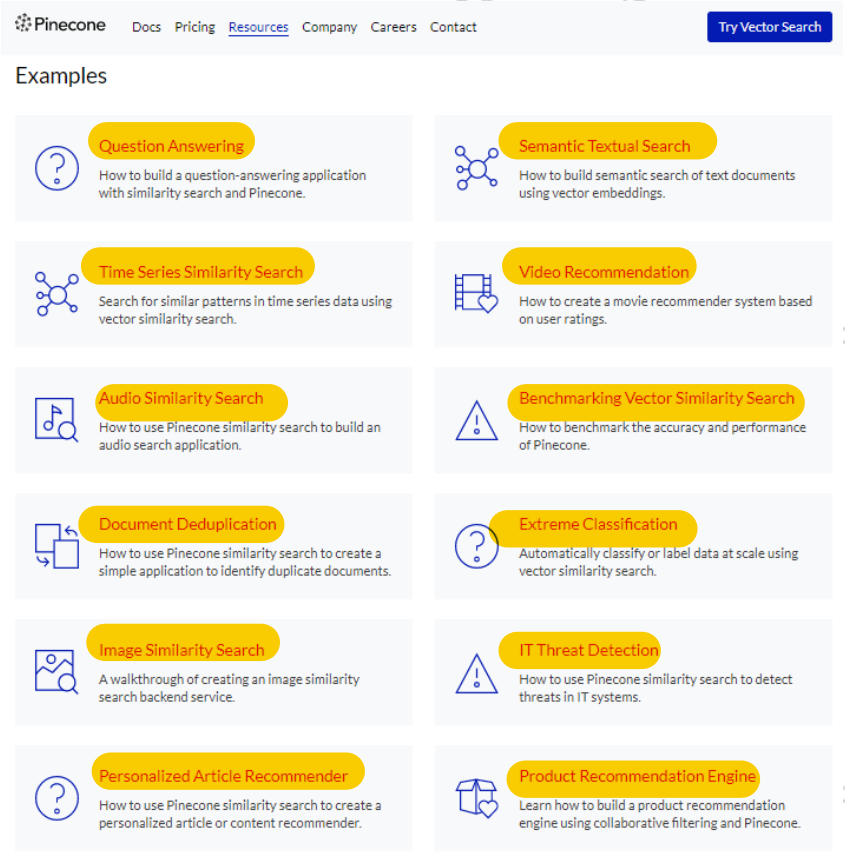


Arquitectura Elasticsearch con bert-as-service
Fuente de la imagen: Towardsdatascience.com

Otra alternativa consiste en utilizar bases de datos de vectores diseñadas y dedicadas específicamente para la indexación y búsqueda de información vectorial. En la actualidad, la mejor opción es Milvus, de código abierto, muy fácil de usar, rápido y que, según afirman, es muy fiable, escalable y robusto, y admite la búsqueda de vectores casi en tiempo real de miles de millones de vectores.

Una alternativa comercial es Pinecone, que ofrece la búsqueda por similitud como un servicio en el que se pueden indexar miles de millones de vectores en tiempo real y buscar las coincidencias más cercanas con una latencia de milisegundos.

La figura siguiente (sobre recursos de Pinecone) sirve como resumen de las aplicaciones más populares: respuesta a preguntas; búsqueda semántica de texto/audio/vídeos/imágenes; recomendación de vídeos, artículos o productos; deduplicación de documentos (búsqueda de duplicados); etc.



Aplicaciones de Pinecone

Como técnica para mejorar los resultados de la búsqueda, se puede aplicar una técnica denominada **expansión de consultas** (QE, *Query Expansion*), que consiste en añadir términos adicionales que orienten el proceso de búsqueda.

Los términos se pueden añadir mediante retroalimentación (utilizando términos que aparecen en documentos que se hayan recuperado previamente) o con el uso de tesauros y diccionarios (incluyendo lista de sinónimos o términos relacionados).

Para saber más sobre la *Query Expansion*, puedes pinchar en este enlace.

4. Métricas de evaluación

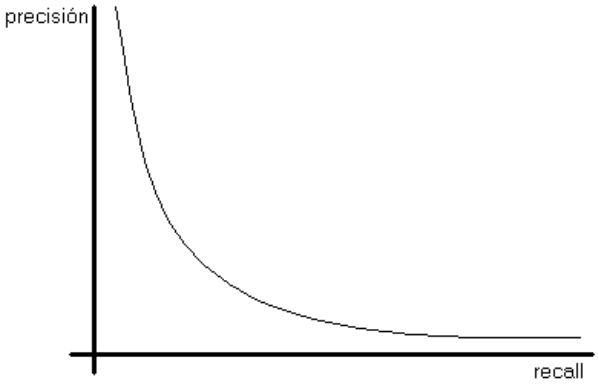
Las métricas de evaluación de los sistemas de recuperación de información son las métricas estándar de las tareas de procesamiento del lenguaje natural (de hecho, estas métricas han surgido del campo de la recuperación de información).

Así, las medidas básicas del **rendimiento de recuperación** son precisión y cobertura *(recall)*. La mayoría de experimentos se centran en analizar, mediante juicios de carácter subjetivo, la relevancia de los documentos (diferentes jueces asignan valores de relevancia a documentos recuperados en función de una consulta dada).

precision = (n.º de documentos recuperados relevantes) / (n.º de documentos recuperados)

recall = (n.º de documentos recuperados relevantes) / (n.º de documentos relevantes en toda la colección)

Ambos parámetros toman valores entre 0 y 1 y, como en muchas ocasiones interesa comparar el rendimiento de los sistemas según ambos parámetros, se han desarrollado métodos de evaluación simultáneos. Uno de ellos involucra el uso de gráficas *precision-recall*, con una relación típica como la mostrada en la figura.



Gráfica precisión-cobertura

Idealmente, se desearía conseguir un alto *recall* y una gran precisión. En realidad, se debe establecer un compromiso. Esta gráfica muestra que están inversamente relacionados. Cuando se trabaja para que la precisión de un sistema de recuperación de información aumente, suele conseguirse a costa de una disminución del *recall*, y viceversa. Esto es porque unos términos de indexación muy específicos producen, lógicamente, una mayor precisión, pero a costa de la cobertura; y al revés: unos términos muy generales producirían un *recall* elevado, pero con una baja precisión.

Además, existe la medida-F, la medida combinada entre precisión y cobertura.

Una consideración adicional en los sistemas de recuperación de información es calcular la precisión o cobertura únicamente para los N primeros documentos recuperados, lo que, para el caso de la precisión (y similar para el *recall*), se denomina "precisión en N" (o "precision@N"), donde N va de 1 a un número máximo de documentos (típicamente, entre 20 y 25 resultados para la primera página de un buscador). Por ejemplo, precision@1=100% significa que el primer documento es relevante; y precision@5=80% significa que, de los 5 primeros documentos, 4 son relevantes (4/5=80%).

Adicionalmente, los sistemas de recuperación de información se pueden analizar también bajo otras consideraciones.

En el **rendimiento de ejecución** se mide el tiempo que emplea el sistema, o algún módulo componente, en realizar una operación. Este parámetro ha sido siempre de gran importancia en los sistemas de recuperación de información, puesto que la mayoría de ellos son interactivos y un tiempo de recuperación largo reduciría la utilidad del sistema. Las necesidades no funcionales de cada sistema en particular son las que especifican los tiempos máximos aceptables para las actividades de mantenimiento de la colección (documentos e índices) y las actividades de consulta.

En el **rendimiento de almacenamiento** se mide el incremento del espacio ocupado por los datos más los índices, respecto al espacio ocupado sólo por los datos. Por ejemplo, si los datos ocupan 100 KB y los índices son 50 KB, el rendimiento de almacenamiento es de (100+50)/100 = 150%.

Otras formas de evaluar comparativamente los diferentes sistemas son:

Evaluación orientada al usuario

La información necesaria para evaluar el sistema se obtiene directamente de los usuarios, analizando factores como su comportamiento (por ejemplo, el tiempo que pasan mirando el documento), la frecuencia de uso del sistema o su grado de satisfacción, medido por una encuesta.

Evaluación orientada a la tarea

Este método consiste en plantear a los usuarios una serie de preguntas difíciles que deben responder empleando la información que les proporciona el sistema. Permite comparar la eficiencia de distintos sistemas, haciéndolos trabajar sobre la misma base documental, y el rendimiento puede medirse a través del número de respuestas correctamente contestadas.

5. Tarea del lingüista

En este tipo de sistemas el papel del lingüista consiste típicamente en:

1

La generación de conjuntos de evaluación del sistema, en este caso, listas de consultas con los documentos que se espera que el sistema devuelva. Como este trabajo podría ser virtualmente infinito, hay que determinar el ámbito de la evaluación, definir una serie de casos de prueba (por ejemplo: consultas con una sola palabra no ambigua, con una palabra ambigua, con varias palabras y el operador AND, el operador OR, etc.) y encontrar los N documentos más relevantes.

2

Colaborar en la definición de los valores de los parámetros del sistema, en función de los objetivos de aplicación. Por ejemplo, las fórmulas que se aplican para calcular la relevancia o los factores correctores de la relevancia que se aplican a términos especiales (por ejemplo, las palabras del título o en negrita).

3

Colaborar en la definición de los procesos de análisis lingüístico del texto que se consideren necesarios (tareas de procesamiento del lenguaje natural) para optimizar la selección de los términos de indexación. Por ejemplo, cómo se aborda la segmentación del texto, el empleo o no de términos lematizados, el interés de usar *tokens* multipalabra (“Reino Unido”) o bien indexar cada palabra por separado, la eliminación de símbolos o caracteres especiales, etc.

4
Llevar a cabo las evaluaciones del sistema a través de las métricas definidas para el proyecto, en concreto: precisión y cobertura, y medida en diferentes puntos (por ejemplo, precision@1, precision@5, precision@10…).

RESUMEN

La recuperación de información se ocupa de la **organización y recuperación de información en grandes colecciones de textos** con el fin de descubrir conocimiento como respuesta a una **solicitud del usuario** (consulta). Un sistema de recuperación de información devuelve la lista de documentos más relevantes respecto a la consulta realizada.

Tiene dos procesos básicos: el **proceso de representación de los textos**, que se denomina indexación y consiste en procesar cada uno de los documentos mediante cualquier técnica que sea capaz de extraer su contenido semántico y permita su representación eficiente para poder compararla rápidamente; y el **proceso de búsqueda**, que consiste en generar la representación de la consulta del usuario y compararla con todas y cada una de las representaciones de cada documento, calculando una puntuación de relevancia *(score)* y devolviendo al usuario como resultado la lista de documentos ordenados por relevancia decreciente.

Existen diferentes modelos de recuperación de información. Los más importantes son el **modelo booleano**, el **modelo de espacio de vectores** y el **modelo probabilístico**.

En el modelo booleano se representa cada documento por la lista de términos que contiene, y la consulta de usuario consiste en una expresión escrita donde se expresan los términos que se quieren encontrar combinándolos con los operadores de la **lógica booleana AND, OR y NOT**.

El modelo de espacio vectorial es el modelo habitual de representación de textos y consiste en que **cada documento se representa por un vector** en el que cada componente representa el peso del término asociado a esa dimensión en el documento. Para calcular el peso de cada palabra en un documento, existen diferentes técnicas como **TF, TF·IDF o *embeddings***. Para calcular la similitud entre un documento y la consulta, se emplea una operación que compara sus vectores, habitualmente con la fórmula del coseno (lo mismo que el producto escalar) entre ambos vectores.

El modelo probabilístico consiste en intentar estimar la probabilidad de que el usuario encuentre un determinado documento utilizando una consulta dada, analizando la **distribución de los documentos** dentro de la colección. Un algoritmo muy conocido es **BM25**, el más utilizado en los sistemas de recuperación de información a día de hoy.

L a **búsqueda semántica** trata de mejorar la precisión de la búsqueda mediante la comprensión del contenido de la consulta. A diferencia de los motores de búsqueda tradicionales, que solo encuentran documentos basándose en las coincidencias léxicas, la búsqueda semántica también **puede encontrar sinónimos**.

La idea que subyace a la búsqueda semántica es representar todos los elementos del corpus en un espacio de vectores vectorial, utilizando vectores de *embeddings*. En el momento de la búsqueda, se procesa la consulta de igual manera y se buscan los vectores de *embeddings* más cercanos en el corpus, que son los que tendrán mayor solapamiento semántico.

Ejercicios

Ejercicio 5: Similitud semántica y búsqueda semántica

El objetivo de este ejercicio es practicar con la similitud semántica y la búsqueda semántica basada en vectores de *embeddings*, programando en código Python en el entorno de Google Colaboratory.

Como siempre, el primer paso es crear un cuaderno de Colab nuevo, activando el entorno de ejecución acelerado con GPU.

1. Búsqueda semántica

En el ejercicio vamos a emplear Sentence Transformers, un paquete Python que proporciona funcionalidad para representación de textos mediante vectores de *embeddings* en más de 100 idiomas y que se emplea para tareas de similitud semántica.

Para saber más sobre *Sentence Transformers*, puedes pinchar en este enlace.

El primer paso es instalar el paquete de Python en el sistema. Copia el siguiente código en una nueva celda de código y ejecútalo pulsando el icono de “Reproducir”.

```
!pip install sentence_transformers
```

Una vez instalado, hay que incluir las importaciones de los paquetes que se van a utilizar. Copia y ejecuta el siguiente código.

```
import sentence_transformers
import numpy as np
```

El siguiente paso es disponer de un corpus de textos de ejemplo. Para este ejercicio, vamos a definir manualmente una lista corta de textos, con el siguiente código Python.

```
corpus = ['A man is eating food.',
          'A man is eating a piece of bread.',
          'The girl is carrying a baby.',
          'A man is riding a horse.',
          'A woman is playing violin.',
          'Two men pushed carts through the woods.',
          'A man is riding a white horse on an enclosed ground.',
          'A monkey is playing drums.',
          'A cheetah is running behind its prey.']
```

Pincha aquí para acceder al código

```
corpus = ['A man is eating food.',
          'A man is eating a piece of bread.',
          'The girl is carrying a baby.',
          'A man is riding a horse.',
          'A woman is playing violin.',
          'Two men pushed carts through the woods.',
          'A man is riding a white horse on an enclosed ground.',
          'A monkey is playing drums.',
          'A cheetah is running behind its prey.']
```


Ahora hay que cargar un modelo preentrenado para el cálculo de vectores de *embeddings*. Sentence Transformers ofrece un gran número de modelos, aquí vamos a utilizar el llamado "paraphrase-multilingual-MiniLM-L12-v2", que ofrece un buen balance entre velocidad de procesamiento y rendimiento para la tarea de similitud semántica.

Para saber más sobre Pretrained Models, puedes pinchar en este enlace.

model = 'paraphrase-multilingual-MiniLM-L12-v2'

embedder = sentence_transformers.SentenceTransformer(model)

A continuación, hay que representar cada texto del corpus con su vector de *embeddings*. Para ello, ejecuta el siguiente código. Si da un error indicando que "sentence_transformers is not defined", ejecuta de nuevo la instalación del paquete en el sistema.

corpus_embeddings = embedder.encode(corpus)

La siguiente función en Python implementa el algoritmo de búsqueda semántica. Primero calcula el vector de *embeddings* de la consulta proporcionada, con el mismo modelo que se empleó para calcular los vectores del corpus. Luego calcula la distancia entre el vector de la consulta con cada uno de los vectores del corpus empleando la similitud del coseno. Finalmente, ordena los resultados por similitud decreciente y muestra los 5 primeros.

```
def query(embedder, corpus, query, top_k=5):
  query_embedding = embedder.encode([query])
  cos_scores = sentence_transformers.util.pytorch_cos_sim(
  query_embedding, corpus_embeddings)[0]
  cos_scores = cos_scores.cpu()
  top_results = np.argpartition(-cos_scores, range(top_k))[0:top_k]
  print('\nQuery:', query)
  for idx in top_results[0:top_k]:
    print(corpus[idx].strip(), "(Score: %.4f)" % (cos_scores[idx]))
```

Pincha aquí para acceder al código

```
def query(embedder, corpus, query, top_k=5):

    query_embedding = embedder.encode([query])

    cos_scores = sentence_transformers.util.pytorch_cos_sim(query_embedding, corpus_embeddings)[0]

    cos_scores = cos_scores.cpu()

    top_results = np.argpartition(-cos_scores, range(top_k))[0:top_k]

    print('\nQuery:', query)

    for idx in top_results[0:top_k]:

        print(corpus[idx].strip(), "(Score: %.4f)" % (cos_scores[idx]))
```

Ahora ya es posible preguntar al sistema, buscando los documentos semánticamente más parecidos a la consulta proporcionada. Por ejemplo, para obtener los documentos más parecidos a "A man is eating pasta":

query(embedder, corpus, 'A man is eating pasta.')

Los resultados de la ejecución son los siguientes:

Query: A man is eating pasta.

A man is eating food. (Score: 0.6734)

A man is eating a piece of bread. (Score: 0.4269)

A man is riding a horse. (Score: 0.2086)

A man is riding a white horse on an enclosed ground. (Score: 0.1020)

A cheetah is running behind its prey. (Score: 0.0566)

Los resultados mostrados muestran una similitud de 0.6734 para el documento donde se está comiendo “comida” en general y 0.4269 cuando se está comiendo un trozo de pan. El resto de casos presentan una similitud es muy baja y se podrían desechar.

Otros ejemplos de consulta podrían ser los siguientes:

query(embedder, corpus, 'Someone in a gorilla costume is playing a set of drums.')

query(embedder, corpus, 'A cheetah chases prey on across a field.')

Con los resultados siguientes:

Query: Someone in a gorilla costume is playing a set of drums.

A monkey is playing drums. (Score: 0.8167)

A cheetah is running behind its prey. (Score: 0.2720)

A woman is playing violin. (Score: 0.1721)

A man is riding a horse. (Score: 0.1291)

A man is riding a white horse on an enclosed ground. (Score: 0.1213)

Query: A cheetah chases prey on across a field.

A cheetah is running behind its prey. (Score: 0.9147)

A monkey is playing drums. (Score: 0.2655)

A man is riding a horse. (Score: 0.1933)

A man is riding a white horse on an enclosed ground. (Score: 0.1733)

A man is eating food. (Score: 0.0329)

Los resultados son increíblemente buenos también.

El modelo empleado es multilingüe, lo que significa que los espacios de vectores están alineados entre diferentes idiomas, es decir, los vectores de *embeddings* de textos que son semánticamente similares en diferentes idiomas están muy cercanos en el espacio vectorial. En concreto, el modelo "paraphrase-multilingual-MiniLM-L12-v2" soporta más de 50 idiomas.

Por ello, es posible utilizar una consulta en español para obtener documentos en inglés, y al revés. Es lo que se conoce como *"cross-lingual information retrieval"*.

query(embedder, corpus, 'Una persona comiendo pasta.')

query(embedder, corpus, 'El músico toca la guitarra.')

query(embedder, corpus, 'El buitre atrapa a su presa.')

Se puede comprobar que los resultados son muy razonables, incluso “mágicos”:

Query: Una persona comiendo pasta.

A man is eating food. (Score: 0.7202)

A man is eating a piece of bread. (Score: 0.6248)

A man is riding a horse. (Score: 0.1179)

A monkey is playing drums. (Score: 0.1001)

A man is riding a white horse on an enclosed ground. (Score: 0.0589)

Query: El músico toca la guitarra.

A woman is playing violin. (Score: 0.2975)

A monkey is playing drums. (Score: 0.2395)

A man is eating a piece of bread. (Score: 0.0664)

A man is riding a white horse on an enclosed ground. (Score: 0.0584)

A cheetah is running behind its prey. (Score: 0.0552)

Query: El buitre atrapa a su presa.

A cheetah is running behind its prey. (Score: 0.5262)

A monkey is playing drums. (Score: 0.2309)

A man is riding a white horse on an enclosed ground. (Score: 0.1388)

A man is eating a piece of bread. (Score: 0.1280)

A man is eating food. (Score: 0.1161)

2. Búsqueda de duplicados

Otra aplicación típica donde se utiliza similitud semántica es la búsqueda de textos duplicados o parecidos.

El código siguiente busca aquellos pares de textos con mayor similitud semántica.

```
paraphrases = sentence_transformers.util.paraphrase_mining(
    embedder, corpus)

for paraphrase in paraphrases[0:10]:
    score, i, j = paraphrase
    print("{}\t{}\t{:.4f}".format(corpus[i], corpus[j], score))
```

```
A man is riding a horse.    A man is riding a white horse on an
enclosed ground. 0.7884
A man is eating food.  A man is eating a piece of bread. 0.7261
A man is eating food.  A man is riding a horse.    0.2784
A monkey is playing drums.  A cheetah is running behind its prey.
     0.2770
A man is riding a white horse on an enclosed ground.    A  cheetah
is running behind its prey.  0.1887
A man is riding a white horse on an enclosed ground.    A  man  is
eating food.    0.1708
A man is eating a piece of bread. A  man  is  riding  a  horse.
     0.1665
A monkey is playing drums.  A woman is playing violin.  0.1628
A man is riding a horse.    A cheetah is running behind its prey.
     0.1519
A man is riding a horse.    Two men pushed carts through the woods.
     0.1178
```

Pincha aquí para acceder al código y su salida

Código:

```
paraphrases = sentence_transformers.util.paraphrase_mining(embedder, corpus)

for paraphrase in paraphrases[0:10]:

  score, i, j = paraphrase

  print("{}\t{}\t{:.4f}".format(corpus[i], corpus[j], score))
```

Cuya salida es:

A man is riding a horse. A man is riding a white horse on an enclosed ground. 0.7884

A man is eating food. A man is eating a piece of bread. 0.7261

A man is eating food. A man is riding a horse. 0.2784

A monkey is playing drums. A cheetah is running behind its prey. 0.2770

A man is riding a white horse on an enclosed ground. A cheetah is running behind its prey. 0.1887

A man is riding a white horse on an enclosed ground. A man is eating food. 0.1708

A man is eating a piece of bread. A man is riding a horse. 0.1665

A monkey is playing drums. A woman is playing violin. 0.1628

A man is riding a horse. A cheetah is running behind its prey. 0.1519

A man is riding a horse. Two men pushed carts through the woods. 0.1178

Se observa que las frases más parecidas entre sí son las siguientes, con una puntuación (score) de 0.7884:

A man is riding a horse.

man is riding a white horse on an enclosed ground.

El segundo par de frases más parecidas semánticamente, con una puntuación de 0.7261, son:

A man is eating food.

A man is eating a piece of bread.

El resto de frases tienen un grado de parecido mucho menor, así que si se considera un umbral de 0.50, no se detectarían como duplicados.

Prueba a insertar la frase siguiente en español en el corpus.

Un mono haciendo ruido con unos tambores.

Añade la frase en el código que hay casi al principio del ejercicio, ejecuta el código y también la celda donde se calculan los vectores de *embeddings* de los documentos del corpus.

Si vuelves a buscar los pares de frases más parecidas semánticamente, ejecutando la celda con el código correspondiente, obtendrás que las frases más parecidas, con un score de 0.9247, son ahora:

Un mono haciendo ruido con unos tambores.

A monkey is playing drums.

Prueba también con otras frases y otros idiomas, por ejemplo, francés, portugués, italiano, e incluso chino o japonés:

Recursos

Enlaces de Interés

- **TREC Logo**
  https://www.nist.gov/image/newlogopng
- **Okapi BM25**
  https://en.wikipedia.org/wiki/Okapi_BM25
- **Semantics at Scale: BERT + Elasticsearch**
  https://towardsdatascience.com/semantics-at-scale-bert-elasticsearch-be5bce877859
- **Milvus**
  https://milvus.io/
- **Pinecone**
  https://www.pinecone.io/
- **Query Expansion**
  https://en.wikipedia.org/wiki/Query_expansion
- **Google Colaboratory**
  https://colab.research.google.com/
- **Sentence Transformers**
  https://www.sbert.net/
- **Pretrained Models**
  https://www.sbert.net/docs/pretrained_models.html