¿Qué me he perdido?

Marta Aguilar Morcillo
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
maragumor@alum.us.es, nmy0786

Candela Jazmín Gutiérrez González
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
cangutgon@alum.us.es, dgl2523

***Resumen*—El objetivo principal del trabajo es la construcción y evaluación de distintos sistemas de recuperación de la información que permita cumplir con las necesidades de información solicitadas sobre un conjunto determinado de documentos.**

En este proyecto se han puesto a prueba nuestros conocimientos sobre el procesamiento del lenguaje natural mediante el cumplimiento de los objetivos del trabajo y los resultados exitosos obtenidos.

***Palabras clave*—Inteligencia Artificial, procesamiento del lenguaje natural, tf-idf, corpus...**

I. Introducción

En la actualidad, somos capaces de realizar una búsqueda en internet y obtener los resultados solicitados o unos muy parecidos. Esto se debe al estudio y desarrollo de la Inteligencia Artificial y del Procesamiento del Lenguaje Natural [2], pues hasta el momento, dichas búsquedas debían ser realizadas por un bibliotecario con una lista inmensa de libros.

La recuperación de información es empleado en infinidad de ámbitos, pero sobre todo es necesitada en tareas cotidianas: búsqueda de correos, cualquier búsqueda en una página web, búsqueda de mensajes en un chat, búsqueda de una canción en una plataforma de música, al igual que películas o series... Dichas recopilaciones también son aplicables a algunos ámbitos específicos, por ejemplo, en el área de la salud es utilizado para la obtención de expedientes de los pacientes; en el área administrativos se emplea para la consulta de archivos históricos.

En este proyecto se desarrolla un modelo en Python que analiza noticias pertenecientes a un corpus para devolver aquellas más similares o que contengan unos términos determinados.

Para ello, se ha obtenido un conjunto de noticias actuales gracias a la página web Kaggle [3], el cual fue modificado. A partir del csv obtenido tras los cambios, se emplearon distintas herramientas para su procesamiento: Pandas [10], para la lectura de csv, NLTK [4], para el análisis y normalización del texto, Whoosh [9], para la creación de índices invertidos y la realización de consultas de búsqueda, Contractions [6], para evitar contracciones de palabras en el proceso de normalización, Sklearn [8], para la aplicación de tf-idf (planificación automática), Spacy [5], para la obtención de palabras compuestas en el procesamiento de los textos.

II. Preliminares

En esta sección se hará una breve introducción de la técnicas empleadas.

A. Métodos empleados

En este trabajo se han empleado distintas técnicas de normalización, así como distintos tipos de sistemas de recuperación de datos. A continuación, se explicarán brevemente.

A partir de la recopilación de noticias modificada se debe realizar un procesamiento de dichos datos. El proceso consistirá en eliminar aquellas palabras que no sean necesarias a la hora de analizar la información, como las StopWorlds, palabras comunes que no aportan significado, para finalmente emplear técnicas de tokenización y stemming.

Se continuará con la misma aplicación de procesamiento de texto a las queries. Posteriormente, se deberá realizar una construcción de índice invertido, para el primer sistema en las queries, y en el segundo sistema, tanto en el corpus como en las queries.

A continuación, se dividirán las técnicas empleadas:

- Sistema de recuperación booleano: tras parsear las queries se evaluará la precisión obtenida.
- Sistema de recuperación de texto libre: se realiza el cálculo de la matriz tf-idf de los documentos filtrados. Posteriormente, se calcula de la misma forma la matriz tf-idf de la query con el mismo vocabulario que la anterior matriz.Finalmente, se calcula el coseno de similitud con cada documento del corpus, devolviendo el índice de los documentos más similares junto al valor calculado. Para la comprobación del cumplimiento del objetivo, se analiza la precisión.

Tras realizar el análisis de cada tipo de sistema de recuperación de información y evaluar sus resultados, se escoge aquellos con mejores resultados.

B. Trabajo Relacionado

Para la realización del trabajo, se han empleado ejercicios realizados en las clases de práctica, fuentes e inteligencia artificial generativa:

1) **Ejercicios de Clase** Se han utilizado ejercicios realizados en las clases de práctica como base para la resolución de problemas y la comprensión de los conceptos clave.

   a) Apuntes del tema de aprendizaje automático [1].
   b) Apuntes del tema de procesamiento del lenguaje natural [2].

2) **Fuentes Consultadas** Se han revisado materiales bibliográficos y artículos relevantes para fundamentar el trabajo, proporcionando un contexto teórico sólido.

   a) NTLK [4]:StopWords, Contractions, pos_tag, WordNetLemmatizer.
   b) Spacy. [5]
   c) Sklearn. [8]
   d) Spacy [5]
   e) Whoosh [9]

3) **Inteligencia Artificial Generativa** Se ha empleado IA generativa para la comprensión de ciertas librerías o conocer nuevas, y para descubrir el significado de errores surgidos. A continuación, se introducirán las prompts que más nos han ayudado:

   a) Prompt: ”Cómo se pueden obtener las palabras compuestas de un texto, pues, no es lo mismo tráfico de drogas que tráfico”.
   b) Prompt: ”¿Se podría modificar el número de palabras que debe de relacionar como una?” .
   c) Prompt: ”Cómo se podría sacar sinónimos de una palabra”.
   d) Prompt: ”Cómo funciona TfidfVectorizer de sklearn”.
   e) Prompt: ”Cómo indicarle a TfidfVectorizer la detección de palabras conjuntas”.
   f) Prompt: ”Cómo calcular el coseno de similitud de tfidf de sklearn, indicame las entradas y salidas de la función”.
   g) Prompt: ”Dado el título y el cuerpo de una noticia, genera de 1 a 3 consultas especificadas según el modelo booleano, cada una representando un posible tema o enfoque informativo relevante que se pueda buscar en un sistema de recuperación de información. Cada consulta debe estar expresada como una cadena booleana con operadores como AND, OR y NOT, incluyendo nombres propios, eventos clave, lugares o conceptos importantes del texto. Para cada consulta, indica también a qué noticia se refiere con un identificador (por ejemplo, un número) en un campo relevant_docs. Devuélvelo como una lista de diccionarios en formato Python, con una estructura por elemento que incluya los campos tema, query y relevant_docs”.
   h) Prompt: ”Explícame de forma clara y estructurada cómo utilizar la biblioteca Whoosh en Python para construir un sistema de recuperación de información. Incluye: Qué es Whoosh y para qué sirve. Cómo crear un índice a partir de un corpus de documentos. Cómo definir el esquema del índice (campos como título, contenido, ID...). Cómo agregar documentos al índice. Cómo realizar búsquedas con queries simples y queries booleanas. Cómo recuperar documentos relevantes y sus puntuaciones. Cómo usar el modelo BM25F u otros modelos de scoring en Whoosh. Cómo evaluar los resultados con precisión y recall. Ejemplos prácticos aplicados a un sistema que recibe consultas y devuelve documentos relevantes. La explicación debe ayudarme a integrar Whoosh en un sistema de recuperación de información basado en consultas booleanas o por similitud textual. Usa ejemplos en código Python si es necesario.”.

III. Metodología

En esta sección se explicará con un mayor nivel de detalles lo descrito en la sección Preliminares divididos en los siguientes pasos:

1) **Recopilación de datos.** Para obtener el corpus necesario para nuestro proyecto se accedió a la página Kaggle, la cual cuenta con numerosos Database en formato CSV, y se buscarón aquellos que fueran de noticias. Tras ello, el CSV fue modificado mediante la eliminación de columnas no necesarias para la recuperación de información y el recorte de noticias para evitar problemas de rendimiento con los dispositivos de trabajo, siendo procesado con la librería Pandas:

	author	headlines	text
0	Chhavi Tyagi	Daman & Diu revokes mandatory Rakshabandhan in...	The Daman and Diu administration on Wednesday ...
1	Daisy Mowke	Malaika slams user who trolled her for 'divorc...	From her special numbers to TV? appearances, Bo...
2	Arshiya Chopra	'Virgin' now corrected to 'Unmarried' in IGIMS...	The Indira Gandhi Institute of Medical Science...
3	Sumedha Sehra	Aaj aapne pakad liya: LeT man Dujana before be...	Lashkar-e-Taiba's Kashmir commander Abu Dujana...
4	Aarushi Maheshwari	Hotel staff to get training to spot signs of s...	Hotels in Mumbai and other Indian cities are t...
5	Sonu Kumari	Man found dead at Delhi police station, kin al...	An alleged suspect in a kidnapping case was fo...
6	Parmeet Kaur	Delhi HC reduces aid for 'negligent' accident ...	In an interesting ruling, the Delhi high court...
7	Chhavi Tyagi	60-yr-old lynched over rumours she was cutting...	A 60-year old Dalit woman was allegedly lynch...
8	Parmeet Kaur	Chopper flying critically low led to 2015 Bomb...	Two years after a helicopter crash near the Bo...
9	Sumedha Sehra	Congress opens 'State Bank of Tomato' in Lucknow	It sounds like satire, but make no mistake: at...

Fig. 1. DataBase con noticias

2) **Procesamiento de texto.** Para eliminar aquellas palabras que no aportan información relevante se empleó la librería NLTK y spacy, una librería similar a la incluida en NLTK de Stop Words que permite hallar aquellas palabras cuyo significado en conjunto es más valioso que separadas, por ejemplo: ”Artificial Intelligence”.

limpiar_texto($texto$)
Entrada: una cadena de texto
Salida: una cadena de texto limpia
reemplazar caracteres no alfabéticos con espacios
eliminar espacios extra
convertir todo el texto a minúsculas
devolver texto limpio

elimina_html($contenido$)
Entrada: una cadena con posible código HTML
Salida: una cadena sin etiquetas HTML
transformar contenido en objeto `BeautifulSoup`
extraer el texto sin etiquetas
devolver el contenido limpio

expandir_constracciones($contenido$)
Entrada: una cadena con contracciones
Salida: una cadena sin contracciones
aplicar la función `contractions.fix` al contenido
devolver la cadena expandida

lematizador($contenido$)
Entrada: una lista de palabras tokenizadas
Salida: una lista con las palabras lematizadas
inicializar el lematizador `WordNetLemmatizer`
obtener etiquetas POS de cada palabra
para cada palabra en contenido
    si la etiqueta es verbo
        lematizar como verbo
    si no, lematizar con su forma base
devolver la lista lematizada

3) **Lectura completa.** Lo explicado anteriormente se reúne en tres tipos de lecturas:

   a) Lectura normal. Se procesan las palabras

   lectura_final()
   Entrada: archivo CSV con noticias
   Salida: lista de textos procesados
   cargar el archivo CSV
   $resultados$ – **lista vacía**
   para cada $index, fila$ **en** $filas(lectura\_csv)$
       extraer el autor, título y cuerpo
       $a_p \leftarrow$ **proceso_contenido(autor)**
       $t_p \leftarrow$ **proceso_contenido(título)**
       $c_p \leftarrow$ **proceso_contenido(cuerpo)**
       $fila \leftarrow a_p + t_p + c_p$
       $resultados \leftarrow resultados + f$
   devolver `resultados`

   b) Lectura con detección de palabras conjuntas.

   lectura_normalizada_corpus()
   Entrada: archivo CSV con noticias
   Salida: lista de textos procesados
   cargar el archivo CSV
   $resultados$ – **lista vacía**
   para cada $index, fila$ **en** $filas(lectura\_csv)$
       extraer el autor, título y cuerpo
       $a_p \leftarrow$ **proceso_contenido(autor)**
       $t_p \leftarrow$ **proceso_contenido(título)**
       $c_p \leftarrow$ **proceso_contenido(cuerpo)**
       $fila \leftarrow a_p + t_p + c_p$
       $res \leftarrow$ **extraer_noun_chunks(fila)**
       $resultados \leftarrow resultados + res$
   devolver `resultados`

   c) Lectura con detección de palabras conjuntas y sinónimos. Para ello, se necesitarán dos funciones adicionales:

   expand_term($term$)
   Entrada: un término en texto
   Salida: un conjunto de sinónimos del término
   $related$ – **conjunto vacío**
   para cada syn **en sinónimo(palabra)**
       para cada $lemma$ **en lemmas(sinónimo)**
           $p \leftarrow$ **normalización(lemma)**
           si $word \neq term$ **entonces**
               $related \leftarrow$ **related + p**
   devolver $related$

   expand_corpus_con_sinonimos($documento$)
   Entrada: un documento tokenizado
   Salida: documento expandido con sinónimos
   $doc\_counter$ – **contador de frecuencias**
   new_doc – **lista vacía**
   para cada $word, count$ **en** $doc\_counter$
       $new\_doc \leftarrow$ **new_doc + word** $\times$ $count$
       $synonyms \leftarrow$ **expandir_term(word)**
       para cada syn **en** $synonyms$
           $new\_doc \leftarrow$ **new_doc + syn** $\times$ $count$
       devolver new_doc

   lectura_normalizada_corpus_sinonimos()
   Entrada: archivo CSV con noticias
   Salida: lista de textos procesados
   cargar el archivo CSV
   $resultados$ – **lista vacía**
   para cada $index, fila$ **en** $filas(lectura\_csv)$
       extraer el autor, título y cuerpo
       $a_p \leftarrow$ **proceso_contenido(autor)**
       $t_p \leftarrow$ **proceso_contenido(título)**
       $c_p \leftarrow$ **proceso_contenido(cuerpo)**
       $fila \leftarrow a_p + t_p + c_p$
       $pc \leftarrow$ **extraer_noun_chunks(fila)**
       $res \leftarrow$ **extraer_corpus_con_sinónimos(pc)**
       $resultados \leftarrow resultados + res$
   devolver `resultados`

4) **Aplicación de índices invertidos.** Para facilitar la comparación de valores tf-idf, realizada más adelante, se aplican índices invertidos con la biblioteca Whoosh:

crear_indice_whoosh($corpus\_normalizado$)
Entrada: lista de documentos normalizados
Salida: índice Whoosh creado con los docs procesados
si existe("$indice\_whoosh$") **entonces**
    eliminar "$indice\_whoosh$"
crear directorio "$indice\_whoosh$"
definir $esquema$
$ix \leftarrow$ **create_in**("$indice\_whoosh$", $esquema$)
$writer \leftarrow$ **ix.writer**()
para cada i, doc **en enumerar**($corpus\_normalizado$)
    $contenido \leftarrow$ **espaciado**(doc")
    writer.add_document($id, contenido$)
writer.commit()

buscar_con_whoosh($tokens\_query, corpus\_n$)
Entrada: lista de tokens de consulta y corpus normalizado
Salida: documentos encontrados y sus índices
$ix \leftarrow$ **abrir directorio**("$indice\_whoosh$")
searcher $\leftarrow$ **buscar_directorio**()
$terms -$ **lista de índices**
para cada i **en tokens_query**
    $terms \leftarrow$ **obtener_term(contenido, token)**
$query \leftarrow$ **union_por_OR**($terms$)
$resultados \leftarrow$ **buscar**($query$)
$docs\_encontrados \leftarrow$ **lista vacía**
$indices\_docs \leftarrow$ **lista vacía**
para cada hit **en** $resultados$
    $docs\_encontrados \leftarrow$
    docs_encontrados + corpus_n[doc_id]
    $indices\_docs \leftarrow$ **indices_docs + doc_id**
devolver $docs\_encontrados, indices\_docs$

5) **Cálculo de valores tf-idf.** Para la comparación de valores tf-idf se debe calcular primero la matriz tf-idf del corpus reducido, para posteriormente calcular los valores tf-idf de la consulta con la misma matriz:

tfidf_por_corpus($corpus\_normalizado$)
Entrada: lista de documentos normalizados
Salida: matriz TF-IDF y vocabulario de términos
$texts -$ **lista de docs con palabras espaciadas**
para cada doc **en corpus**
    $texto \leftarrow$ **texts +**
    docs_separadas_espacios(word)
$v \leftarrow$ **vectorizados_tfidf(ngrama 1,2)**
$X \leftarrow$ **matriz_tfidf_tfidf(texts)**
$terminos \leftarrow$ **vocabulario_matrix_tfidf(X)**
devolver $X, terminos, vectorizer$

Para obtener la matriz de la query se le aplica el mismo procedimiento pero con el vectorizador del corpus y siendo solo un documento la entrada, en vez de varios.

6) **Comparación de tf-idf (S.R.I. texto libre).** Comparación de TF-IDF: Para determinar qué palabras de la consulta (aquellas con valores de TF-IDF distintos de cero) son más similares a las presentes en cualquier documento del corpus, se calcula la similitud del coseno entre sus respectivos vectores TF-IDF. Esta métrica permite cuantificar la semejanza semántica entre el contenido de la consulta y los documentos analizados.

similitud_coseno($tfidf\_corpus, tfidf\_doc, indices$)
Entrada: matriz tf-idf corpus, de la query, vectorizer
Salida: lista de tuplas (índice documento, similitud)
$similitudes \leftarrow$ **cosine_sim(tfidf_doc, tfidf_corpus)**
$i\_filtrados -$ **lista de índices con sim no nula**
para cada i, sim **en i_similitudes**
    $i\_filtrados \leftarrow$ **i_filtrados + i**
$i\_ordenados \leftarrow$ **ordena_i_por_sim(i_filtrados)**
$res -$ **lista de tuplas resultado**
para cada i **en i_ordenados**
    $res \leftarrow$ **res + (indices[i], similitudes[i])**
devolver res

7) **Evaluación de consultas.** Las consultas se encuentran en formato json incluyendo cada una la query y los documentos interesantes. Por lo tanto, para la evaluación será necesario realizar la lectura correspondiente y comparar los resultados con las supuestas soluciones:

calcular_metricas($res\_sim, docs\_relevantes, i\_posibles$)
Entrada: tuplas (i_doc, sim),docs relevantes, índices pos.
Salida: métricas (AP, precisión, recall), [i], [scores]
$y\_tp, y\_fp, y\_scores, idxs \leftarrow$ **listas vacías**
para cada $idx, score$ **en** res_sim
    si $idx \in docs\_relevantes$ **entonces**
        $y\_tp \leftarrow y\_tp + 1$
    sino
        $y\_fp \leftarrow y\_fp + 1$
    $y\_scores \leftarrow y\_scores + score$
    $idxs \leftarrow idxs + idx$
$no\_relevantes \leftarrow$ **índices_no_en(i_posibles)**
$y\_tn -$ **lista de verdadero negativos**
para cada i **en i_posibles**
    $y\_tn \leftarrow$ **y_tn + 1**
$y\_fn -$ **lista de falsos negativos**
para cada i **en i_posibles**
    $y\_fn \leftarrow$ **y_tn + 1**
si $existe\_algun(y\_tp)$ **entonces**
    $ap \leftarrow$ **average_precision_score**(y_tp, y_scores)
    $precision \leftarrow \frac{\sum y\_true\_positive}{\sum y\_true\_positive + \sum y\_false\_positive}$
    $recall \leftarrow \frac{\sum y\_true\_positive}{\sum y\_true\_positive + \sum y\_false\_negative}$
    devolver $ap, precision, recall, idxs, y\_scores$
sino
    devolver $None, None, None, idxs, y\_scores$

evaluar_consultas_por_similitud$(consultas, corpus\_n)$
Entrada: lista de consultas y corpus normalizado
Salida: MAP scores, precision, recall

```
crear_indice_whoosh(corpus_n)
m_scores, m_precision, m_recall − listas vacías
total_i − lista de índices (0,148)
para cada consulta en consultas
    texto_query ← consulta["query"]
    docs_correctos ← consulta["documentos"]
    lectura ← lectura_documento(texto_query)
    docs_relevantes, indices_docs ←
    buscar_con_whoosh(lectura, corpus_n)
    tfidf_corpus, terms, vectorizer ←
    tfidf_por_corpus(docs_relevantes)
    tfidf_documento ←
    tfidf_del_documento(lectura, vectorizer)
    res_sim ←
    similitud_coseno(tfidf_corpus, )
    tfidf_documento, indices_docs)
    ap, precision, recall, idxs, y_scores ←
    calcular_metricas(res_sim, docs_relevantes,
    total_i)
    si ap ≠ None entonces
        m_scores ← m_scores + ap
        m_precision ← m_precision + precision
        m_recall ← m_recall + recall
        mostrar_resultados(texto_query,
        documentos_relevantes, res_sim,
        ap, precision, recall)
    sino
        error
devolver m_scores, m_precision, m_recall
```

calcular_metricas$(doc\_encontrados, doc\_relevantes)$
Entrada: documentos encontrados y relevantes
Salida: precisión y recall

```
verdaderos_positivos ←
doc_encontrados ∩ doc_relevantes
si doc_encontrados ≠ ∅ entonces
    precision ← |verdaderos_positivos| / |doc_encontrados|
sino
    precision ← 0
si doc_relevantes ≠ ∅ entonces
    recall ← |verdaderos_positivos| / |doc_relevantes|
sino
    recall ← 0
devolver precision, recall
```

ejecutar_y_evaluar_consultas$(consultas, indice\_path)$
Entrada: lista de consultas y ruta al índice Whoosh
Salida: precisión total, recall total y número de consultas

```
ix ← abrir_directorio_indice(indice_path)
parser ← crear_parser("content", ix.schema)
total_precision ← 0
total_recall ← 0
num_consultas ← total(consultas)
usar searcher ← ix.searcher(BM25F):
    para cada consulta en consultas
        query_text ← consulta["query"]
        documentos_relevantes ←
        consulta["documentos"]
        tema ← consulta["tema"]
        intentar
            query ← parsear(query_text)
            resultados ← buscar(query, sin_límite)
            documentos_encontrados ← {r["doc_id"]
            para cada r ∈ resultados}
        si error
            mostrar_error(query_text)
            continuar
        precision, recall ← calcular_metricas
        total_precision ← total_precision + precision
        total_recall ← total_recall + recall
        imprimir_metricas_individuales
devolver total_precision, total_recall, num_consultas
```

8) **Comparación de sistemas).** Tras la realización de la comparación de sistemas, explicado en la sección Resultados se han elegido los sistemas que devolvieron los resultados más exitosos, dejando como resultado:

IV. Resultados

Este proyecto trata dos tipos de recuperación de información, una a partir de una consulta booleana y la otra a partir de un texto. Por ello, se han evaluado distintos sistemas para cada tipo:

A. Sistema de recuperación de información booleano

Para aquellas consultas que cumplen con el modelo booleano, se han evaluado las siguientes alternativas, variando las técnicas de normalización de términos y las consultas a evaluar:

1) **S.R.I. con lematización y consultas generales.** Consiste en que entre las técnicas de normalización de términos aplicadas al procesamiento de los documentos del corpus se incluye la lematización. Las consultas utilizadas para evaluar el sistema de recuperación de información están diseñadas para ser generales, es decir, se espera que sean satisfechas por varios documentos del corpus.

```
===== MÉTRICAS PROMEDIO PARA CONSULTAS GENERALES CON LEMATIZACIÓN =====
Precisión promedio: 0.86
Recall promedio: 1.00
```

Fig. 2. Métricas MAP del S.R.I. booleano con lematización y consultas generales

2) **S.R.I. sin lematización y consultas generales.** En este caso, entre las técnicas de normalización aplicadas no se incluye la lematización. Las consultas generales están diseñadas para ser satisfechas por múltiples documentos del corpus.

```
===== MÉTRICAS PROMEDIO PARA CONSULTAS GENERALES SIN LEMATIZACIÓN =====
Precisión promedio: 0.86
Recall promedio: 1.00
```

Fig. 3. Métricas MAP del S.R.I. booleano sin lematización y consultas generales

3) **S.R.I. con lematización y consultas específicas.** Se aplica lematización al preprocesar los documentos del corpus. Las consultas utilizadas en este caso están diseñadas para recuperar un documento concreto, lo que permite evaluar la precisión del sistema en situaciones más restrictivas.

```
===== MÉTRICAS PROMEDIO PARA CONSULTAS ESPECÍFICAS CON LEMATIZACIÓN =====
Precisión promedio: 0.98
Recall promedio: 1.00
```

Fig. 4. Métricas MAP del S.R.I. booleano con lematización y consultas específicas

4) **S.R.I. sin lematización y consultas específicas.** En esta configuración, se omite la lematización en la normalización del corpus. Las consultas específicas están pensadas para recuperar un único documento relevante, poniendo a prueba la sensibilidad del sistema frente a la falta de lematización.

```
===== MÉTRICAS PROMEDIO PARA CONSULTAS ESPECÍFICAS SIN LEMATIZACIÓN =====
Precisión promedio: 0.97
Recall promedio: 0.98
```

Fig. 5. Métricas MAP del S.R.I. booleano sin lematización y consultas específicas

Para su evaluación, cada sistema obtiene una lista con los índices de los documentos que cumplen la consulta y la compara con la lista de índices relevantes asociada a dicha consulta (almacenada en un archivo `.json`). A partir de esta comparación, se calculan las métricas **MAP**, **precisión** y **recobrado (recall)**.

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

- **Configuración:** se varían las técnicas de normalización (con y sin lematización) y la naturaleza de las consultas (generales vs. específicas).
- **Objetivo:** determinar cómo afectan estas variaciones al rendimiento del sistema de recuperación.
- **Resultados:** se obtienen las métricas MAP, precisión y recall para cada configuración.
- **Análisis:** permite observar qué combinación ofrece un mejor equilibrio entre precisión y cobertura en función del tipo de consulta y el procesamiento aplicado.

a) Análisis de resultados.: Como podemos observar en las imágenes, para los sistemas que se evalúan usando **consultas generales** se obtiene:

- **Precisión promedio:** 0.86. Indica que, en promedio, el 86% de los documentos que el sistema recuperó para las consultas fueron realmente relevantes (es decir, estaban en la lista de documentos esperados). Esto significa que hay algo de ruido o falsos positivos (documentos recuperados que no deberían estar), pero no es muy alto.
- **Recall promedio:** 1.00. Significa que el sistema recuperó todos los documentos relevantes en promedio para cada consulta. En otras palabras, no se dejó fuera ningún documento que debía recuperar. Esto es muy positivo, indica que la recuperación es completa.

Mientras que para los sistemas que se evalúan usando **consultas específicas** se obtiene:

- **Precisión promedio:** 0.97 sin lematización y 0.98 con lematización. Indica que, en promedio, entre el 97% y el 98% de los documentos que el sistema recuperó para las consultas fueron realmente relevantes.
- **Recall promedio:** 1.00 con lematización, lo que significa que el sistema recuperó todos los documentos relevantes en promedio para cada consulta. Sin lematización se obtiene un recall de 0.98 porque hay un caso en el que no se recuperó el documento relevante.

B. Sistema de recuperación de información texto libre

Para aquellas consultas realizadas a partir de texto libre, se han evaluado las siguientes alternativas:

1) **S.R.I. sin sinónimos.** Consiste en el análisis de ambos textos y comparar las similitudes de sus términos, obteniendo las siguientes métricas MAP:

```
======= MÉTRICAS FINALES SIN SINÓNIMOS =======
Mean Average Precision (MAP): 0.9006

========================
Average Precision: 0.1739

========================
Average Recall: 0.9574
```

Fig. 6. Métricas MAP del S.R.I. texto libre sin sinónimos

2) **S.R.I. con sinónimos.** Consiste en realizar el mismo procesamiento que el sistema anterior, sin embargo, se le añade otro procesamiento mediante la función explicada anteriormente $expand\_corpus\_con\_sinonimos(documento)$ incluida en $lectura\_normalizada\_corpus\_sinonimos()$

```
======= MÉTRICAS FINALES CON SINÓNIMOS =======
Mean Average Precision (MAP): 0.8786

========================
Average Precision: 0.0675

========================
Average Recall: 0.9747
```

Fig. 7. Métricas MAP del S.R.I. texto libre con sinónimos

En las imágenes de ambos sistemas, se puede ver primeramente la precisión promedia de las queries, la cual mide qué tan pronto y ordenados aparecen los documentos relevantes en los resultados.

A continuación, se visualiza la precisión, la cual nos indica la proporción de documentos devueltos que sí eran relevantes. Para ello, se realiza una división entre los verdaderos positivos, documentos relevantes correctamente recuperados, y la suma de los verdaderos positivos, y los falsos positivos, que serían aquellos documentos recuperados que no son documentos relevantes.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Finalmente, se ha querido incluir el recall, el cual nos indica la proporción de todos los relevantes posibles que fuiste capaz de recuperar. Para ello, se realiza una división entre verdaderos positivos y la suma de verdaderos positivos y falsos negativos, los cuales hacen referencia a aquellos documentos que son recuperados pero no son documentos relevantes.

$$\text{Recall} = \frac{TP}{TP + FN}$$

V. Conclusiones

A lo largo del trabajo se ha desarrollado un sistema de recuperación de información basado en técnicas de procesamiento del lenguaje natural, con el objetivo de evaluar la eficacia de distintos métodos aplicados sobre un corpus de noticias. Se implementaron tanto sistemas booleanos como de texto libre, usando bibliotecas como NLTK, Spacy, Sklearn y Whoosh. Los experimentos se centraron en analizar el impacto de técnicas como la lematización y el uso de sinónimos, evaluando su rendimiento mediante métricas como MAP, precisión y recall.

Los resultados obtenidos indican que la inclusión de lematización mejora ligeramente la precisión en consultas booleanas específicas, pues en las generales el cambio es nulo, mientras que para sistemas de texto libre, la inclusión de sinónimos en el corpus incrementa la cobertura de documentos relevantes, es decir, mejora significativamente el recall. No obstante, también se observó una disminución en la precisión, lo cual se debe a la indeterminación semántica.

Como conclusión, llegamos a la decisión de elegir aquellos sistemas que devolvieran mayor precisión, pues es más interesante para nosotros la recuperación correcta de documentos. Sin embargo, tampoco sería incorrecto elegir, en el caso de consultas de texto libre, aquella con mayor recall, pues su punto fuerte sería la gran variedad de palabras que puede aceptar debido a su cobertura.

Una de las posibles mejoras a implementar sería el uso de modelos más avanzados para la detección de relaciones semánticas más profundas pues podría mejorar la precisión de los sistemas que implementen sinónimos.

Referencias

[1] Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla, "Aprendizaje automático".
[2] Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Sevilla, "Procesamiento del lenguaje natural".
[3] Kaggle. URL: https://www.kaggle.com/datasets/sunnysai12345/news-summary.
[4] NLTK, URL: https://www.nltk.org/.
[5] Spacy, URL: https://spacy.io/.
[6] Contractions, URL: https://pypi.org/project/contractions/.
[7] Wordnet, URL: https://wordnet.princeton.edu/.
[8] Sklearn, URL: https://scikit-learn.org/stable/.
[9] Whoosh, URL: https://whoosh.readthedocs.io/en/latest/.
[10] Pandas, URL: https://pandas.pydata.org/.