Proyecto de prácticas:
Buscador web

Sistemas de Almacenamiento y Recuperación de Información
(Proyecto coordinado SAR - ALT)
Primera parte

2024-2025, v0.3

Trabajo en grupo (grupos de 4 personas). Los grupos se deben apuntar en PoliformaT.

Objetivo

El presente proyecto de prácticas de SAR es la primera parte del proyecto conjunto entre las asignaturas de *Sistemas de Almacenamiento y Recuperación de Información* (SAR, semestre 3B) y *Algorítmica* (ALT, semestre 4A). Esto implica que el código que desarrolle cada grupo se utilizará, por el mismo grupo, como base para el proyecto de la asignatura de ALT en el próximo curso.

El objetivo del presente proyecto es la implementación de un sistema de recuperación de información para artículos de la Wikipedia.

En PoliformaT/Lessons se proporcionan los cinco ficheros siguientes con extensión `.py`:

- `SAR_Indexer.py`. Programa principal para la indexación de artículos. Crea un objeto de la clase `SAR_Project` para extraer los artículos de una colección alojada en un directorio local, los indexa y guarda en disco la instancia del objeto. **No se debe modificar**.
- `SAR_Searcher.py`. Programa principal para la recuperación de artículos. Carga de disco una instancia de la clase `SAR_Project` para responder a las consultas que se le realicen. **No se debe modificar**.
- `SAR_lib.py`. Librería para la definición de la clase `SAR_Project`. Las funcionalidades de indexación y recuperación se pueden conseguir completando un número reducido de métodos de esa clases. **Se debe modificar**.

Nota: Solamente hay que modificar el fichero `SAR_lib.py`, los otros dos (`SAR_Indexer.py` y `SAR_searcher.py`) ni se modifican ni se entregan).

Entrega y Evaluación

En PoliformaT se abrirá una tarea para que el representante de cada grupo suba:

- El fichero `SAR_lib.py` con las funcionalidades implementadas.
- Una **memoria** del proyecto en formato pdf con el nombre `informe.pdf`.

El proyecto de prácticas se debe hacer en **grupos de 4 personas**, se aceptarán grupos de 3 integrantes en casos de fuerza mayor, en ningún caso se aceptarán grupos de más de 4 personas. Debéis apuntaros a uno de los grupos "ProyPrac" del Poliformat (*Sar/Información del sitio/Grupos a los que puede unirse*) **antes del día 16 de abril** (tercera sesión de prácticas del proyecto). Debéis apuntaros a uno de los grupos correspondiente a vuestro grupo de prácticas; los que comienzan por 3CO11 para mañanas, y 3CO21 para tardes. En caso de haber integrantes de mañana y tardes, se deberán inscribir a la sesión a la que pertenezcan la mayoría de componentes. En caso de empate, es indistinto a qué grupo os unáis.

Se permite realizar nuevas entregas mientras la tarea de PoliformaT esté abierta (sólo se considerará la última entrega). Tanto en el fichero fuente como en la memoria deberán identificarse TODAS las funcionalidades implementadas, TODOS los miembros del grupo y qué miembro ha implementado cada funcionalidad. La no inclusión de dicho listado (funcionalidad, miembro que la ha realizado) conllevará una penalización.

El proyecto de prácticas tiene un peso total de **2 puntos**. Se hará una evaluación de cada grupo en el laboratorio, puede incluir una prueba escrita individual. La aplicación debe contar con unas **funcionalidades mínimas** que se puntuarán con un máximo de **1 punto**. Adicionalmente, se pueden **ampliar las funcionalidades** para obtener mayor nota, hasta un máximo total de **1 punto adicional**. La parte adicional sólo se tendrá en cuenta si la parte obligatoria funciona correctamente.

Esta prueba es recuperable y podrán presentarse tanto los alumnos suspendidos como aprobados. La recuperación podrá variar en función del trabajo inicial realizado. Los alumnos suspendidos que se presenten a la recuperación optarán a una nota máxima de 5 sobre 10 (1 punto). Aquellos alumnos que estando aprobados se presenten para subir nota podrán optar al 10 (2 puntos). En ambos casos la nota final será la de la recuperación, sea más alta o más baja que la nota original.

1. Funcionalidades básicas (hasta 1 punto)

1.1. Indexador (`SAR_Indexer.py`)

La Figura 1 muestra el mensaje de ayuda del programa `SAR_Indexer.py`. Se ha incluido la biblioteca `argparse`, para facilitar el análisis de los argumentos de entrada del programa. Este análisis ya está incluido en el código que hay disponible en PoliformaT y, por tanto, no se debe modificar ni subir el fichero `SAR_Indexer.py`.

```
$ python SAR_Indexer.py --help
usage: SAR_Indexer.py [-h] [-P] [-S] dir index

Index a directory with Wikipedia articles in json format.

positional arguments:
  dir               directory with the Wikipedia articles.
  index             name of the index.

options:
  -h, --help        show this help message and exit
  -P, --positional  compute positional index.
  -S, --semantic    compute the semantic index.
```

Figura 1: Ayuda del `SAR_Indexer.py` (incluye las funcionalidades ampliadas).

Las funcionalidades básicas con las que debe contar el indexador son las siguientes:

- Requiere obligatoriamente dos argumentos de entrada:
  - **dir**, el directorio donde está la colección de artículos en formato `json`, y
  - **index**, el nombre del fichero donde se guardará la información del índice creado.
- El programa crea un objeto de la clase `SAR_Indexer` e invoca a su método `index_dir`.

  El método `index_dir`, junto con el resto de métodos a los que éste llame, deberá:
  - Procesar los documentos `.json` del directorio de manera recursiva y extraer los artículos.
  - Tokenizar cada artículo:
    - Eliminando símbolos no alfanuméricos (comillas, interrogantes, ...) y
    - extrayendo los términos (consideraremos separadores de términos los espacios, los saltos de línea y los tabuladores).

    No se deben distinguir mayúsculas y minúsculas en la indexación.
  - Asignar a cada documento procesado un identificador único (docid) que será un entero secuencial. En este contexto un documento es un fichero en memoria secundaria.
  - Asignar a cada artículo un identificador único. Se debe saber a qué documento (fichero) pertenece cada artículo y qué posición relativa ocupa dentro de él.
  - Crear un índice invertido accesible por término. Cada entrada contendrá una lista con los artículos en los que aparece ese término.
- Finalmente, el programa principal guarda en disco toda la información del índice con el método `save_info`. También muestra información sobre el proceso de indexado.

```
indexer = SAR_Indexer()
t0 = time.time()
indexer.index_dir(args.dir, **vars(args))
t1 = time.time()
indexer.save_info(args.index)
t2 = time.time()
indexer.show_stats()
print("Time indexing: %2.2fs." % (t1 - t0))
print("Time saving: %2.2fs." % (t2 - t1))
print()
```

A continuación se muestra un extracto del método `index_dir` ya incluido en la biblioteca.

```
def index_dir(self, root, **args):
    ...
    for d, subdirs, files in os.walk(root):
        for filename in files:
            if filename.endswith('.json'):
                fullname = os.path.join(d, filename)
                self.index_file(fullname)
    ########################################
    ## COMPLETAR PARA FUNCIONALIDADES EXTRA ##
    ########################################
```

Como se puede observar, el método realiza el recorrido recursivo por el directorio utilizando la función `walk` de la biblioteca `os`. Para conseguir la funcionalidad básica de indexado sólo hace falta completar el método `index_file`.

```
def index_file(self, filename):
    ...
    for i, line in enumerate(open(filename)):
        j = self.parse_article(line)

        #################
        ### COMPLETAR ###
        #################
```

El método `parse_article` recibe una cadena (objeto json) y devuelve un diccionario procesado en el que todos los valores son cadenas. Las claves de este diccionario son:

- `url`, dirección del artículo. Este campo identifica al artículo, no se pueden indexar dos artículos con la misma url.
- `title`, título del artículo.
- `summary`, resumen del artículo. Parte inicial del artículo previa a las secciones.
- `all`, el contenido de todo el artículo. Incluye título, resumen y todas las secciones y subsecciones.
- `section-name`, nombre de todas las secciones y subsecciones del artículo.

La variable `DEFAULT_FIELD = 'all'` indica el campo que se debe indexar.

1.2. Recuperador de artículos (SAR_Searcher.py)

La Figura 2 muestra el mensaje de ayuda del programa `SAR_Searcher.py`. Este fichero no se debe modificar ni subir a PoliformaT.

```
$ python SAR_Searcher.py --help

usage: SAR_Searcher.py [-h] [-C] [-A] [-Q query | -L qlist | -T test] [-S STHRESHOLD | -R] index

Search the index.

positional arguments:
  index                 name of the index.

options:
  -h, --help            show this help message and exit
  -C, --count           show only the number of documents retrieved.
  -A, --all             show all the results. If not used, only the first 10 results are showed. Does

Batch modes:
  -Q query, --query query
                        query.
  -L qlist, --list qlist
                        file with queries.
  -T test, --test test  file with queries and results, for testing.

Semantic Search:
  -S STHRESHOLD, --semantic_threshold STHRESHOLD
                        threshold for the semantic search.
  -R, --semantic_ranking
                        active the semantic ranking of the binary search.
```

Figura 2: Ayuda del `SAR_Searcher` (incluye funcionalidades ampliadas).

Las funcionalidades básicas del recuperador de artículos (*searcher*) son las siguientes:

- Sólo tiene un argumento obligatorio, `index`, que es el nombre del fichero que contiene el índice guardado previamente con el programa `SAR_Indexer.py`.
- Tiene cuatro modos de funcionamiento distintos determinados por utilizar los argumentos `-Q`, `-L`, `-T` o no indicar ninguno de ellos:
  - `-Q` permite pasar una consulta directamente en la llamada al programa. Por ejemplo, `python SAR_Searcher.py indice.bin -Q 'python precisión'` , resuelve la consulta `'python precisión'` utilizando para ello el índice guardado en el fichero `indice.bin`, muestra el resultado y finaliza.
  - `-L` recibe una lista de consultas mediante un fichero, muestra el resultado consulta por consulta y finaliza. Por ejemplo, `python SAR_Searcher.py indice.bin -L query_list.txt`, resuelve las consultas contenidas en el fichero `query_list.txt` utilizando el índice guardado en el fichero `indice.bin`.
  - `-T` se utiliza para evaluar el recuperador utilizando una lista de consultas resueltas. Esto permite determinar si el programa funciona correctamente. Cada línea

del fichero de consultas debe tener una consulta y el número de artículos que se deben recuperar para esa consulta; los dos valores (consulta y cardinalidad de la respuesta) deben estar separados por un tabulador. Por ejemplo, `python SAR_Searcher.py indice.bin -T test_list.txt` comprueba si todas las consultas contenidas en `test_list.txt` obtienen el resultado esperado sobre el índice guardado en el fichero `indice.bin`. Si todas las consultas devuelven el número de artículos esperado, se mostrará el mensaje `'Parece que todo ha ido bien, buen trabajo!'`, en caso contrario, se mostrará un mensaje de error resaltando la consulta con el resultado incorrecto.

  - En otro caso, el programa entrará en un bucle de petición de consulta y devolución de artículos relevantes.

- Si la consulta tiene más de un término, implícitamente se debe hacer un AND de las *postings list* de todos los término. Se permite anteponer a un término la palabra NOT, en ese caso, se recuperarán los artículos que no contienen dicho término. El orden de evaluación y prioridad será de izquierda a derecha. Por ejemplo, la consulta `'term1 NOT term2 term3'` deberá devolver los artículos que contienen `term1` AND los que NO contienen `term2` AND los que contienen `term3`. Es decir, los artículos que contienen los términos `term1` y `term3` pero NO `term2`.

  IMPORTANTE: para realizar la intersección (AND) de *postings list* se debe implementar el algoritmos de *merge* vistos en teoría, en el método:

  - `and_posting(self, p1, p2)`

- Cuando se elige la opción `-Q` o el modo interactivo, la presentación de los resultados se realizará en función de los argumentos opcionales con los que se ejecute el programa `python SAR_Searcher.py`:

  - `-C`, muestra la consulta y el número de artículos devuelto por el buscador, como en el modo `-L`.
  - Si no se indica `-C`, se mostrará en una línea por artículo:
    - Un número de orden para numerar los artículos recuperados
    - El identificador del artículo
    - El título del artículo
    - La url.

- Si se incluye el argumento `-A`, el recuperador mostrará todos los artículos recuperados, en caso contrario (la opción por defecto) sólo se mostrarán los 10 primeros.

1.3. Búsqueda posicional

La búsqueda posicional permite la búsqueda de varios términos consecutivos utilizando las dobles comillas. Esto hace necesario el uso de *postings list* posicionales.

Ejemplo de funcionamiento: buscar `"fin de semana"` con la funcionalidad implementada deberá devolver sólo los artículos en los que los tres términos aparecen de forma consecutiva,

mientras que buscar `fin de semana` encontraría todos los artículos en los que aparecen los tres términos sin importar la posición.

Para conseguir este funcionamiento será necesario almacenar en el índice invertido, además de en qué artículos aparece cada termino, en qué posiciones aparecen.

Para el recuperador, será necesario completar el método `get_positionals` para obtener la *postings list* de una secuencia consecutiva de términos.

2. Funcionalidades ampliadas: Búsqueda semántica (hasta 1 punto)

Para obtener la máxima puntuación, además de las funcionalidades básicas, se deben implementar correctamente las 2 funcionalidades extra relacionadas con la búsqueda semántica:

- Recuperar los artículos con mayor similitud semántica con la consulta, independientemente de que contengan o no los términos de búsqueda.
- Ordenar los resultados de una búsqueda binaria en función de la similitud semántica de los artículos con la consulta.

Para llevar a cabo esta ampliación se debe utilizar la librería `SAR_semantics.py`. Esta librería permite crear representaciones vectoriales densas a partir de textos, almacenarlas en una estructura KDTree y posteriormente recuperar aquellos vectores más próximos a una consulta.

Disponemos de cuatro alternativas para generar los embeddings:

- SpacyStaticModel. El embeddings del texto se calcula como la media de los vectores incontextuales de los tokens que lo forman. Se trata e una representación incontextual del texto basada en la librería spacy. Podemos decidir eliminar stopwords o tokens no alfabéticos.
- BetoEmbeddingModel. Utiliza el modelo BETO basado en Transformers para el castellano. El embeddings del texto se calcula como la media de los vectores de la última capa de atención del modelo.
- BetoEmbeddingModel. Utiliza el mismo modelo que el anterior, pero utiliza el embedding conrrespondiente al token [CLS] como embedding del texto.
- SentenceBertEmbeddingModel. Utiliza la salida de pooling de un modelo SentenceBert para generar la representación del texto.

La diferencia entre las cuatro alternativas es la forma de calcular los embeddings del texto, la gestión del KDTree y la estructura que devuelve una consulta es idéntica.

Flujo de trabajo para generar el índice semántico:

- Se deben extraer las frases de cada artículo, utilizando el `sent_tokenize` de la librería `nltk`.

- Todas las frases se almacenan en una lista.
- Una vez analizados todos los artículos se crea el KDTree con el método `fit` del codificador elegido.

Flujo de trabajo para recuperar los documentos utilizando el índice semántico:

- El método `query` devuelve una lista de pares (distancia, índice) con los vectores más cercanos a la query. El primer elemento es la distancia entre el vector de la consulta y el de la frase recuperada. El segundo elemento es el índice de la frase dentro de la lista que contiene todas las frases.
- Sabiendo los índices de las frases de deben recuperar los artículos que contienen esas frases.
- Eliminar artículos repetidos.

Importante:

- Debemos tener una estructura que nos permita enlazar frases y artículos.
- Debemos guardar toda la estructura del índice semántico junto con el otro índice.

Recuerda que las funcionalidades extra sólo se puntuarán si las funcionalidades básicas funcionan correctamente.

3. FAQ

A continuación se da respuesta a algunas preguntas frecuentes. Si surgen más dudas se irán añadiendo en revisiones sucesivas de este documento.

¿Qué se entrega?

El responsable del grupo deberá subir a la tarea de PoliformaT únicamente 2 ficheros:

- `SAR_lib.py`, con las funcionalidades del indexador y buscador implementadas.
- `informe.pdf`, con el informe descriptivo del trabajo realizado. El informe debe incluir como mínimo:
    - Enumeración de los miembros del grupo y descripción de la contribución al proyecto de cada miembro.
    - Enumeración de las funcionalidades extra implementadas y descripción de su implementación.
    - Descripción y justificación de las decisiones de implementación realizadas.

- Descripción del método de coordinación utilizado por los miembros del grupo en la realización del proyecto.
- Podéis añadir toda la información que consideréis importante, incluidas opiniones subjetivas de la marcha del proyecto y las dificultades afrontadas.

Se permiten los reenvíos mientras la tarea esté activa, se evaluará la última entrega.

¿Cómo será la evaluación del proyecto?

Para evaluar el proyecto se tendrán en cuenta múltiples factores:

1. El funcionamiento correcto del indexador y del búscador, tanto de las funcionalidades básicas como de las funcionalidades extras implementadas. Se utilizará la opción de test, argumento `-T` en el buscador, para comprobar que los resultados son los esperados. Para hacer esta evaluación se utilizarán los programas `SAR_Indexer.py` y `SAR_Searcher.py` originales y la última versión de la biblioteca `SAR_lib.py` subida por el responsable del grupo a la tarea de PoliformaT.

2. El código enviado. Se hará una evaluación de la eficiencia y corrección del código desarrollado por el grupo, penalizándose las implementaciones poco eficiente o excesivamente farragosas. Se recomienda añadir comentarios donde se considere oportuno (sin llegar al extremo de comentar las cosas demasiado obvias).

3. La memoria del proyecto. La memoria del proyecto debe ser lo suficientemente descriptiva para entender como se ha realizado el proyecto y ser correcta tanto en contenido como en formato.

4. También se realizará una prueba objetiva individual y por grupo. **Todos** los miembros del grupo **deben ser conocedores del funcionamiento completo de la aplicación**, no sólo de la parte que han implementado.

A título orientativo y asumiendo que la memoria y el código es de calidad y se responde correctamente a la prueba del laboratorio:

- Funcionalidades básicas sin posicionales: 0.5 puntos.
- Funcionalidades básicas con posicionales: 1 punto.
- Cada una de las dos funcionalidad extra, SI todas las funcionalidades básicas funcionan: +0.5 puntos.

¿Hay que procesar los documentos con alguna biblioteca?

Los documentos que se utilizan en el **indexador** y el **buscador** son ficheros de texto donde cada línea es un objeto en formato *json*. En PoliformaT hay disponibles ficheros para poder hacer pruebas controladas.

Parcialmente, la grabación y la carga de los ficheros *json* ya está incluida en la plantilla de la biblioteca `SAR_lib.py`.

¿Se pueden usar operadores de conjuntos (tipo `set` en python) en lugar del algoritmo de intersección de postings lists vistos en teoría?

NO, para intersectar *postings lists* se deben utilizar el algoritmo vistos en teoría, el AND de dos *postings list*. También se debe implementar el NOT de una *postings list* sin utilizar los métodos de conjuntos.
No es por una cuestión de eficiencia, sino para practicar lo visto en la teoría de la asignatura. Además, ten en cuenta que los *postings list* serán posicionales para permitir la búsqueda con comillas.

¿Qué pasa si el número de artículos no coincide con la referencia?

En la mayoría de casos esto es debido a uno de estos dos factores:

- Una implementación errónea de los métodos de indexación o de recuperación de los artículos.
- Una normalización y *tokenización* de los artículos distinta de la utilizada para el cálculo de la referencia.

Además de la biblioteca `SAR_lib.py`, ¿se pueden añadir más ficheros que contengan funciones o clases adicionales?

NO. Eso sí, podéis añadir a la clase `SAR_Project` todos los atributos y/o métodos que consideréis conveniente. Incluso se pueden añadir más clases o funciones pero dentro de la biblioteca, sin modificar los programas de indexación y recuperación de artículos ni añadir más ficheros.
No obstante, si es TOTALMENTE imprescindible añadir otro fichero de código, ponte en contacto con tu profesor de prácticas.