جامعة دمشق
كلية الهندسة المعلوماتية
السنة الخامسة
قسم هندسة البرمجيات ونظم المعلومات

# محرك بحث

## وظيفة مقرر عملي نظم استرجاع المعلومات

**إعداد الطلاب:**

أَحمَدْ خَلْدُون سُلْطَان

حَمْزَة سَمِير عَمّار

رَاما مُحَمّد فِراسْ الرَنَّة

رَنَا عَبْدُ القَادِر الدَهَّان

**بإشراف:**

م. لِينْ قويدر

الجُمعة 2023/06/09

# الفهرس

## توصيف مجموعات البيانات المستخدمة

### المجموعة الأولى (lotte/technology)

وهي عبارة عن مجموعة بيانات تمثل إجابات من المنتديات المتعلقة بالتكنولوجيا، بما في ذلك ( android, apple, askabuntu, electronics, network-engineering, security, server-fault, software-engineering, superuser, unix, and wabapps).

### المجموعة الثانية (beir/quora)

وهي عبارة عن مجموعة بيانات تمثل مجموعة من الأسئلة من مجموعة بيانات Quora.

## توصيف خطوات المشروع وخدماته الأساسية

### توصيف المشروع

تم تقسيم المشروع إلى العديد من الخدمات، ومن أجل كل خدمة يوجد العديد من التوابع منها ما هو عام (Public) والذي يمكن الوصول له من خلال (API) منفصل أو يمكن لباقي الخدمات التواصل مع هذه الخدمة عن طريق هذه التوابع وتقوم هذه الخدمة بتصديره، ومنها ما هو خاص (Private) وهي توابع لا يمكن للخدمات الأخرى الاستفادة منها أو الوصول إليها بشكل مباشر إنما هي عبارة عن توابع مساعدة تستخدمها التوابع العامة من أجل تحقيق الهدف النهائي للخدمة بأفضل طريقة ممكنة، ولمعالجة استعلام ما، يجري تمرير عملية المعالجة عبر هذه الخدمات بالاستفادة من النتائج المخزنة مسبقاً والتي تم القيام بها بشكل (Offline)، حيث أنه في حالة الـ (Offline)، يتم القيام بالعمليات التالية:

1. بناء الفهرس العكسي المثقل (بقيم tfidf) لكل المستندات من أجل كل مجموعة بيانات (technology | quora).
2. بناء فهرس مثقل للمصطلحات الموجودة ضمن الاستعلامات الجاهزة التي توفرها قاعدة البيانات، وذلك من أجل حساب قيم (tfidf) للمصطلحات الواردة ضمن الاستعلام بالاستفادة من قيم المصطلحات المخزنة مسبقاً مع إعطاء قيمة عالية من أجل (idf = 1) في حال لم يرد المصطلح بشكل مسبق ضمن الاستعلامات المخزنة.

ومن أجل الخدمات الأساسية فإن الاستعلام يمر عبر خمس خدمات منها ما هو (Offline) ومنها ما هو (Online) سيجري عرض كل واحدة منها مع شرح مختصر عنها وعن التوابع الموجودة ضمنها بالإضافة لمخطط توضيحي لكيفية التواصل بين عناصر هذه الخدمة أو بين الخدمة وخدمات أخرى.

## خدمات المشروع الأساسية

### معالجة النص

يتم استقبال البيانات المُراد معالجتها كسلسلة نصية (String)، ومن ثم تقسيم السلسلة إلى رموز (Tokens)، وبعد ذلك يجري تحويلها إلى رموز صغيرة (Lowercase)، ومن ثم حذف علامات الترقيم، وإزالة كلمات التوقف (Stop words)، ثم تطبيع التواريخ وأسماء الدول، ومن ثم القيام بعملية التجذير.
وتم اعتماد الخطوات السابقة بعد التجريب والتقييم واعتماد الخطوات ذات النتائج الأفضل، حيث أدى إضافة التصحيح الإملائي، إلى انخفاض تقييم أداء محرك البحث، لذلك بعد التجريب تم اعتماد هذا الترتيب النهائي لعملية المعالجة، وتحتوي هذه الخدمة على التوابع التالية:

a) list <- get_words_tokenize:
مهمة هذا التابع هو تقسيم السلسلة النصية واستخراج الرموز منها ومن ثم إعادة الرموز على شكل مصفوفة.

b) list <- lowercase_tokens:
يتمثل عمل التابع بالقيام بعملية التحويل (conversion) لكل الرموز إلى أحرف صغيرة، ويعد مصفوفة الرموز بعد تحويلها.

c) list <- remove_punctuations:
يقوم هذا التابع بإزالة علامات الترقيم من مصفوفة الرموز التي تم تمريرها إليه، ومن ثم يعيد المصفوفة بعد تنظيفها.

d) list <- filter_tokens:
تتمثل مهمة التابع بإزالة الـ Stop Words، ولكن مع مراعاة مجموعة البيانات المستخدمة حيث في حال كانت مجموعة البيانات هي (Qoura)، فيتم استثناء كلمات السؤال من الإزالة، كون مجموعة البيانات هذه عبارة عن أسئلة وتمثل فيها كلمة السؤال أهمية كبيرة، ويعيد المصفوفة بعد إزالة كلمات التوقف منها.

e) list <- normalize_dates:
يعمل هذا التابع على استخراج التواريخ بمساعدة Regex، ومن ثم تحويل التواريخ المستخرجة إلى صيغة محددة وإعادة دمجها مع المصفوفة بعد حذف الصيغ الغير موحدة.

f) list <- normalize_country_names:
يستخرج هذا التابع اختصارات الدول بالاستعانة بمكتبة pycountry، ويقوم باستبدال كل اختصار بالاسم الكامل للدولة وذلك من أجل توحيد الأسماء الخاصو بالدول.

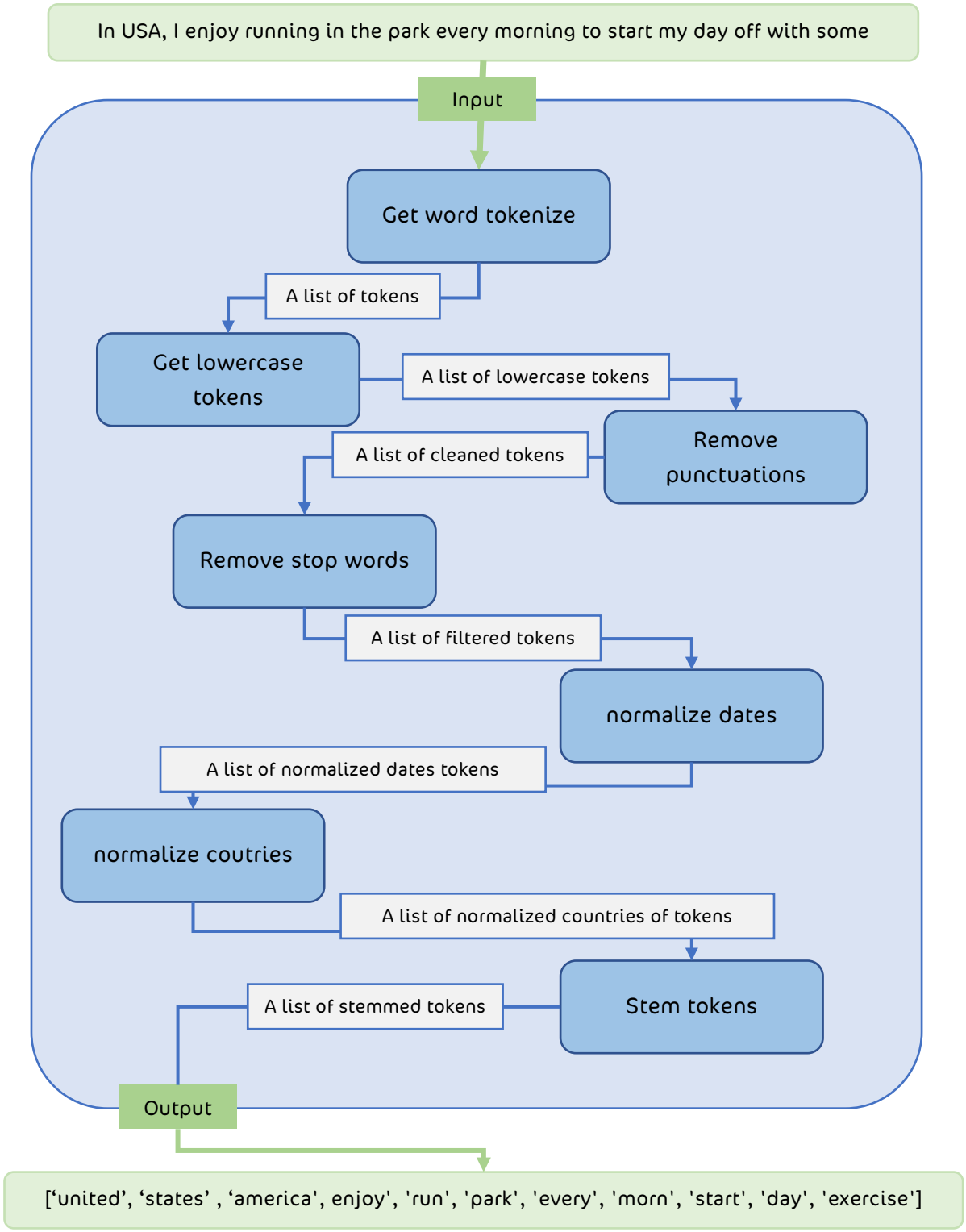
g) list <- stem_tokens:
يقوم التابع بتجذير الرموز وإعادتها إلى أصلها، مما يساعد في تخفيف عدد

الرموز المخزنة أو التي نرغب بمعالجتها، ويعيد مصفوفة الرموز بعد تجذيرها.

h) get_preprocessed_text_terms <- list:

يستقبل هذا التابع السلسة النصية التي نرغب بمعالجتها (من الممكن أن تكون السلسة هي الاستعلام أو المستند)، بالإضافة إلى اسم مجموعة البيانات المستخدمة من أجل تحديد حذف كلمات السؤال أم لا، ومن ثم يقوم بالعمليات السابقة بالترتيب على السلسلة النصية ويعيد المصفوفة الناتجة.

مخطط يوضح طريقة عمل الخدمة



## الفهرسة وتمثيل المستندات

يتم بناء الفهرس العكسي بشكل offline، وعند بداية إقلاع البرنامج يتم جلب الفهرس الذي تم بناءه من قاعدة البيانات وتخزينه ضمن قيم موجودة بالـ runtime وذلك من أجل استخدامه لاحقاً ضمن عملية المطابقة والتصنيف عند معالجة الاستعلام وجلب نتائجه، بالإضافة أيضاً إلى تخزين الـ vectors الخاصة بالمستندات بعد جلبها أيضاً من قاعدة البيانات ضمن قيم في الـ runtime وذلك من أجل سهولة استخدامها لاحقاً وتحقيق السرعة في معالجة الاستعلامات وجلب النتائج، وتحتوي هذه الخدمة على التوابع التالية:

a) get_corpus -> Dict[str:str]:
يقوم بإعداد القاموس الذي سيحوي ids المستندات مع الـ content الخاص بكل مستند، وذلك حسب إما لمجموعة البيانات الأولى (technology)، أو الثانية (quora)، حيث يتم أخذ 100,000 مستند من الـ Testing Data ودمجها مع المستندات الخاصة بالـ qrels.

b) create_unweighted_inverted_index -> Dict[str:list]:
ومهمة هذا التابع هي إعداد فهرس عكسي غير مثقل يربط ما المصطلحات الموجودة بالـ corpus وids المستندات التي تحوي هذه المصطلحات.

c) calculate_tf -> Dict[str:float]:
يقوم بحساب الـ Term frequency من أجل مستند ما، ويعيد قاموس يحتوي على كل مصطلح وعدد مرات تكراره ضمن هذا المستند.

d) calculate_idf -> Dict[str:float]:
يقوم بحساب الـ Inverse document frequency، وذلك بالاعتماد على الـ corpus، والفهرس العكسي الغير مثقل، ويُعيد قاموس يمثل كل مصطلح مقابل قيمة اللوغاريتم العشري لناتج قسمة العدد الإجمالي للمستندات على عدد المستندات التي تحوي هذا المصطلح.

e) calculate_tf_idf -> Dict[str:float]:
يعمل على حساب قيمة الـ tf_idf لمستند ما وذلك بالاستعانة بالتابعين السابقين، حيث يقوم بإعادة قاموس يمثل كل مصطلح ضمن المستند ويقابله قيمة الـ tf_idf لهذا المصطلح والتي يتم حسابها عبر ضرب قيمة الـ tf بقيمة الـ idf.

f) create_docs_vectors -> Dict[str, Dict[str, float]]:
يقوم بحساب الـ tf_idf من أجل كل مستند موجود ضمن الـ corpus، ومن ثم يقوم بتخزين الناتج ضمن قاموس يمثل الـ id الخاصة بالمستند مع الـ tf_idf الخاصة به، وبعد ذلك يقوم القاموس الناتج ضمن db باستخدام الـ shelve.

g) create_weighted_inverted_index -> None:

يعمل هذا التابع على حساب الفهرس العكسي المثقل من أجل مجموعة بيانات ما، حيث يقوم بالاستعانة بالتابع السابق من اجل بناء vector خاص بكل مستند ضمن الـ corpus، ومن ثم يقوم بعملية بناء قاموس يتضمن جميع المصطلحات الموجودة ضمن مجموعة البيانات، ومقابل كل مصطلح يوجد مجموعة قواميس يمثل كل منها id المستند الذي يحتوي المصطلح مقابل قيمة الـ tf_idf له وبعد ذلك يتم تخزين القاموس الناتج ضمن db باستخدام الـ shelve.

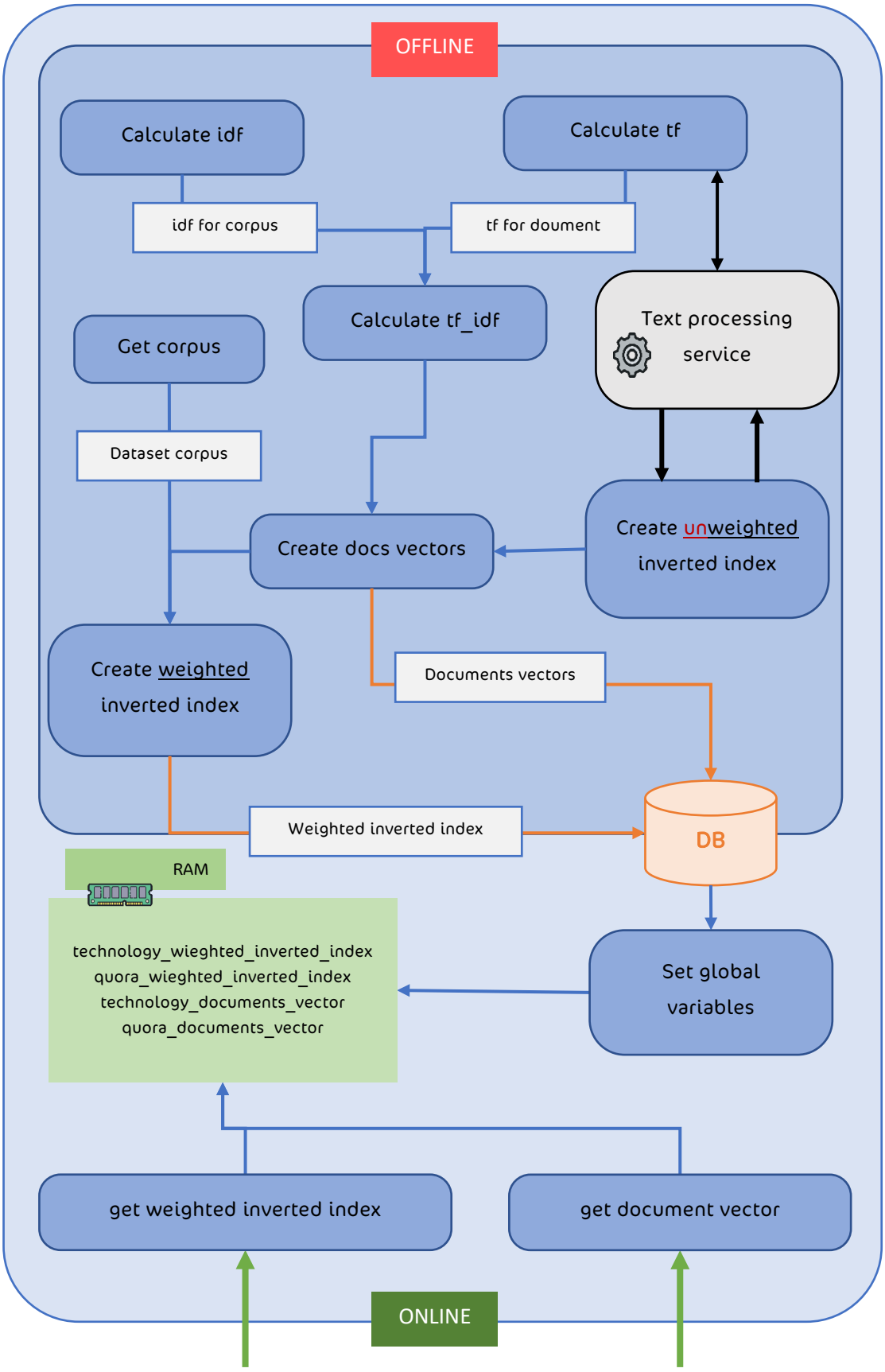
h) get_weighted_inverted_index() -> Dict[str, list]:

يعيد هذا التابع الفهرس العكسي المثقل الذي تم بناءه مسبقاً بشكل offline، حيث يقوم بجلب الفهرس من الـ db باستخدام الـ shelve ومن ثم إعادته.

i) get_document_vector() -> Dict[str, float]:

يعيد هذا التابع الـ vector الخاص بمستند ما، وذلك عن طريق الـ documents vectors التي تم بناءها مسبقاً وتخزينها ضمن الـ db.

مخطط يوضح طريقة عمل الخدمة



## معالجة الاستعلامات وتمثيلها

يتم بناء فهرس عكسي غير مثقل للاستعلامات الموجودة مسبقاً ضمن مجموعة البيانات حيث ويتكون من المصطلحات الموجودة ضمن الاستعلامات ومقابل كل مصطلح يوجد id الاستعلام، وتتم هذه العملية بشكل offline، أما بالنسبة للاستعلام الذي يقوم المستخدم بادخاله، فيتم حساب الـ tf-idf للكل مصطلح ضمن الاستعلام وذلك بالاستعانة بالفهرس العكسي الذي تم بناءه مسبقاً على تعليمات مجموعة البيانات وتتم هذه العملية بشكل online ويكون خرجها عبارة عن قاموس كل عنصر فيه يشير الى مصطلح يقابله قيمة الـ tf_idf للمصطلح، وتحتوي هذه الخدمة على التوابع التالية:

a) get_queries_corpus -> [str:str]:
يقوم بإعداد القاموس الذي سيحوي ids الاستعلامات الموجودة مسبقاً ضمن مجموعة البيانات ومقابل كل id يوجد محتوى هذه التعليمة.

b) create_unweighted_inverted_index -> None:
يقوم ببناء فهرس العكسي للاستعلامات وذلك بالاستعانة بالتابع السابق، حيث يكون الفهرس عبارة عن قاموس يحتوي على المصطلحات الخاصة بالاستعلامات، ومقابل كل مصطلح يوجد سلسلة من ids الاستعلامات التي تحوي هذا المصطلح، وبعد انتهاء عملية البناء يتم تخزين الفهرس ضمن قاعدة البيانات باستخدام shelve.

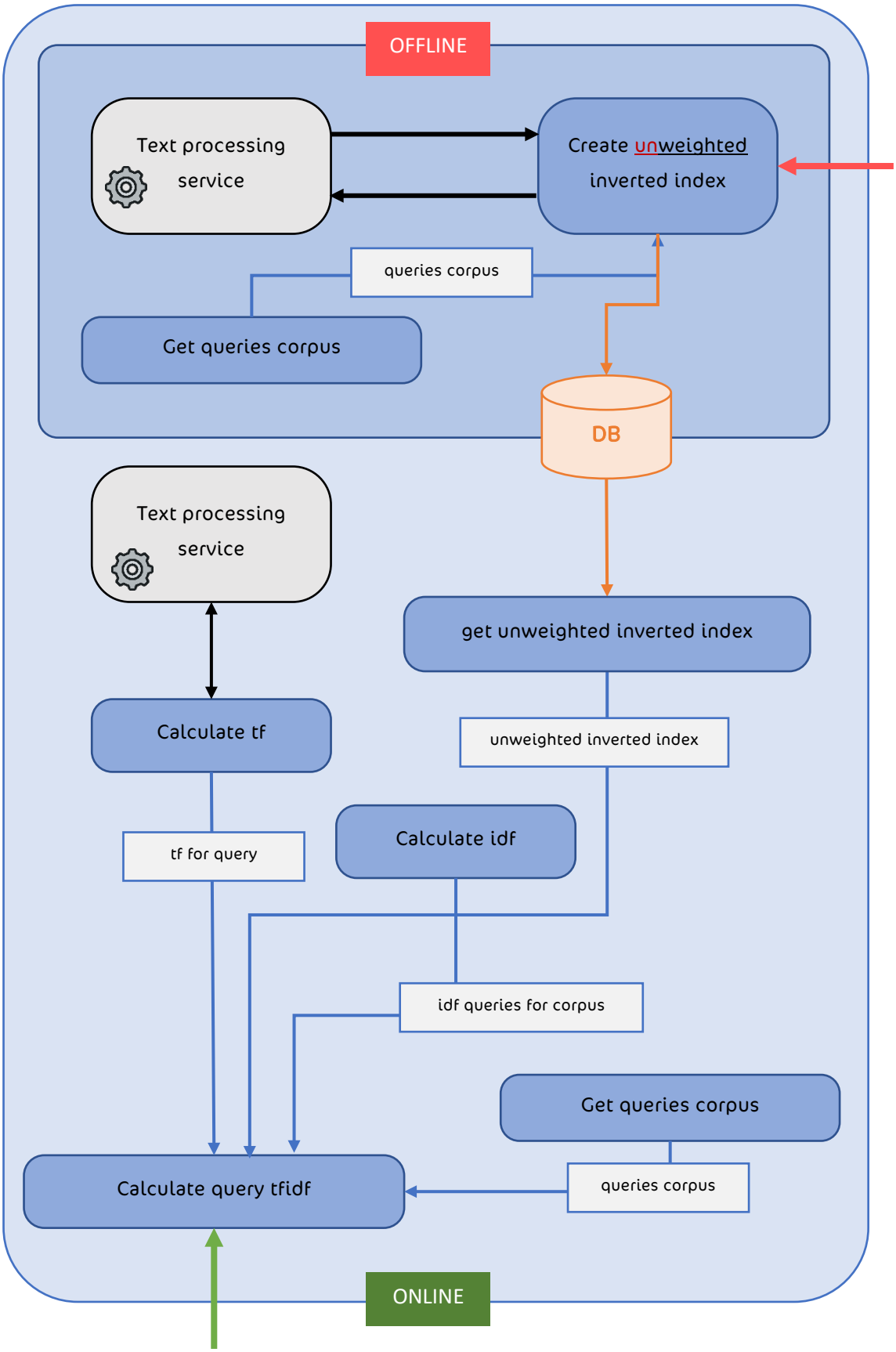
c) get_unweighted_inverted_index -> [str, list]:
يقوم بإعادة الفهرس العكسي الذي تم بناؤه مسبقاً بشكل offline عن طريق قاعدة البيانات التي تم انشاؤها باستخدام shelve.

d) calculate_tf -> [str, float]:
يقوم بحساب قيمة الـ Term Frequency وذلك من أجل تعليمة ما، ويعيد قاموس يحتوي على المصطلحات الموجودة ضمن هذه التعليمة ومن أجل كل مصطلح عدد مرات تكراره.

e) calculate_idf -> [str, float]:
يقوم بحساب قيمة الـ Idf من أجل كل الاستعلامات بالاستعانة بالفهرس العكسي، ويعيد هذا التابع قاموساً يحتوي على المصطلحات ومقابل كل مصطلح يوجد اللوغاريتم العشري لنتيجة قسمة العدد الإجمالي للاستعلامات على عدد الاستعلامات الحاوية لهذا المصطلح.

f) calculate_query_tfidf -> [str, float]:
يقوم بحساب قيمة الـ Idf من أجل كل الاستعلامات بالاستعانة بالفهرس العكسي، ويعيد هذا التابع قاموساً يحتوي على المصطلحات ومقابل كل مصطلح يوجد اللوغاريتم العشري لنتيجة قسمة العدد الإجمالي للاستعلامات على عدد الاستعلامات الحاوية لهذا المصطلح.

مخطط يوضح طريقة عمل الخدمة



## مطابقة الاستعلام وترتيب النتائج

تجري عملية مطابقة الاستعلام في البداية عن طريق مطابقة الـ Query Vector الناتج عن الخدمة السابقة مع الفهرس العكسي المثقل، وتكون نتيجة المطابقة مجموعة تحوي على الـ ids المتعلقة بهذا الاستعلام، بعد ذلك يتم جلب الـ Documents Vector للمستندات الناتجة (وهي عبارة عن قاموس يمثل ids المستندات مقابل كل id يوجد قاموس يحوي المصطلحات وقيمة الـ tfidf لكل مصطلح).

بعد الحصول على مجموعة vectors للمستندات المطابقة للاستعلام، يجري إنشاء مصفوفة مستندات ومصفوفة للاستعلام، الأعمدة فيها تمثل المصطلحات وتقاطع السطر مع العمود يمثل وزن (قيمة tfidf)، لهذا المصطلح ضمن المستند او الاستعلام، ومن ثم حساب درجة التشابه باستخدام الـ Cosine Similarity، والقيام بعملية zip بين درجة التشابه و ids المستندات ضمن قاموس من أجل الحصول على قاموس يمثل كل عنصر فيه id لمستند يقابله درجة التشابه بين هذا المستند والاستعلام المطلوب، ومن ثم يتم ترتيب هذا القاموس وإعادته للقيام بعملية المطابقة ضمن الخدمة القادمة، وتتضمن هذه الخدمة التوابع التالية:

a) get_documents_related_to_query -> [str]:Set
يقوم باسترجاع المستندات الحاوية على المستندات التي وردت ضمن الاستعلام، وذلك عبر مطابقة الفهرس العكسي المثقل الذي تم بناءه خلال مرحلة الفهرسة مع مصطلحات الاستعلام بعد تثقليها بالاستعانة بالخدمة السابقة (معالجة الاستعلامات)، ويعيد مجموعة ids للمستندات المتشابهة.

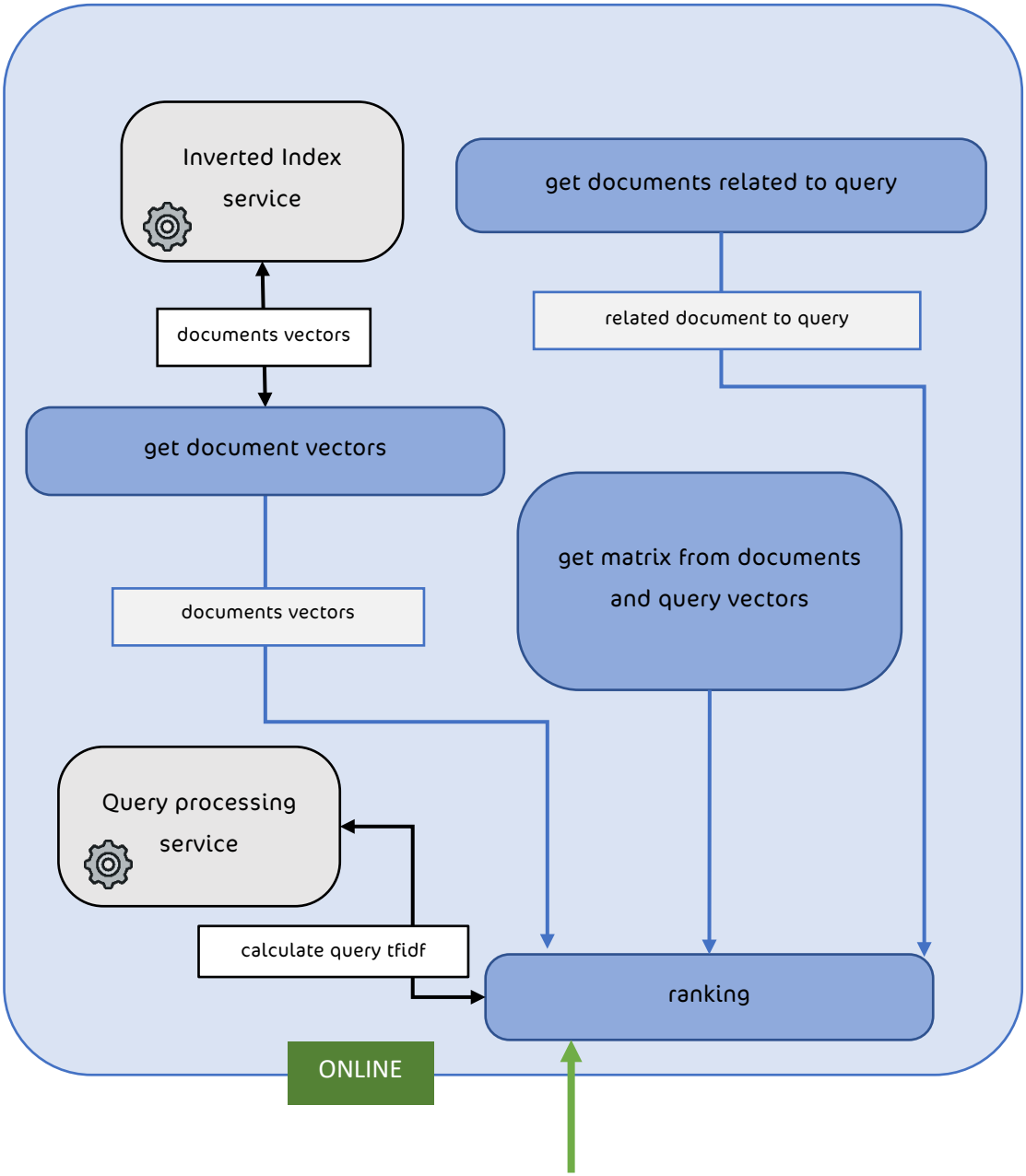
b) get_document_vectors -> [str, Dict[str, float]]:Dict
يقوم بطلب الـ vectors الخاصة بالمستندات المتشابهة وذلك بالاستعانة بخدمة الفهرسة التي تم فيها بناء الـ vectors الخاصة بالمستندات وتخزينها ومن ثم وضعها بالذاكرة خلال التشغيل من أجل سهولة الوصول، ويعيد هذا التابع مجموعة الـ vectors التي حصل عليها.

c) get_matrix_from_documents_and_query_vectors
-> [np.ndarray, np.ndarray, List[str]]:Tuple
مهمة هذا التابع هو بناء المصفوفات اللازمة للقيام بحساب التشابه لاحقاً، ويقوم بتشكيل مصفوفة للمستندات، ومصفوفة للاستعلام، وفي كلتا المصفوفتين تكون الأعمدة ممثلة للمصطلحات وتقاطع السطر مع العمود يمثل وزن هذا المصطلح ضمن المستند او الاستعلام.

d) ranking -> [str, float]:Dict
ويتمثل عمل هذا التابع كتجميع لعمل التوابع السابقة وحساب النتيجة النهائية ومن ثم إعادتها، حيث يقوم بداية بطلب الفهرس العكسي المثقل

من خدمة الفهرسة، وأيضاً يقوم بطلب حساب vector الاستعلام (الذي يمثل قيم tfidf لمصطلحات الاستعلام) وذلك بالاستعانة بخدمة معالجة الاستعلام، ومن ثم يمرر هذه المتحولات للتابع الأول الذي يعيد المستندات المشابهة، وبعد حساب الـ vectors الخاصة بالمستندات باستخدام التابع الثاني، يقوم باستدعاء التابع الثالث من أجل حساب المصفوفات وبعد ذلك يجري العمل على حساب التشابه باستخدام الـ cosine similarity، ومن ثم مطابقة النتائج بين id المستند ودرجة التشابه وترتيب النتيجة بشكل تنازلي (من الأكثر تشابهاً وحتى الأقل تشابهاً) وإعادتها ليتم معالجتها في الخدمة التالية.

مخطط يوضح طريقة عمل الخدمة



### المطابقة النهائية وتشكيل الرد

بعد الحصول على قاموس مرتب بشكل تنازلي حسب الوزن ويتضمن ids المستندات يجري الإعداد النهائي للرد وذلك بمساعدة الـ docs_store الخاص بمجموعة البيانات المستخدمة وذلك من أجل جلب المحتوى الخاص بالمستند عن طريق الـ id الخاص به، وبعد هذه المرحلة يصبح لدينا قاموس كامل من المستندات مرتب حسب الأهمية كل سطر منه يمثل id المستند ويقابله محتوى المستند، ويتم اقتصاص أول مئة مستند (لأن النتيجة تكون كبيرة جداً في بعض الأحيان)، ومن ثم إعادة النتيجة إلى المستخدم، وضمن هذه الخدمة نجد التوابع التالية:

a) docs_store <- get_docs_store:

يقوم بإرجاع object من الـ docs_store حسب مجموعة البيانات المستخدمة.

b) dict[str, namedtuple] <- get_full_docs_content:

يعيد هذا التابع المحتوى الكامل للمستندات بالاستعانة بـ docs_store وذلك حسب الـ id الخاص بكل مستند.

c) dict <- get_ordered_full_docs:

مهمة هذا التابع هو إعادة ترتيب المستندات بنفس الترتيب الذي تم استلامه من تابع الـ ranking، وذلك بسبب اختلاف الترتيب الذي يعيد به الـ docs_store المستندات عن الترتيب الذي تم تمريره له، ويعد ذات القاموس ولكن مرتب حسب نتيجة تابع الترتيب من الخدمة السابقة.

d) dict <- get_sliced_results_template:

إعداد الشكل النهائي للرد، مع اقتطاع النتائج والإبقاء على أول مئة فقط ومن ثم إعادة النتيجة.

e) dict <- get_search_result:

مهمة هذا التابع تتمثل في تجميع نتائج التوابع السابقة، حيث يقوم بداية باستدعاء التوابع السابقة بالترتيب من أجل الحصول على نتيجة البحث النهائية ومن ثم إعادتها إلى المستخدم.

## توصيف بنية النظام العامة

تم استخدام تقنية (Flask) من أجل التعامل مع طلبات البحث الخارجية ومعالجتها، وبشكل أساسي تم تقسيم النظام إلى الخدمات الرئيسية التي تم ذكرها سابقاً، حيث توجد كل خدمة في ملف منفصل، على الشكل التالي:

1. معالجة النصوص: text_preprocessing.py
2. الفهرسة وتمثيل المستندات: inverted_index_factory.py
3. معالجة الاستعلامات وتمثيلها: query_processing.py
4. مطابقة النتائج وترتيبها: matching_and_ranking.py
5. المطابقة النهائية وتشكيل الرد النهائي: search_query_processing.py.

بالإضافة لوجود بعض الملفات المساعدة، على الشكل التالي:

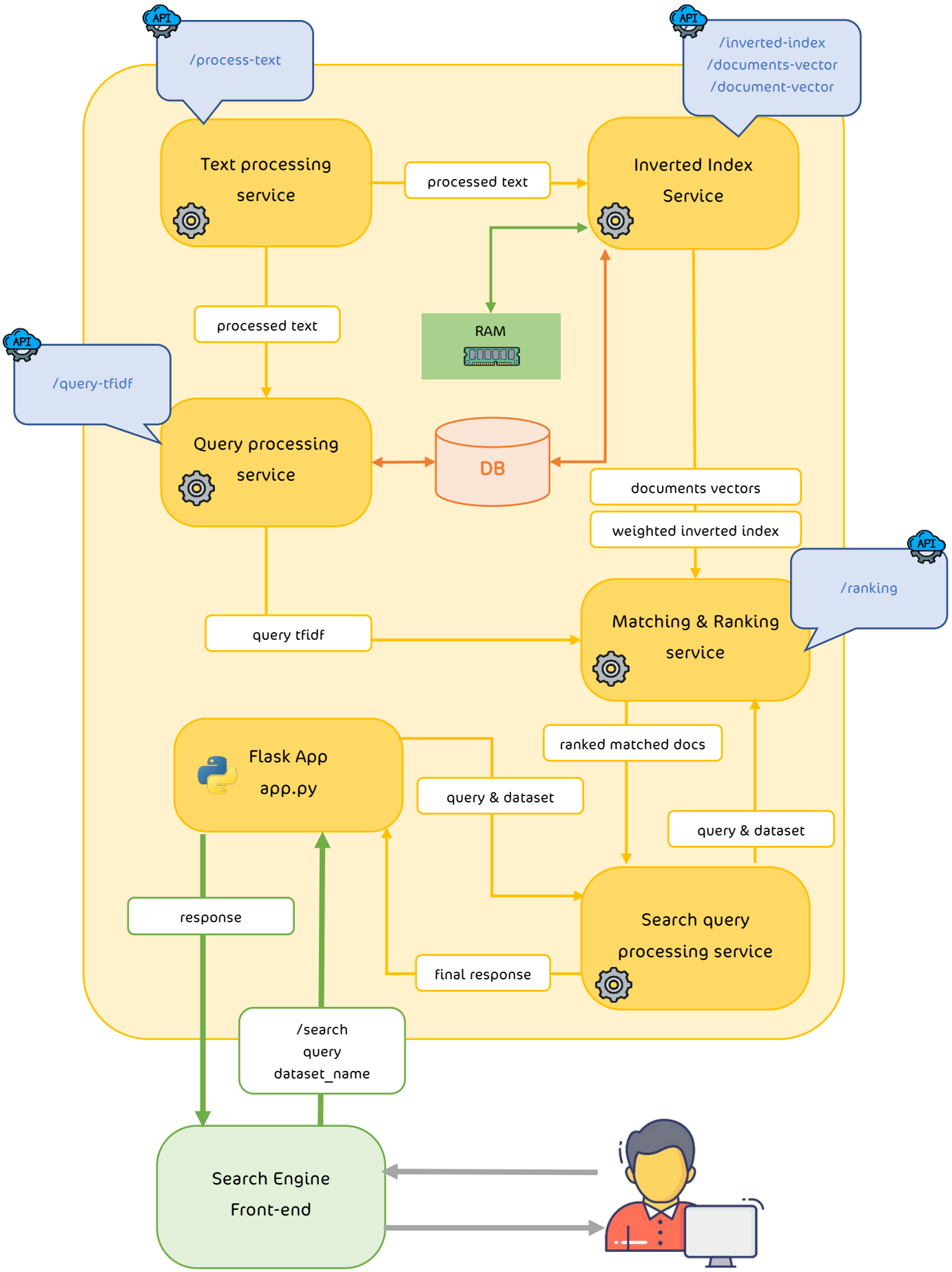
a) inverted_index_factory.py:

يتم استدعاء وتنفيذ هذا الملف بشكل (Offline)، حيث يقوم هذا الملف باستدعاء توابع البناء ضمن خدمة الفهرسة (2) ومعالجة الاستعلامات (3) وذلك من أجل بناء الفهارس بشكل مسبق، وطباعة جملة تفيد بانتهاء التنفيذ بعد اكتمال بناء كل فهرس.

b) app.py:

ضمن هذا الملف تم تعريف نقطة وصول للبحث (Search)، والتي عن طريقها يتم استقبال الاستعلام ومجموعة البيانات المحددة، ومن ثم استدعاء تابع الـ Ranking ضمن خدمة المطابقة والذي بدوره يقوم باستدعاء التوابع اللازمة من الخدمات الأخرى (كل خدمة تقوم باستدعاء ما يتطلب لاكتمال عملها من التوابع الموجودة في الخدمات الأخرى والتي تم تعريفها بشكل (Public) وتصديرها من ملف الخدمة.
بالإضافة للبحث تم تعريف نقاط وصول من أجل كل تابع تم تصديره في كل خدمة من الخدمات التي تم بناؤها ضمن النظام.

مخطط يوضح بنية النظام وطريقة التفاعل بين خدماته



## الطلبات الاضافية

### اقتراح الاستعلامات Query Suggestions

وتهدف هذه الخدمة إلى مساعدة المستخدم في الوصول إلى استعلام أفضل، عبر اقتراح أفضل الاستعلامات المطابقة للاستعلام الذي يقوم بإدخاله، حيث تمت الاستفادة من الاستعلامات الموجودة مسبقاً ضمن كل مجموعة بيانات وتخزينها ضمن قاعدة بيانات، بالإضافة لتخزين كل استعلام يأتي إلى النظام مما يساهم في زيادة عدد الاستعلامات المخزنة واقتراح نتائج أفضل للمستخدم، ومن أجل تحقيق هذه الخدمة تمت إضافة نقطة وصول إضافةً لنقط الوصول السابقة، يقوم محرك البحث بطلبها كل نصف ثانية وذلك من أجل الحصول على اقتراحات ملائمة لما يدخله المستخدم، وتحتوي هذه الخدمة على التوابع التالية:

a) initialize_queries_db <- None:
ويتم استدعاء هذا التابع مرة واحد فقط وبشكل (Offline)، ويقوم بتهيئة قاعدة البيانات الخاصة بالاستعلامات من أجل كل مجموعة بيانات.

b) set_query_refinement_global_variables <- None:
مهمة هذا التابع هي بوضع البيانات المخزنة مسبقاً ضمن قاعدة البيانات الخاصة بالاستعلامات ضمن متحولات تخزن في الذاكرة، وذلك من أجل تحقيق سرعة الوصول والإضافة في حالة النظام (Online).

c) get_query_suggestions -> list:
ويقوم هذا التابع بمطابقة مصطلحات الاستعلام، مع مصطلحات الاستعلامات المخزنة مسبقاً، ومن ثم جلب كل استعلام متطابق مع تحدد التكرار (Frequency) للمصطلح ضمن الاستعلام وذلك من أجل ترتيب الاستعلامات لاحقاً حسب التكرار من أجل الوصول إلى مجموعة اقتراحات من الاستعلامات مرتبة تنازلياً حسب الأكثر تطابق.

d) get_ranked_suggestion <- list:
يرتب هذا التابع مصفوفة الاستعلامات الناتجة من التابع السابق، ويعيد المصفوفة بعد ترتيبها تنازلياً حسب التكرار.

e) get_ranked_query_suggestions -> list:
يستقبل هذا التابع اسم مجموعة البيانات والاستعلام المُراد الحصول على اقتراحات من أجله، ويقوم بالاستدعاءات المناسبة من التوابع السابقة ثم يعيد أول خمس عشرة نتيجة من المصفوفة الناتجة من أجل عرضها للمستخدم.

## Topic Detection

تم بناء هذه الخدمة بالاعتماد على طريقة (Laten Semantic Indexing) LSI والتي تستخدم الأداة الرياضية (Singular value decomposition) SVD وقمنا باستعمال الفهرس العكسي المثقل ومتجهات المستندات إضافة للمتجه الخاص بالاستعلام عوضاً عن التطبيق على المستندات الموجودة ضمن قاعدة البيانات وبهذه الطريقة أصبحنا نستخدم أوزان tfidf التي تم حسابها والمصطلحات التي تم إزالة الضجيج منها، وبشكل أساسي قمنا بتدريب الـ LSI model على مجموعة البيانات الخاصة بنا لإيجاد نسبة كل موضوع (Topic) في كل مستند ليكون قادراً على تمثيل كل مستند كمتجه في فضاء المواضيع(Topic Space) ، بعد ذلك قمنا بتمثيل الاستعلام كمتجه في نفس الفضاء لنقوم بعدها باستخدام مصفوفة التشابه (Similarity Matrix) من أجل إيجاد درجة تشابه الاستعلام مع كل مستند ومن ثم ترتيب المستندات حسب الأكثر تشابه حسب الأكثر تشابه (Similarity) من مواضيع الاستعلام (Query topic) المدخل، وتحتوي هذه الخدمة على التوابع التالية:

a) create_lsi_corpus_and_dictionary

-> corpora.Dictionary, List[List[Tuple[int, float]]]Tuple[:]

مهمة هذا التابع هي تشكيل بيانات المستندات بشكل يتناسب بأن يكون دخلاً للـ LSI Modelالذي نرغب بتدريبيه، فيتم إنشاء Corpora Dict وهو عبارة عن قاموس ضمن مكتبةGensim ، ونقوم بإيجاد قيم token2id وهي عبارة عن Attribute ضمن هذا القاموس وتكون قيمتها عبارة عن كل مصطلح ضمن الفهرس المثقل العكسي ورقم تسلسلي مقابل له ويشكل هذا القاموس الباراميتر الأول من أجل الموديل) مع ملاحظة أنه سيقوم باستخدام الـ id2token وهي فقط معكوس القيم التي قمنا بإنشائها(، وبعدها نقوم بإنشاء الـ Corpus بالاستفادة من القاموس الناتج، حيث يكون هذا الـ corpus عبارة عن مجموعة المستندات كلاً منها موصف كـ bow (bag of words)وهو عبارة عن مجموعة ثنائيات (tuples) كل منها عبارة عن مصطلح وقيمة tfidfالمقابلة له.

b) create_trained_lsi_model -> None:

ويتم استدعاء هذا التابع بشكل(Offline) ، ويقوم بتدريب LSI Model لكل مجموعة بيانات بالاستفادة من التابع السابق، حيث نستفيد من الـ id2token والـ Corpus وقمنا بتحديد عدد الـ Topic بـ 300، وذلك حسب المتعارف عليه بالنسبة لهذا الحجم من البيانات، وتخزين الموديل الناتج ليتم استخدامه بشكل online دون إعادة التدريب،

ويستخدم الموديل Singular Value Decomposition (SVD) ليقوم بتحديد نسبة كل topic في كل مستند، نقوم بتمرير الـ Corpus والذي هو بشكل bow لكل مستند الذي قمنا بتشكيه مسبقا إلى الـ LSI model ليقوم الموديل بتمثيل كل مستند كمتجه في الـtopic space ، الآن نقوم بتصنيف المستندات المتشابهة مع بعضها التي تتشابه نسب الـ topics بينها باستخدام طريقة الـ cosine similarity بين كل زوج من المستندات وذلك باستخدام التابع MatrixSimilarity وهو يقوم بتشكيل مصفوفة التشابه (similarity matrix) والتي سيتم استخدامها لإيجاد تشابه (similarity) الاستعلام مع المستندات فنقوم بتخزينها لاستخدامها online دون إعادة حسابها في كل مرة يرِد فيها استعلام. الآن قمنا بالانتهاء من تجهيز المستندات ضمن الـ topic space

c) compute_similarity_and_rank_documents -> Dict[str,float]:

يقوم هذا التابع بإيجاد التشابه بين الاستعلام والمستندات لإعادة المستندات التي تشابه الاستعلام في نسب الـ topics الموجودة فيه وسيتم استخدامه في التابع التالي، البارامتر الأول لهذا التابع هو مصفوفة التشابه بين المستندات (similarity matrix)التي تم حسابها offline والثاني هو الاستعلام ممثل كمتجه في الـ topic space، يقوم التابع باستخدام مصفوفة التشابه لإيجاد التشابه سريعا بين الاستعلام وكل مستند ثم يقوم بترتيب المستندات حسب الأكثر تشابها ترتيباً تنازلياَ ويعيد المستندات مرتبة.

d) ranking -> Dict[str,float]:

يقوم هذا التابع بتمثيل الاستعلام بنفس شكل التمثيل الذي مثلت فيه المستندات وهو bow وقمنا باستخدام القيم المحسوبة مسبقا للاستعلام والتي تحوي كل مصطلح من الاستعلام مع الوزن tfidf الخاص به ثم باستخدام الـ LSI model يتم تمثيل الاستعلام بشكل متجه في الـ topic space ثم يتم إرسال متجه الاستعلام مع مصفوفة التشابه التي يتم تحميلها إلى الذاكرة من الملف التي التابع السابق

## Crawling

الهدف من هذه الخدمة هو البحث ضمن الروابط الموجودة في كل مستند وإضافة محتوى هذه الروابط إلى المستندات، ولكن لا يتم القيام بالعملية بشكل عشوائي بل يجري التحقق بدايةً من الرابط قبل فتحه ليحقق شرط أنه يعيد محتوى نصي ليس binary،لذلك يجب أن يحقق أحد الشروط الآتية: إما يجب أن يكون للresource ضمن الURL لاحقة لملف نصي مثل (... ,txt ,html)، أو يكون الـ Content-Type الذي يعود من response يشير أنه أحد أنواع النوع أي يتضمن "text/*"، وفي حال لم تتحقق أي من هذه الشروط، وللاختبار ألا يكون الرابط أيضاً هو رابط تحميل لملف، يتم تحميل أول 1024 بايت من البيانات وفحص المحارف في حال كانت Binary أم لا، وعند مطابقة أحد الشروط والتأكد من أن محتوى الرابط هو نصي، يقوم الcrawler بأخذ محتوى الصفحة التي تم إجراء crawling عليها ويعيدها ليتم تطبيق عملية معالجة النص ذاتها التي يتم تطبيقها على النصوص والاستعلامات، ومن ثم إضافة المحتوى إلى النص، وتتم هذه العملية خلال بناء الفهرس العكسي المثقل عبر التحقق من محتوى كل مستند واستخراج البيانات ان وجدت، وكما أن هذه الميزة استلزمت إضافة خطوة جديدة ضمن الtext preprocessing وذلك لفلترة الterms التي تحتوي محارف من لغة مختلفة عن اللغة الإنكليزية وأرقامها، تحتوي هذه الخدمة على التوابع التالية:

a) list <- extractURLs:
مهمة هذا التابع استخراج الروابط من النص ومن ثم إعادتها.

b) boolean <- is_text_url:
يتحقق هذا التابع من كون محتوى الرابط نصي أما لا، وذلك عن طريق التحقق من الـ Content-Type ضمن الـ Headers، إضافة للتحقق من أنه ليس رابط لتحميل ملف ما، أو أن اللاحقة الخاصة به هي من اللواحق التي تعد لملفات النصية، وفي حالة فشل كل ما سبق يتم تحميل أول 1024 بايت والتحقق من المحتوى أنه ليس binary ومن ثم رد النتيجة إما بقبول هذا الرابط أو لا.

c) str <- crawl:
يقوم هذا التابع بالدخول إلى الرابط المرسل إليه ومن ثم الحصول على النص الموجود داخل الصفحة وإعادته.

d) str <- expand_document_with_crawled_data:
ويقوم هذا التابع بتجميع كل التوابع السابقة حيث أنه بداية يتحقق من وجود روابط ضمن المستند أم لا بالاستعانة بالتابع الأول، ومن في حال وجود روابط يقوم باستخراجها بعد التحقق من محتواها بالاستعانة بالتابع الثاني والثالث وبعد ذلك يعيد المحتوى الذي حصل عليه ويقوم بدمجه مع محتوى الـ document الأصلي.

## النتائج والتقييم

### أولاً: نتائج مرحلية

قبل استعراض النتائج النهائية للنظام سنقوم باستعراض بعض نتائج التقييمات الجزئية المطبقة على جزء من ال dataset والتي من خلالها تم اختيار الخطوات والطرائق المناسبة لل text processing/query vectorization وذلك بما يلائم طبيعة مجموعة البيانات التي تم اختيارها.

1. نتائج عدة تجارب في خطوات ال text processing:

بدايةً سنستعرض نتائج تقييم النظام مع تطبيق الخطوات الآتية في معالجة النص text processing:

1. Tokenization
2. Punctuation's removal
3. Stop words filtering
4. Dates normalization
5. Country names normalization
6. Spell checking and correction
7. Lower case conversion
8. lemmatization

| Dataset / Metric | P@10 | recall | MAP | MRR |
|---|---|---|---|---|
| technology/search | 0.11761744966442969 | 0.8674839016925631 | 0.3112619229517189 | 0.14594212613280144 |
| beir/quora | 0.18199999999999897 | 0.9979302944862155 | 0.8170962932346364 | 0.6086079780760248 |

تمت ملاحظة أن خطوة ال spell checking تسبب مشكلة في تصحيح بعض التعابير/الاختصارات الخاصة بمجال التكنلوجيا، وبما أن طبيعة ال dataset الأولى تحتوي على الكثير من هذه التعابير فتم تجريب الاستغناء عن خطوة ال spell checking لنحصل على النتائج الآتية:

| Dataset / Metric | P@10 | recall | MAP | MRR |
|---|---|---|---|---|
| technology/search | 0.1604026845637584 | 0.9188824448906029 | 0.4352648992904729 | 0.2012880087086564 |
| beir/quora | 0.1904999999999989 | 0.9985761278195487 | 0.8636418397286204 | 0.6465743019562987 |

نلاحظ تحسن ملحوظ في النتائج، على الرغم من أن خطوة الـ spell checking قد تكون مفيدة في حالات تصحيح الأخطاء الإملائية المدخلة من قبل استعلامات المستخدم إلا أن تأثير إضافتها كان سلبياً على نتائج تقييم النظام وذلك بسبب طبيعة مصطلحات الـ dataset التخصصية، لذلك تم اختيار الاستغناء عن هذه الخطوة.

ونهاية تم تجريب استبدال عملية الـ lemmatization بـ stemming وتم الحصول على نتائج أفضل كالتالي:

| Dataset / Metric | P@10 | recall | MAP | MRR |
|---|---|---|---|---|
| technology/search | 0.1734899328859060 | 0.9499356956846542 | 0.4770653117212763 | 0.2234917482167527 |
| beir/quora | 0.1964999999999881 | 0.9989317042606516 | 0.8912133978764623 | 0.6683602222202664 |

وبذلك تم اعتماد الخطوات النهائية لـ text processing لتكون:

1. tokenization
2. lower case conversion
3. punctuations removal
4. stop words removal
5. dates normalization
6. country names normalization
7. stemming

أما فيما يخص خطوة إزالة الـ stop words نميز فيها حالتين حسب الـ dataset المستعلم عنها، ففي حال كانت الـ dataset هي quora وبما أن طبيعة الـ documnets الموجودة فيها عبارة عن أسئلة، لذلك قمنا بإزالة قائمة من كلمات الأسئلة (e.g: what, where, when.....) من قائمة الـ stop words وذلك لكون هذه الكلمات مهمة ضمن هذه الـ dataset.

2. نتائج عدة تجارب في طريقة إنشاء الـ query vector:

- نتائج أول طريقة: حساب الـ query vector عن طريق حساب الـ TF_IDF وذلك بالاعتماد على الـ queries الملحقة بالـ dataset (الطريقة المستخدمة بالتجارب الماضية):

| Dataset / Metric | P@10 | recall | MAP | MRR |
|---|---|---|---|---|
| technology/search | 0.17348993288590603 | 0.9499356956846542 | 0.47706531175212763 | 0.2234917482167527 |
| beir/quora | 0.19649999999999881 | 0.9989317042606516 | 0.8912133978764623 | 0.668360222202664 |

- نتائج ثاني طريقة: حساب الـ query vector عن طريق حساب الـ TF_IDF وذلك بالاعتماد على الـ documents الموجودة في الـ dataset:

| Dataset / Metric | P@10 | recall | MAP | MRR |
|---|---|---|---|---|
| technology/search | 0.16476510067114095 | 0.9499356956846542 | 0.4408671915705113 | 0.20935011057335803 |
| beir/quora | 0.19649999999999884 | 0.9989317042606516 | 0.8870073644329981 | 0.6649401111750254 |

- نتائج ثالث طريقة: حساب الـ query vector عن طريق حساب ال TF:

| Dataset / Metric | P@10 | recall | MAP | MRR |
|---|---|---|---|---|
| technology/search | 0.16526845637583903 | 0.9499356956846542 | 0.45669684022459334 | 0.2100530758638375 |
| beir/quora | 0.1933333333333321 | 0.9989317042606516 | 0.8829837472986823 | 0.6634916595193862 |

نجد أن الطريقة الموافقة للتقييم الأعلى هي الطريقة الأولى لذلك هي الطريقة التي تم اعتمادها.

- تم تجريب العديد من قوانين الـ vectors similarities و قوانين حساب الTF_IDF وتم الاعتماد نهايةً على الطرق التي أعطت تقييم أعلى، والتي هي: cosine similarity، وكما تم حساب الTF_IDF بالشكل التالي:

$$tf\_idf_{t,d} = tf_{t,d} * idf_t$$

$$tf_{t,d} = row\ term\ frequency$$

$$idf_t = log_{10}(\frac{N\ documents}{df_t})$$

## ثانيا: نتائج النظام كاملاً

### نتائج الأداء النظام الكاملة (على DATASET كاملةً)

| Dataset / Metric | P@10 | recall | MAP | MRR |
|---|---|---|---|---|
| technology/search | 0.11275167785234921 | 0.9499356956846542 | 0.2920115293460843 | 0.4170565236946883 |
| technology/forum | 0.12714570858283306 | 0.9172200085895893 | 0.1644351214385608 | 0.3703081238427964 |
| beir/quora | 0.15433333333333835 | 0.9992998329156224 | 0.7954364519874446 | 0.8479536165845210 |

### نتائج الأداء النظام الكاملة مع الميزة الإضافية الخاصة بال TOPIC DETECTION

| Dataset / Metric | P@10 | recall | MAP | MRR |
|---|---|---|---|---|
| technology/search | 0.10805369127516798 | 1.0 | 0.2581655234963904 | 0.3625404488044011 |
| technology/forum | 0.12430139720 | 1.0 | 0.1660616964836923 | 0.3467139965604779 7 |
| beir/quora | 0.12300000000000014 | 1.0 | 0.6090461401335018 | 0.6772857618161856 |

نلاحظ أن النتائج قد تحسنت بالنسبة لل recall ولكنّ باقي ال metrics ساءت بشكل بسيط، يرجح السبب بأن الـ trained model الخاص بعملية الـ topic detection لم يتم تدريبه على عدد كافي من المستندات وبالتالي لم يكتسب القدر الكافي من المعرفة ليتمكن من تمييز الtopics الخاصة بالqueries

### نتائج الأداء النظام الكاملة مع الميزة الإضافية الخاصة بالـ CRAWLING

يجدر الذكر بأن عملية الـ crawling يمكن تحقيقها فقط على أول dataset وذلك لخلوّ الـ dataset الثانية من أي روابط، كما أنه لم يتم تقييم أداء النظام مع هذه الميزة الإضافية على كامل الـ dataset وذلك لأن عملية الـ crawling ستستغرق الوقت الطويل لوجود الكثير من الروابط المتواجدة في المستندات وكما أن عملية الـ crawling ستؤدي إلى زيادة كبيرة في حجم المستند الـ crawled لذلك تم الاقتصار على تجريب أداء هذه الميزة على 17,000 doc يحتوون على قرابة 2500 رابط وكانت النتائج كما يلي:

- أداء النظام على 17000 مستند قبل الميزة:

| Dataset / Metric | P@10 | recall | MAP | MRR |
|---|---|---|---|---|
| technology/search | 0.17399328859060403 | 0.9499356956846542 | 0.480827915001817 | 0.6170499830862066 |
| technology/forum | 0.20404191616766612 | 0.9171368415902545 | 0.27955774912754117 | 0.5393996908360522 |

- أداء النظام على 17000 مستند بعد الميزة:

| Dataset / Metric | P@10 | recall | MAP | MRR |
|---|---|---|---|---|
| technology/search | 0.17365771812080538 | 0.9499935526622058 | 0.4794866389104406 | 0.6145816896271341 |
| technology/forum | 0.20309381 2375251 | 0.91795624 00892939 | 0.27751597 89850543 | 0.5376063284692265 |

نلاحظ أن التقييم قد انخفض بنسبة بسيطة ويرجح السبب إلى ما يلي: قد لا يتم أخذ بعين الاعتبار محتوى الروابط المذكورة في المستندات أثناء إنشاء الـ qrels ولذلك قد ينجم عن ذلك عدم وجود qrel تؤيّد ارتباط المستند الذي يحتوي على محتوى crawled لأي query، وفي هذه الحالة سيكون التأثير الإيجابي على النتائج غير ممكناً. أما بالنسبة للتأثير السلبي فإنّ أحد الأسباب المحتملة له هو وجود noise (داتا ليس لها صلة بالمستند) في crawled data.

## تقسيم العمل

- أحمد سلطان: مرحلة التقييم.
- حمزة عمار: مرحلة معالجة النص text processing والطلب الإضافي query refinement.
- راما الرنة: مرحلة query matching & ranking والطلب الإضافي topic detection.
- رنا الدهان: مرحلة indexing،document/query representation والطلب الإضافي crawling.

## المصادر


- Course’s slides and labs.
- https://ir-datasets.com/index.html
- https://colab.research.google.com/github/allenai/ir_datasets/blob/master/examples/ir_datasets.ipynb
- https://www.geeksforgeeks.org/latent-semantic-analysis/
- https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50
- https://www.baeldung.com/cs/euclidean-distance-vs-cosine-similarity#:~:text=The%20Euclidean%20distance%20corresponds%20to,the%20product%20of%20their%20magnitudes
- https://www.crummy.com/software/BeautifulSoup/bs4/doc/