Information Retrieval

10672344

Smart Document Retrieval System:

Prof. Hamed Abd AlHaq

Zaina Abd Alhaq
Basmala Shtayeh
Brooj Edili
Mayar Basheer

1. Introduction:

Traditional information retrieval systems mainly depend on keyword matching to retrieve documents. While this approach works for basic text search, it becomes limited when users want to search for documents related to a specific **time period** or **geographical location**. In many real-world cases, such as news analysis, users are interested in events that happened *when* and *where*, not only in documents that contain certain words.

This project addresses this limitation by building a **Smart Document Retrieval System** that supports **spatio-temporal information retrieval**. The system is designed to retrieve documents based on textual content while also considering **temporal information** (such as dates and time expressions) and **spatial information** (such as cities and countries).

The dataset used in this project consists of **Reuters news articles**, which naturally contain references to events, locations, and dates. These characteristics make the dataset suitable for applying spatio-temporal retrieval techniques. During preprocessing, the system extracts temporal expressions and georeferenced place names from the documents and stores them in structured fields within an Elasticsearch index.

The goal of this project is to design and implement a retrieval system that:

I. understands the importance of time and location in document retrieval,
II. combines textual, temporal, and spatial information in the search process,
III. and returns more relevant and meaningful results compared to traditional keyword-based search systems.

By integrating spatio-temporal processing with modern search and ranking techniques, the system demonstrates a clear understanding of spatio-temporal information retrieval concepts and their practical application.

2. System Architecture:

The Smart Document Retrieval System is built using a multi-component architecture:

2.1 Frontend Layer:

I. **Technology**: Vanilla JavaScript with Leaflet.js for interactive map visualization
II. **Features**: Real-time autocomplete, spatial filtering interface, result visualization on maps

III. **Communication**: RESTful API calls to the FastAPI backend

2.2 Backend Layer

I. **Technology**: FastAPI (Python web framework)
II. **Responsibilities**: Query processing, request validation, semantic embedding generation
III. **API Endpoints**: /autocomplete, /search, /spatiotemporal, analytics endpoints

2.3 Search Engine Layer

I. **Technology**: Elasticsearch
II. **Index Name**: smart-docs-ir
III. **Features**: Full-text search, geospatial queries, temporal filtering, nested field aggregations

2.4 Preprocessing Pipeline

I. **NLP Engine**: spaCy (en_core_web_lg model) for named entity recognition
II. **Geocoding**: geopy with Nominatim service for place name resolution
III. **Semantic Embeddings**: sentence-transformers (all-MiniLM-L6-v2 model)
IV. **Temporal Parsing**: dateparser library for temporal expression normalization

3. Elasticsearch Index Mapping and Settings:

To support advanced textual, temporal, and spatial search, a custom Elasticsearch index was designed with carefully defined **mappings** and **analysis settings**. The goal of this design is to ensure correct indexing, efficient retrieval, and proper handling of different data types such as text, dates, locations, and embeddings.

3.1 Index Settings and Text Analysis:

Custom analyzers were configured to process document text according to the project requirements.

For the **content field**, the system removes HTML tags, eliminates English stop words, filters out very short tokens (less than 3 characters), and applies stemming. This helps reduce noise and improves retrieval quality by focusing on meaningful terms.

For the **title field**, a special autocomplete analyzer based on edge n-grams is used. This analyzer generates n-grams of lengths 3 to 20 characters, allowing the system to support autocomplete suggestions after the third character. A separate search analyzer is defined

to ensure correct matching during retrieval by treating the query as a complete term rather than generating n-grams from it.

These settings ensure that:

I. irrelevant tokens are removed,
II. words are normalized through stemming,
III. and autocomplete is efficient and responsive.

3.2 Field Mappings:

Each document is indexed using structured and well-justified field types:

1. **Title** is indexed as text with autocomplete support and also stored as a raw keyword field (`title.raw`) for exact matching and boosting.
2. **Content** is indexed as analyzed text for full-text search using the custom content analyzer.
3. **Authors** are stored as a nested field, where each author has a first name, last name, and email.
4. **Date** is stored as a date field to allow range queries and temporal filtering.
5. **Geopoint** is stored using the `geo_point` type, which enables distance-based spatial queries and decay functions.
6. **Temporal Expressions** are stored as a nested field, including the original text, normalized date, source (dateline/title/content), whether the expression has an explicit year, whether it is relative, and a confidence score.
7. **Georeferences** are stored as a nested field containing place names, normalized keys for aggregation, country codes, and confidence scores.
8. **Country keys** are stored as keyword fields to support efficient aggregation and country-based analytics.
9. **Content embeddings** are stored using a dense vector field (384 dimensions) with cosine similarity for semantic search.
10. **Approximation metadata** includes: `date_is_approx` (True when date was inferred from temporal expressions), `geopoint_is_approx` (True when location was geocoded from extracted places), and `geopoint_from` (the place name used for geocoding).

3.3 Design Justification:

The chosen mappings ensure that each type of data is stored in a form that best supports its intended use. Text fields are optimized for search and autocomplete, temporal and spatial fields enable filtering and ranking, and nested structures preserve the relationship between extracted entities and their attributes. This design allows the system to combine lexical, semantic, temporal, and spatial retrieval in a consistent and efficient way.

4. Document Indexing Pipeline:

This section describes how documents are prepared and indexed into Elasticsearch. The indexing process starts **after** documents have already been parsed and preprocessed, and it focuses only on inserting valid documents into the Elasticsearch index.

4.1 Input Data for Indexing:

The indexing pipeline reads documents from a directory that contains **JSONL files**. Each JSONL file contains documents that were previously processed and enriched with temporal and geographical information. The index name used in the system is smart-docs-ir.

The pipeline does not modify document structure during this stage. It only reads documents and prepares them for insertion into Elasticsearch.

4.2 Document Validation:

Before a document is indexed, the pipeline applies **explicit validation rules** defined in the code:

I. The document **must have an id field**.
II. The document **must have non-empty content** after trimming whitespace.

If a document does not satisfy these conditions, it is skipped and not indexed. These same rules are also used when counting how many documents will be indexed.

4.3 Semantic Embedding Generation:

For every valid document, a semantic embedding vector is generated. The embedding is computed **only from the document content** using a pre-trained Sentence Transformer model (all-MiniLM-L6-v2). The resulting 384-dimensional vector is L2-normalized to enable accurate cosine similarity computation during semantic search, ensuring scores remain in a consistent [0,1] range.

This embedding is added to the document as the content_embedding field **just before indexing**, and it is later used to support semantic similarity during search.

4.4 Bulk Indexing Process:

Documents are indexed into Elasticsearch using the **bulk indexing API**. Each valid document is sent as a bulk action with:

I. the index name (smart-docs-ir),
II. the document ID,
III. and the full document content including the generated embedding.

Progress tracking is implemented using the tqdm library, which updates once per indexed document. After the process finishes, the total number of indexed and failed documents is printed.

5. Temporal Expression Extraction:

This section describes how temporal information is extracted and processed from the documents. Temporal extraction is implemented during the preprocessing stage and is applied to every document before indexing.

5.1 Temporal Extraction Sources:

Temporal expressions are extracted from three specific parts of each document:

I. the **dateline**,
II. the **title**,
III. and the **content**.

These sources are processed separately, and the origin of each temporal expression is stored to indicate where it was found.

5.2 Detection of Temporal Expressions

The system uses the spaCy NLP model to identify entities labeled as **DATE**. Only valid date candidates are kept. Very short numeric tokens or ambiguous values that do not represent real dates are ignored through explicit validation checks.

Each detected temporal expression is stored with its original text and its position within the document.

5.3 Temporal Normalization:

I. Each extracted temporal expression is normalized into a standard date format **(YYYY-MM-DD)**.
II. Normalization follows explicit rules implemented in the code:
III. **Decade expressions** (e.g., "1990s") are normalized to the first year of the decade (1990-01-01).
IV. If a temporal expression contains an explicit year, it is normalized directly.

V. If the expression does **not** contain a year and is **not relative**, the year is inherited from a reference date.
VI. **Relative expressions** (such as "today" or "last year") are detected using pattern matching and marked with a flag.
VII. **Month-only expressions** are normalized to the first day of that month (e.g., "February" becomes YYYY-02-01) to avoid inheriting inappropriate day values from the reference date.
VIII. **Year-only expressions** are normalized to the first day of the year (YYYY-01-01).
IX. A fixed reference year (**1987**) is used when no publication date is available, which matches the temporal context of the Reuters dataset.

5.4 Temporal Metadata and Confidence:

For each temporal expression, additional metadata is stored:

I. whether the expression contains an explicit year,
II. whether it is relative,
III. its source (dateline, title, or content),
IV. and a confidence score.

The confidence score is computed based on:

I. the presence of an explicit year (+2.0),
II. the source where it appears (+1.5 for dateline, +0.5 for others),
III. whether the expression is relative (-1.0 penalty),
IV. and the position of the expression in the document (decay from 1.0 to 0.0).

The score is clamped to the range [0.0, 5.0] to ensure consistency across documents.

5.5 Date Approximation:

If a document does not contain an explicit publication date, the system attempts to **approximate** a document-level date. This approximation is chosen from the extracted temporal expressions using the following strategy:

I. Non-relative expressions are preferred over relative ones.
II. Among the preferred pool, the expression with the highest confidence score is selected.
III. The normalized date from this expression becomes the document's date field.

If no suitable temporal expression exists, the document date remains undefined (null).

6. Geographical Reference Extraction:

This section describes how geographical information is extracted and processed from the documents. Geographical processing is implemented during the preprocessing stage and is applied before documents are indexed into Elasticsearch.

6.1 Sources of Geographical Information:

Geographical references are extracted from multiple explicit sources in the document:

I. the **dateline**,
II. the **title**,
III. the **content**,
IV. and the predefined **places** field provided in the Reuters dataset.

All these sources are combined to form a list of candidate geographical references for each document.

6.2 Named Entity Recognition for Places:

The system uses the spaCy NLP model to identify entities labeled as **GPE** (geopolitical entities), **LOC** (locations), or **FAC** (facilities). These labels correspond to places that can potentially be geocoded.

Extracted place names are cleaned through multiple steps:

I. Removal of month suffixes (e.g., "London, Jan" becomes "London")
II. Normalization of whitespace and newlines
III. Removal of corporate suffixes (e.g., "Inc.", "Corp.", "Ltd.")

Place candidates that are clearly invalid are filtered out. This includes:

I. Product codes or model numbers (e.g., "XJ-6")
II. Strings containing multiple digits with connector symbols
III. Tokens without any alphabetic characters

Duplicate place names are removed while preserving unique geographical references per document.

6.3 Dateline-Based Location Handling:

If a dateline is present in the document, the system attempts to extract a location name from it using pattern matching (looking for capitalized text before a comma).

The dateline location is treated with higher importance compared to other extracted places. When possible, the dateline location is combined with an additional regional hint from other extracted places to improve geocoding accuracy (e.g., "London, United Kingdom" rather than just "London").

6.4 Geocoding and Geopoint Selection:

To assign a geographical coordinate to a document, the system attempts to geocode extracted place names using the geopy library with the Nominatim geocoding service.

Geocoding follows a priority-based strategy:

I. Dateline locations are attempted first (with regional hints if available).
II. If dateline geocoding fails or no dateline exists, other extracted places are tried in order of confidence.
III. For each place, the system may append a country hint (derived from the Reuters places tags) to improve accuracy.

Results are cached in memory to avoid redundant geocoding requests for the same place name.

If geocoding succeeds, the resulting latitude and longitude are stored as the document's geopoint. The place name that produced the geopoint is also recorded in the geopoint_from field.

If no location can be successfully geocoded, the document remains without a geopoint.

6.5 Geographical Confidence and Metadata:

Each extracted geographical reference is stored with:

I. the original place name,
II. a normalized key used for aggregation (lowercase, alphanumeric only),
III. and a confidence score.

The confidence score is computed based on:

I. whether the place appears in the dateline (+2.0),
II. whether it appears in the title (+1.5),
III. and how frequently it appears in the title text (+0.3 per occurrence).

6.6 Country Code Normalization:

After initial geographical processing, documents are enriched with standardized ISO country codes through a post-processing step. Place names are mapped to alpha-2 codes (e.g., "us", "gb") using the pycountry library.

The normalization process handles:

I. Abbreviations with punctuation (U.S.A. → USA → us)
II. Common aliases (UK → gb)
III. Two-letter and three-letter country codes
IV. Exact country name matching against the pycountry database

The system performs exact matching only—no fuzzy matching is used to avoid incorrect mappings (e.g., preventing "Salvador" from being treated as "El Salvador").

The resulting countryKeys field is stored as a keyword array, enabling efficient country-based aggregation and analytics.

6.7 Approximation Flags:

If a geopoint is successfully generated from extracted place names rather than from an explicit document field, the system marks the geopoint as approximated (geopoint_is_approx: true) and records the source place name used to generate it in the geopoint_from field.

This information is stored as part of the document metadata to distinguish inferred locations from explicitly provided ones.

7. Query Processing and Ranking

This section describes how user queries are processed and how documents are ranked in the system. Query processing is implemented in the backend API and relies on Elasticsearch queries combined with semantic similarity scoring.

7.1 Query Interfaces

The system exposes multiple query endpoints through a FastAPI backend:

I. **Autocomplete query** (/autocomplete) for title suggestions
II. **Textual search query** (/search) with optional location and georeference parameters

III. **Spatio-temporal search query** (/spatiotemporal) combining text, time range, and location

Each query is handled by a dedicated function and translated into an Elasticsearch query.

7.2 Query Examples:

Text Search Example:

Query: "oil prices"

Optional parameters: lat=32.22, lon=35.25, georef="Palestine"

Result: Returns documents about oil with recency/location boosts applied

Spatiotemporal Search Example:

Query: "election"

Time range: 1987-01-01 to 1987-12-31

Location: lat=32.22, lon=35.25, distance=500km

Georeference: "Nablus"

Result: Returns election-related documents from 1987 near Nablus

7.3 Textual Search and Lexical Retrieval:

For standard text search, the system performs a **lexical retrieval stage** using Elasticsearch's query capabilities.

Key characteristics:

I. Both **title** and **content** fields are searched using multi_match.
II. The **title field is boosted** more heavily (^6) than content (^1).
III. Phrase matching on the title receives additional boosting (^10).
IV. **Exact title matches** receive the highest boost (^50) through a term query on title.raw.

This design ensures that documents with strong title matches are ranked higher than those matching only the content.

7.4 Georeference Boosting:

When a georeference parameter is provided by the user, the system applies an additional boosting mechanism. Documents containing matching place names in their georeferences nested field receive significant boosts.

The boost is computed using a script that multiplies the base score by the georeference's confidence value and an additional factor (5.0). This ensures that documents with high-confidence mentions of the requested location are prioritized.

The georeference boost is added to the should clause, meaning it enhances rankings but does not filter out documents.

7.5 Semantic Re-ranking with Embeddings:

After lexical retrieval, the system optionally applies **semantic re-ranking** using document embeddings.

If a query embedding is provided:

I. Cosine similarity is computed between the query vector and the document's content_embedding.
II. The similarity score is added to the lexical score using the formula:

final_score = base_lexical_score × (1.0 + cosine_similarity)

III. For exact title matches detected through the title.raw field, the score is further boosted:

final_score = base_score × 10.0

Semantic scoring does **not** filter documents, it only adjusts their ranking to surface semantically similar results.

7.6 Recency and Localization Boosting:

The final document score is adjusted using additional scoring functions based on function_score:

IV. **Recency boosting** is applied using a Gaussian decay function on the document date field. Documents closer to "now" receive higher scores, with a decay scale of 3650 days (approximately 10 years) and a decay factor of 0.7.
V. **Localization boosting** is applied when latitude and longitude are provided in the query. This uses a Gaussian decay function on the document's geopoint field, with a decay scale of 300 km and a decay factor of 0.6.

These boosts are combined **multiplicatively** with the base score using **boost_mode: multiply**, allowing time and location factors to scale the relevance score proportionally. The individual function scores are summed together using **score_mode: sum**.

7.7 Spatio-temporal Query Processing:

For spatio-temporal queries, the system combines multiple constraints:

I. **Textual matching** on title (^4 boost) and content using multi_match with fuzziness.
II. A **date range filter** using the range query on the document date field.
III. **Geographical distance filtering** applied through Gaussian decay functions rather than strict radius constraints.

If a georeference parameter is provided, it is added as an **optional boost** in the should clause (not as a required filter).

If a semantic embedding is available, semantic similarity is computed and added on top of these constraints. Additional recency and spatial decay functions are applied to refine the ranking further, using a decay scale of 365 days for time and 500 km for distance.

Documents must satisfy the textual query and fall within the specified date range. Georeference matching provides an **optional boost** rather than a hard filter. Geographical location filtering is implemented through decay-based scoring rather than strict distance cutoffs, allowing documents slightly outside the nominal range to still appear if they are otherwise highly relevant.

8. Autocomplete and Fuzzy Search:

The system provides an autocomplete feature to help users discover relevant documents while typing their queries. This functionality is designed specifically for **document titles** and is triggered only after a minimum number of characters is entered.

8.1 Autocomplete Trigger Condition:

Autocomplete suggestions are activated **only when the user enters at least three characters**. This condition is enforced both on the frontend (to avoid unnecessary requests) and on the backend (through API validation) to ensure that suggestions are meaningful.

Once the condition is met, the frontend sends the current input to the backend autocomplete endpoint.

8.2 Autocomplete Query Design:

The autocomplete query searches only within the **title field** of documents. It uses a bool query that combines two matching strategies:

I. **Prefix-based matching** using match_bool_prefix, which allows partial title matches as the user types. This leverages the edge n-gram analyzer configured during indexing.
II. **Fuzzy matching** using a standard match query with fuzziness: AUTO, which tolerates minor spelling mistakes in the user input (up to 2 character edits).

The query structure requires that at least one of these matching strategies must succeed (minimum_should_match: 1) for a document to be considered a valid suggestion.

The results are limited to the **top 10 matching titles**, ranked according to Elasticsearch scoring.

8.3 Autocomplete Index Support:

The title field is indexed using a dedicated autocomplete analyzer based on edge n-grams. This analyzer:

I. Generates n-grams of lengths 3 to 20 characters from the beginning of each token
II. Enables fast prefix matching without requiring complex query-time processing

A separate search analyzer is used during retrieval to ensure correct matching behavior. The search analyzer treats the query as a complete term rather than generating n-grams from it, which improves matching accuracy.

8.4 Frontend Interaction:

On the frontend side, autocomplete requests are debounced using a 300ms delay to avoid excessive API calls while the user is typing. When suggestions are returned, they are displayed in a dropdown list below the search input field.

Selecting a suggestion automatically fills the search box with the chosen title and triggers a full search request.

9. Analytics and Aggregations:

In addition to document retrieval, the system provides basic analytics over the indexed collection. These analytics are computed directly using Elasticsearch aggregations and are exposed through dedicated API endpoints.

9.1 Top Georeferences Analytics:

The system provides an endpoint that returns the **top mentioned geographical references** across the entire document collection.

This analytics query:

I. operates on the georeferences nested field,
II. groups locations using their normalized key field,
III. and returns the most frequently occurring locations.

The result is limited to the **top 10 georeferences**, sorted by frequency. These results allow users to understand which locations are most commonly mentioned in the dataset.

9.2 Temporal Distribution Analytics:

Another analytics endpoint returns the **distribution of documents over time**.

This query:

I. uses the document-level date field,
II. groups documents using a date histogram aggregation,
III. and returns a time-series showing how many documents appear on each date.

The aggregation interval is fixed at **one day**, which allows fine-grained temporal analysis of the document collection and can reveal patterns in publication frequency over time.

10. Limitations and Future Work:

While the system successfully demonstrates spatio-temporal information retrieval, several limitations should be acknowledged:

10.1 Current Limitations:

I. **Geocoding Dependency**: The system relies on the Nominatim external geocoding service, which can experience failures, timeouts, or rate limiting. Geocoding results are cached only in memory during preprocessing and are not persisted across runs.
II. **Temporal Context Assumptions**: Temporal extraction assumes a Reuters 1987 context when no explicit date is available. This may not generalize well to other document collections or time periods.

III. **Semantic Similarity Scope**: Semantic embeddings are computed only from document content, not from titles. This means semantic search may miss documents whose titles are highly relevant but whose content is less similar.
IV. **Place Name Ambiguity**: The system does not handle ambiguous place names (e.g., "Paris, Texas" vs. "Paris, France") beyond basic country hints. More sophisticated disambiguation would require additional context analysis.
V. **Single Geopoint per Document**: Each document is assigned only one geopoint, even if multiple distinct locations are mentioned. This limits the spatial precision for documents covering multiple regions.

11. Conclusion

In this project, a **Smart Document Retrieval System** was designed and implemented using Elasticsearch and Natural Language Processing techniques. The system goes beyond traditional keyword-based search by supporting **spatio-temporal information retrieval**, allowing users to search documents based on text, time, and geographical location.

11.1 Summary of Achievements:

The system successfully processes a collection of **Reuters news articles** by parsing raw SGM files, extracting temporal expressions and geographical references, and indexing structured documents into Elasticsearch. Custom index mappings and analyzers were used to support autocomplete, full-text search, and efficient aggregation. Semantic embeddings were integrated to enhance ranking through similarity-based re-ranking.

The implemented query engine supports:

I. **Autocomplete** with fuzzy matching and typo tolerance after 3 characters
II. **Textual and semantic search** with title boosting (^6 for multi_match, ^10 for phrase, ^50 for exact) and georeference confidence-based boosting (×5.0)
III. **Spatio-temporal queries** combining text, date ranges, and location constraints with decay-based scoring (3650-day and 365-day temporal scales, 300km and 500km spatial scales)
IV. **Analytics** including top-10 georeferences aggregation and daily temporal distribution

11.2 Technical Contributions:

Key technical achievements include:

Indexing & Preprocessing:

I. Robust document validation (id + non-empty content) before indexing
II. Semantic embedding generation using all-MiniLM-L6-v2 with L2 normalization
III. Multi-source temporal extraction (dateline, title, content) with confidence scoring
IV. Geocoding pipeline with dateline prioritization and country code normalization

Query Processing:

I. Multi-stage retrieval: lexical (BM25) → semantic (cosine similarity) → function scoring (recency + location)
II. Multiplicative boost combination (boost_mode: multiply) for proportional scoring
III. Nested field boosting for georeferences with confidence weighting
IV. Spatio-temporal filtering with soft constraints (decay functions vs. hard cutoffs)

Text Analysis:

I. Custom content analyzer: HTML removal, stopword filtering, min length 3, stemming
II. Edge n-gram autocomplete analyzer (3-20 characters) for responsive suggestions
III. Dual field indexing (analyzed title + keyword title.raw) for exact matching