Semantic Search in Articles using NLP

1.Introduction

Traditional keyword-based search systems often fail to grasp user intent, as they rely on exact word matching and struggle with synonyms and context. Semantic search addresses these limitations by understanding the meaning behind a query, matching concepts rather than just keywords. This project's primary objective is to design, implement, and evaluate an end-to-end pipeline for performing semantic search on a corpus of news articles, demonstrating a clear progression from a traditional baseline to a sophisticated, optimized deep learning solution.

The pipeline developed in this project encompasses four key stages:

1. **Data Preprocessing:** Cleaning and normalizing the raw text of news articles to prepare them for analysis.

2. **Baseline Modeling:** Implementing a keyword-based search using the classic Term Frequency-Inverse Document Frequency (TF-IDF) method to establish a performance benchmark.

3. **Advanced Semantic Modeling:** Utilizing a state-of-the-art Sentence-BERT model to generate dense vector embeddings that capture the semantic meaning of the articles.
4. **Performance Optimization:** Accelerating the search process using Facebook AI Similarity Search (FAISS), a library designed for efficient retrieval in massive-scale vector datasets.

2.Data Description

The dataset used for this project is a collection of news articles from the BBC, provided in a CSV file named `bbc_news_20220307_20240703.csv`. This dataset contains articles published over a period of more than two years.

2.1. Data Preprocessing

To enable effective analysis, a rigorous preprocessing pipeline was applied to both the `title` and `description` columns. This process standardizes the text and reduces its complexity, allowing the models to focus on meaningful content. The output of this pipeline was stored in two new columns: `title_clean` and `news_clean`.

The preprocessing methodology involved the following sequential steps:

1. **Lowercasing:** All text was converted to lowercase. This is a fundamental normalization step that ensures the model does not treat words with different capitalization (e.g., "War" and "war") as distinct entities.
2. **HTML Tag Removal:** The text was scanned for any residual HTML tags which were removed using regular expressions. These tags provide no semantic value for this task and are considered noise.
3. **Non-Alphabetic Character Removal:** All characters that were not letters or spaces were removed. This step eliminates punctuation, numbers, and special symbols, simplifying the vocabulary.
4. **Tokenization:** The cleaned string of text was broken down into a list of individual words, or "tokens." This was achieved using the NLTK library's tokenizer.
5. **Stopword Removal:** Common words that provide little semantic meaning, known as stopwords (e.g., "the," "a," "in," "is"), were removed from the token lists. Additionally, very short tokens (less than three characters) were discarded.
6. **Lemmatization:** The remaining tokens were reduced to their base or dictionary form, a process known as lemmatization.

3. Baseline Experiment 1: TF-IDF

Before implementing complex semantic models, it is crucial to establish a baseline. A baseline provides a point of comparison to quantify the improvements offered by more advanced techniques. For this project, the Term Frequency-Inverse Document Frequency (TF-IDF) method was chosen as the baseline, as it represents a robust and widely-used standard for traditional information retrieval and keyword-based search.

3.1. Goal

The primary goal of the baseline experiments was twofold:

1. To create a keyword extraction system that can identify the most statistically significant terms within each news article.
2. To build a functional search engine that ranks documents based on lexical similarity to a given query

3.2. Methodology

TF-IDF is a numerical statistic that reflects how important a word is to a document within a collection or corpus. It is the product of two metrics: Term Frequency (TF), which measures how often a term appears in a document, and Inverse Document Frequency (IDF), which measures how rare or common a term is across all documents. The core idea is that terms that are frequent in a specific document but rare across the entire corpus are highly descriptive of that document's content.

3.3. Vectorization

A `TfidfVectorizer` from the Scikit-learn library was used to convert the preprocessed text corpus (the `news_clean` column) into a matrix of TF-IDF features. The vectorizer was configured with specific parameters to refine its output:

- **N-gram Range:** Set to `(1, 2)`, meaning the vectorizer considered both individual words (unigrams) and pairs of adjacent words (bigrams) as potential features. This helps capture more context (e.g., "prime minister" is more meaningful than "prime" and "minister" separately).
- **Max Features:** Limited to `5000`, meaning only the top 5,000 most frequent n-grams across the corpus were kept in the vocabulary. This helps control the dimensionality of the feature space and removes extremely rare, potentially noisy terms.
- **Min Document Frequency:** Set to `1`, allowing terms that appear in at least one document.

- **Stop Words:** An English stopword list was applied again at this stage to ensure comprehensive filtering.

This process resulted in a sparse matrix where each row represented a news article and each column represented a unique term (or bigram) from the vocabulary, with the cell value being the TF-IDF score of that term in that article.

3.4. Keyword Extraction

Using the generated TF-IDF matrix, a function was created to extract the top N keywords for any given document. For a specific article's vector, this function identifies the terms with the highest TF-IDF scores and returns them. This demonstrates the model's ability to summarize a document's key topics based on statistical term importance.

3.5.TF-IDF Search

A search function was implemented to retrieve articles based on a user query. The process was as follows:

1. The user's query string is passed through the exact same text preprocessing pipeline as the original articles.
2. The cleaned query is then transformed into a TF-IDF vector using the already-fitted `TfidfVectorizer.`

3. **Cosine Similarity** is computed between the query vector and all the document vectors in the corpus matrix. Cosine similarity measures the cosine of the angle between two vectors, providing a score from 0 to 1 that represents their orientation similarity. In this context, a score of 1 means the documents share the exact same terms with the same TF-IDF proportions, while a score of 0 indicates no shared terms.
4. The articles are ranked in descending order of their cosine similarity scores, and
5. the top K results are returned.

3.6.Results

The keyword extraction experiment provided insightful, albeit surface-level, summaries of the articles. For example, for an article with the title "Ukraine: Angry Zelensky vows to punish Russian atrocities," the top extracted keywords were directly related to the main entities and actions described.

Original Title	Extracted Keywords
Ukraine: Angry Zelensky vows to punish Russian atrocities	['ukrainian president', 'president say', ';civilian', 'murder', 'ukrainian', 'country', 'president', 'say']

Table 1: Top 8 TF-IDF Keywords for a Sample Article

The TF-IDF search experiment returned documents that contained a high number of the exact keywords from the query. For a query like "higher energy costs," it prioritized documents that literally contained the words "energy" and "costs," but struggled to find conceptually similar articles.

Article Title	Cleaned Description Snippet	Similarity Score
Warning of 'significant' blow to households	household face significant blow living standar...	0.297
Energy price cap to rise by 'at least' another £600	typical household energy bill could rise least...	0.283
Cost of living: How to get help with energy bills	government announced package help household st...	0.271

Table 2: Top 3 TF-IDF Search Results for "higher energy costs"

3.7. Conclusion

The TF-IDF baseline is fast and simple but lacks semantic understanding. It cannot match concepts, only exact keywords, highlighting the need for a more advanced approach.

4.Experiment 2: Semantic Search with Sentence-BERT

4.1. Goal

To build a search engine that understands meaning by replacing TF-IDF vectors with dense semantic embeddings from Sentence-BERT (SBERT).

4.2.Methodology

A pre-trained SBERT model converted the article corpus into semantically rich vector embeddings. Scikit-learn's NearestNeighbors was used to find the most similar articles to a query based on cosine similarity.

4.3. Results

To test the semantic capabilities, the query "higher energy costs" was used. The results, shown below demonstrate a significant improvement over what a keyword-based system would produce.

Article Title	Cleaned Description Snippet	Similarity Score
Rich to get twice as much cost-of-living support	resolution foundation say limit energy cost ne...	0.694
Schools to cut staff in budget squeeze, union ...	funding increasing cost pay energy rising fast	0.648
What is the UK inflation rate and why is the c...	cost living increasing fastest rate year due r...	0.620

Table 3: Top 3 Semantic Search Results using Sentence-BERT and NearestNeighbors

The model correctly identified articles discussing "cost-of-living,"; "budget squeeze," and "inflation" as highly relevant to "higher energy costs."

5.Experiment 3: Accelerating Search with FAISS

5.1. Goal

The goal of this final experiment was to address the scalability bottleneck of the previous step. The objective was to replace the Scikit-learn `NearestNeighbors` index with a highly optimized one from Facebook AI Similarity Search (FAISS) to enable massive-scale, low-latency similarity search while maintaining the high accuracy of the SBERT embeddings.

5.2. Methodology

FAISS is a vectordb specifically designed for efficient similarity search and clustering of dense vectors

FAISS Search Function: A new search function was implemented. It encodes the user query with SBERT, converts the resulting vector to a 32-bit float, and then uses the index.search(query_vector, k) method. This method returns the similarity scores and indices of the top K matching documents directly and with very low latency.

5.3. Results

The same query, "higher energy costs," was executed using the FAISS-powered search function. The results were compared against the previous experiment to validate accuracy.

Article Title	Cleaned Description Snippet	Similarity Score
Rich to get twice as much cost-of-living suppo...	resolution foundation say limit energy cost ne...	0.694
Schools to cut staff in budget squeeze, union ...	funding increasing cost pay energy rising fast	0.648
What is the UK inflation rate and why is the c...	cost living increasing fastest rate year due r...	0.620

Table 4: Top 3 Semantic Search Results using Sentence-BERT and FAISS

6. Overall Conclusion

This project successfully built a scalable semantic search pipeline. It proved the superiority of **Sentence-BERT** over the TF-IDF baseline and used **FAISS** to ensure high-speed, production-ready performance. The final system effectively understands user intent rather than just matching keywords.

7.Tools and Libraries Used

- **Language**: Python
- **Libraries:** pandas, numpy, nltk, scikit-learn, sentence-transformers, faiss-cpu, re.

8.External Resources

Dataset: `bbc_news_20220307_20240703.csv`

Kaggle: https://www.kaggle.com/datasets/gpreda/bbc-news

Documentation: The official documentation for libraries such as Scikit-learn, Sentence-Transformers, and FAISS were consulted during implementation.

9.Project Reflection

1) What was the biggest challenge you faced when carrying out this project?

- Demonstrating the qualitative difference between methods through example queries.
- Comparing retrieval speed and semantic quality side by side instead of only relying on metrics.

2) What do you think you have learned from the project?

Key learnings include the practical superiority of semantic search over lexical methods, the necessity of an end-to-end pipeline perspective (from data cleaning to performance optimization), and the importance of using specialized tools like FAISS to build scalable, real-world applications