Semantic Search in articles using NLP

1 Introduction

1.1 The Problem

Users expect to find information not just by matching keywords, but by expressing what they need in natural language. Traditional search methods, like TF-IDF, excel at finding documents with exact keyword overlap but often fail to understand the user's true goal, especially for conceptual or nuanced queries. This project addresses the challenge of building a really smart search engine that can use semantic meaning.

1.2 Project Goal & Accomplishment

The approach of this project is to develop, compare, and evaluate two different search pipelines for a corpus of Medium articles:

1. A **classical, keyword-based search engine** using the TF-IDF algorithm.
2. A **modern, semantic search engine** using a state-of-the-art deep learning Sentence Transformer model.

2 Core Technologies and Libraries

2.1 Core Technologies

- **Python:** The primary programming language used for the entire project.
- **Jupyter / Google Colab:** The development environment, indicated by the .ipynb file format and the use of shell commands (`!pip install`).

2.2 NLP & Machine Learning Libraries

- **Scikit-learn:** A fundamental machine learning library used for:
  - **TF-IDF:** Implementing the baseline keyword-based search model (`TfidfVectorizer`).

  - **Cosine Similarity:** Calculating the similarity between document and query vectors.
  - **Data Splitting:** Creating stratified train, validation, and test sets (`train_test_split`).

- **Sentence-Transformers:** A state-of-the-art library for generating dense vector embeddings for text. It is the core of the semantic search engine, used to evaluate and deploy models like `all-mpnet-base-v2`.
- **KeyBERT:** A transformer-based library used for advanced, context-aware keyword extraction. It leverages `Sentence-Transformers` to find keywords that are most semantically similar to the document's main theme.

2.3 Data Sourcing & Handling

- **Kaggle Hub (kagglehub):** The official Kaggle library used to programmatically download the "Medium Articles" dataset.
- **Pandas:** The primary tool for data manipulation, used for reading the CSV, cleaning the data, creating new features (like `word_count`), and managing the data splits.
- **NumPy:** Used for efficient numerical operations, particularly for handling the scores and indices returned by the search models.
- **Pickle:** A standard Python library used for serializing and saving the pre-computed document embeddings to disk, which saves significant time by preventing re-computation.

2.4 Text Processing & Keyword Extraction

- **BeautifulSoup (bs4):** A library used to parse and remove HTML tags from the raw article text.
- **Regular Expressions (re):** Used extensively in the text cleaning pipeline to remove URLs, email addresses, and special characters.
- **YAKE (yake):** An open-source, unsupervised keyword extraction library that was installed (likely for initial exploration).

2.5 Data Visualization

- **Matplotlib:** The foundational plotting library in Python, used for creating visualizations of the results.
- **Seaborn:** A statistical data visualization library built on top of `Matplotlib`, included for potentially creating more advanced plots.

3 Data Description

3.1 Original Dataset

The project utilizes the **"Medium Articles" dataset** sourced from Kaggle (hsankesara/medium-articles). The original raw dataset contained **337 instances** (articles) and featured columns such as title, text, claps, reading_time, and url. For the purpose of building a search engine, this project focused on the text column, which contains the full content of each article.

3.2 Data Preprocessing Pipeline

To ensure the quality of the data used for modeling, I implemented a preprocessing pipeline. Those are the pipeline steps:

- **Duplicate Removal:** Identified and removed **107 duplicate articles**.
- **Text Cleaning:** Standardized the text by removing HTML tags, URLs, and special characters, followed by lowercasing and normalizing whitespace.
- **Content Filtering:** Filtered out articles that were too short (under 50 words) or excessively long (top 1% quantile), resulting in the removal of **4 articles**.

3.3 Data Splitting

To facilitate a methodologically sound machine learning workflow, the preprocessed dataset was divided into three sets:

- **Training Set: 158 articles (70%)** Used for the final models.
- **Validation Set: 34 articles (15%)** Used for hyperparameter tuning and model selection.
- **Test Set: 34 articles (15%)** A completely held-out set used only for the final, unbiased evaluation of the models.

Table 1: Distribution of Article Lengths Across Data Splits

Word Quartile	Train Set %	Validation Set %	Test Set %
short	24.68	26.47	26.47
medium	25.32	23.53	23.53
long	24.68	26.47	23.53
very_long	25.32	23.53	26.47

4 Baseline Experiment: TF-IDF Search

4.1 Goal

The goal of the baseline experiment is to build and optimize a keyword-based search engine using the TF-IDF (Term Frequency-Inverse Document Frequency) algorithm. This model serves as a robust benchmark against which the more advanced deep learning model can be compared. At first I wanted to determine the optimal hyperparameters for the TfidfVectorizer to ensure the baseline was as strong as possible.

4.2 Experimental Steps & Results

The optimization process was iterative using the **validation set** and a set of validation queries.

4.2.1 Round 1: Broad Hyperparameter Search

A broad search was initially performed across 72 different parameter combinations, focusing on max_features, min_df, max_df, and ngram_range. The top-performing models from this round consistently converged on a vocabulary size of **1,000 features**.

- **Top Result (Round 1):**
  - **Score:** 0.1501
  - **Parameters:** {'max_features': 1000, 'min_df': 2, 'ngram_range': (1, 1)}

4.2.2 Round 2: Refined Hyperparameter Search

While the first round provided a good starting point, it was possible that a smaller, more focused vocabulary might yield better performance, especially that the top results at the last round was all for the least max_features. A second search was done over 30 combinations, focusing on max_features values around and below 1000.

This refined search yielded a significant improvement in the validation score and identified a clear better vocabulary size.

- **Top 5 Results (Round 2):**

The results clearly show that performance peaked with a vocabulary of **50 features** and then began to decline as the feature count increased, indicating that larger vocabularies were introducing noise.

Table 2: Top 5 Results from Refined Hyperparameter Search

Rank	Validation Score	max_features	min_df	ngram_range
1	**0.1684**	**50**	**3**	**(1, 2)**
2	0.1684	50	4	(1, 2)
3	0.1684	50	5	(1, 2)
4	0.1347	5	3	(1, 2)
5	0.1347	5	4	(1, 2)

4.3 Conclusion

The iterative hyperparameter tuning process was highly successful. It revealed that a smaller, more curated vocabulary that includes bigrams provides the best performance for a general-purpose keyword search on this dataset.

The best configuration for the baseline TF-IDF model is:

- **max_features**: 50
- **min_df**: 3
- **ngram_range**: (1, 2)
- **max_df**: 0.9

5 Other Experiments: Semantic Search Engine

5.1 Goal

The goal of this experiment was to move beyond keyword-matching and build a search engine using modern deep learning. The objective was to use pre-trained Sentence Transformer models to create a system capable of understanding the *semantic meaning* of the articles and user queries. The experiment was designed to know the most effective pre-trained model for this specific task and dataset.

5.2 Experimental Steps & Results

The experiment was conducted using the **validation set**.

5.2.1 Steps

1. **Candidate Selection:** Three popular and high-performing Sentence Transformer models were chosen as candidates, each with different strengths:

- all-MiniLM-L6-v2: A fast and efficient model, well-balanced for performance and resource usage.
- all-mpnet-base-v2: A larger, higher-quality model known for generating excellent general-purpose embeddings.
- msmarco-distilbert-base-v4: A model specifically fine-tuned on a massive question-answering dataset, making it a strong candidate for information retrieval.

2. **Evaluation:** Each model was tasked with encoding the validation documents and processing the general-purpose validation_queries. The average cosine similarity score for the top 5 results for each query was used as the performance metric.
3. **Model Selection:** The model with the highest average validation score was selected as the champion for the final semantic search engine.

5.2.2 Results

The evaluation produced a winner. The performance of each model on the validation set is summarized in the table below:

Table 3: Performance of Sentence Transformer Models

Model Name	Average Validation Score	Notes
all-MiniLM-L6-v2	**0.2936**	**Best Performance**
all-mpnet-base-v2	0.2925	Close second, slightly lower score
msmarco-distilbert-base-v4	0.2506	Lower performance on this task

5.3 Conclusion

The experiment successfully identified **all-MiniLM-L6-v2** as the optimal model for this semantic search task.

After its selection, the final Semantic Search engine was built by:

1. Encoding all **158 articles** from the training set using the all-MiniLM-L6-v2 model, resulting in an embedding matrix of shape (158, 384).
2. Integrating the KeyBERT library to provide a mechanism for extracting thematic keywords from search results, thereby adding a layer of explainability to the deep learning model.

This optimized semantic engine is now ready for the final comparison against the TF-IDF baseline.

6 Overall Conclusion

Now the models were subjected to a final, definitive evaluation on an unseen test set. This final comparison provides a clear and multi-faceted answer to the central question: which method is best?

6.1 Findings

Table 4: Comparison of TF-IDF and Semantic Search on Test Queries

Query	**TF-IDF Score / Keywords**	**Semantic Score / Keywords**	**Best Model**	**Notes**
Clean code practices	0.3025 / code	0.1479 / overfitting, deep learning	Neither	Both struggled
Handle burnout at work	0.4473 / work	0.2850 / motivation	Semantic	Semantic found concepts
Content marketing for social media	0.0000 / —	0.3062 / interface, recs	Semantic	Semantic domain-relevant
Startups' first customers	0.0000 / —	0.2590 / recs, interface	Semantic	Semantic relevant
ML for beginners	0.9704 / learning, machine	0.6163 / machine learning	Both	Both relevant
Creativity in tech	0.0000 / —	0.2811 / tech jobs, stories	Semantic	Semantic conceptual

- **Relevance and Quality of Results:** The Semantic Search model showed a profound and consistently ability to understand user intent.
  - For the query **'how to handle burnout at work'**, the semantic model correctly identified conceptually related articles about motivation and workplace interruptions, whereas TF-IDF could only match the generic term "work".
  - For **'content marketing strategies for social media'** and **'how can startups find their first customers?'**, the TF-IDF model found zero relevant documents. The Semantic model, however, successfully returned documents from the correct domain, discussing user-facing products, search interfaces, and company growth.
  - Most impressively, for the abstract query 'the role of creativity in technology', the semantic model surfaced highly relevant articles on AI-driven story generation and the future of programming jobs, showing a true grasp of the underlying concept. In contrast, the TF-IDF model failed entirely.

  The only instance where the TF-IDF model performed comparably was for the query **'explaining machine learning to a beginner'**, where the keywords themselves were the core topic, making it an ideal case for a lexical search.
- **Speed and Performance:**The trade-off for the semantic model's intelligence is a significant difference in computational cost. As summarized in the performance evaluation, the TF-IDF model is approximately 16.7 times faster than the Semantic model, with an average search time of just 86.35 ms compared to 1439.15 ms. This latency difference is due to the computational expense of generating a dense vector embedding for the user's query and then performing a vector similarity search across

the entire document database, whereas TF-IDF relies on a much faster inverted index lookup.

6.2 Final Verdict

While the TF-IDF model offers a significant advantage in search latency, its performance is brittle and unreliable. It is only effective for a narrow range of queries and fails completely when a user's search terms fall outside its limited, keyword-based worldview. This makes it unsuitable for a modern, user-centric application where understanding natural language is paramount.

The **Semantic Search engine is the best model**. Its ability to know the meaning and intent behind a query even in the absence of direct keyword matches represents a huge improvement in search quality. It provides a far more robust, intelligent, and useful experience for the end-user.

7 How to re-produce the results

Run in colab environment the Semantic_keywords.ipynb notebook from the repo.

8 Questions

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

The biggest challenge was to understand and apply the best way to use (training/test and validation) splits for such an unsupervised learning task.

8.2 What do you think you have learned from the project?

How (training/test and validation) can help in the phase of development in unsupervised learning tasks.