

NirDiamant /  
RAG\_Techniques

&lt;&gt; Code

Issues

Pull requests

Actions

Projects

Security

Insights

RAG\_Techniques / all\_rag\_techniques / semantic\_chunking.ipynb



oaustegard Update semantic\_chunking.ipynb -- added sources



470843b · 3 months ago



243 lines (243 loc) · 8.2 KB

# Semantic Chunking for Document Processing

## Overview

This code implements a semantic chunking approach for processing and retrieving information from PDF documents, [first proposed by Greg Kamradt](#) and subsequently [implemented in LangChain](#). Unlike traditional methods that split text based on fixed character or word counts, semantic chunking aims to create more meaningful and context-aware text segments.

## Motivation

Traditional text splitting methods often break documents at arbitrary points, potentially disrupting the flow of information and context. Semantic chunking addresses this issue by attempting to split text at more natural breakpoints, preserving semantic coherence within each chunk.

## Key Components

1. PDF processing and text extraction
2. Semantic chunking using LangChain's SemanticChunker
3. Vector store creation using FAISS and OpenAI embeddings
4. Retriever setup for querying the processed documents

## Method Details

### Document Preprocessing

1. The PDF is read and converted to a string using a custom

[RAG\\_Techniques](#) / [all\\_rag\\_techniques](#) / [semantic\\_chunking.ipynb](#)

[↑ Top](#)

Preview

Code

Blame

Raw



1. Utilizes LangChain's `SemanticChunker` with OpenAI embeddings.
2. Three breakpoint types are available:
  - 'percentile': Splits at differences greater than the X percentile.
  - 'standard\_deviation': Splits at differences greater than X standard deviations.
  - 'interquartile': Uses the interquartile distance to determine split points.
3. In this implementation, the 'percentile' method is used with a threshold of 90.

### Vector Store Creation

1. OpenAI embeddings are used to create vector representations of the semantic chunks.
2. A FAISS vector store is created from these embeddings for efficient similarity search.

## Retriever Setup

1. A retriever is configured to fetch the top 2 most relevant chunks for a given query.

## Key Features

1. Context-Aware Splitting: Attempts to maintain semantic coherence within chunks.
2. Flexible Configuration: Allows for different breakpoint types and thresholds.
3. Integration with Advanced NLP Tools: Uses OpenAI embeddings for both chunking and retrieval.

## Benefits of this Approach

1. Improved Coherence: Chunks are more likely to contain complete thoughts or ideas.
2. Better Retrieval Relevance: By preserving context, retrieval accuracy may be enhanced.
3. Adaptability: The chunking method can be adjusted based on the nature of the documents and retrieval needs.
4. Potential for Better Understanding: LLMs or downstream tasks may perform better with more coherent text segments.

## Implementation Details

1. Uses OpenAI's embeddings for both the semantic chunking process and the final vector representations.
2. Employs FAISS for creating an efficient searchable index of the chunks.
3. The retriever is set up to return the top 2 most relevant chunks, which can be adjusted as needed.

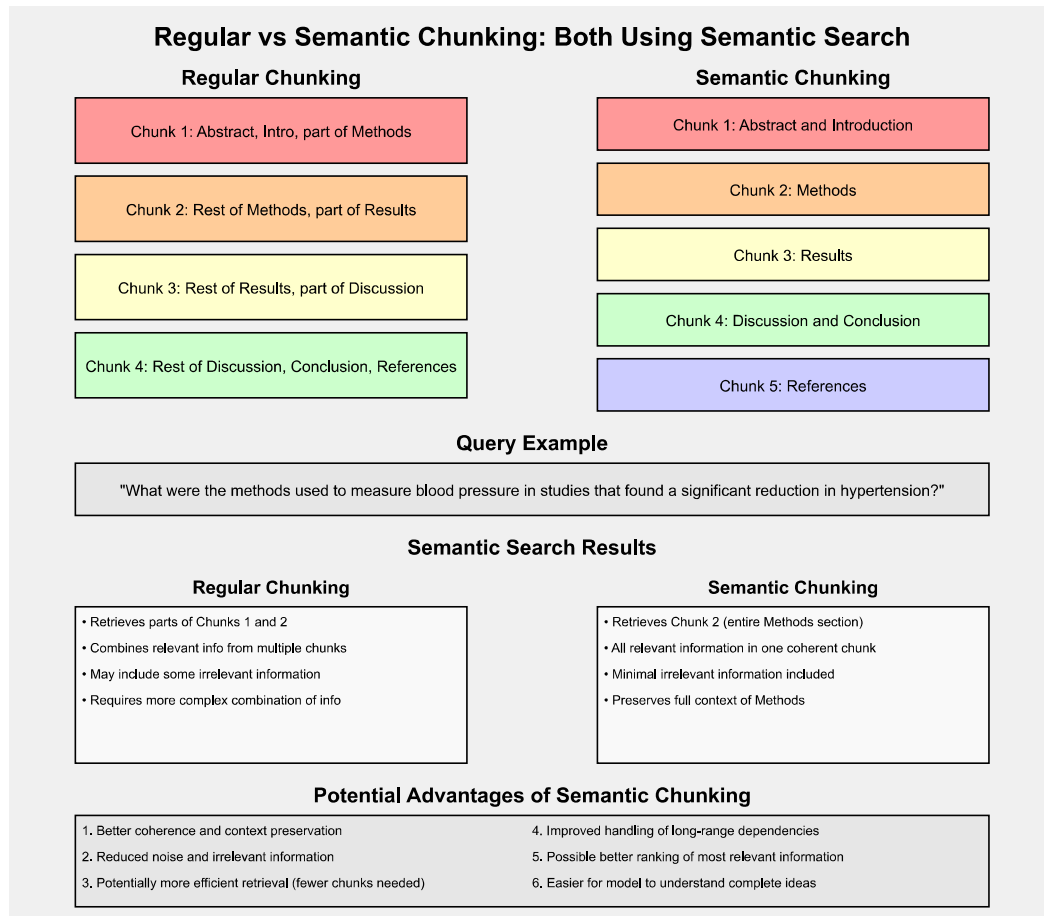
## Example Usage

The code includes a test query: "What is the main cause of climate change?". This demonstrates how the semantic chunking and retrieval system can be used to find relevant information from the processed document.

## Conclusion

Semantic chunking represents an advanced approach to document processing for

retrieval systems. By attempting to maintain semantic coherence within text segments, it has the potential to improve the quality of retrieved information and enhance the performance of downstream NLP tasks. This technique is particularly valuable for processing long, complex documents where maintaining context is crucial, such as scientific papers, legal documents, or comprehensive reports.



## Import libraries

In [57]:

```
import os
import sys
from dotenv import load_dotenv

sys.path.append(os.path.abspath(os.path.join(os.getcwd(), '..'))) # Add the
from helper_functions import *
from evaluation.evaluate_rag import *

from langchain_experimental.text_splitter import SemanticChunker
from langchain_openai.embeddings import OpenAIEmbeddings

# Load environment variables from a .env file
load_dotenv()

# Set the OpenAI API key environment variable
os.environ["OPENAI_API_KEY"] = os.getenv('OPENAI_API_KEY')
```

## Define file path

In [3]:

```
path = "../data/Understanding Climate Change.pdf"
```

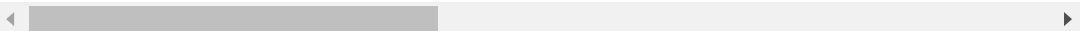
## Read PDF to string

```
In [18]: content = read_pdf_to_string(path)
```

## Breakpoint types:

- 'percentile': all differences between sentences are calculated, and then any difference greater than the X percentile is split.
- 'standard\_deviation': any difference greater than X standard deviations is split.
- 'interquartile': the interquartile distance is used to split chunks.

```
In [51]: text_splitter = SemanticChunker(OpenAIEmbeddings(), breakpoint_threshold_ty
```



## Split original text to semantic chunks

```
In [53]: docs = text_splitter.create_documents([content])
```

## Create vector store and retriever

```
In [54]: embeddings = OpenAIEmbeddings()  
vectorstore = FAISS.from_documents(docs, embeddings)
```