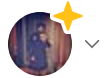




Search Medium

Write



Document Topic Extraction with Large Language Models (LLM) and the Latent Dirichlet Allocation (LDA) Algorithm

A guide on how to efficiently extract topics from large documents using Large Language Models (LLM) and the Latent Dirichlet Allocation (LDA) algorithm.



Antonio Jimenez Caballero · Follow

Published in Towards Data Science · 8 min read · Sep 14



258



3



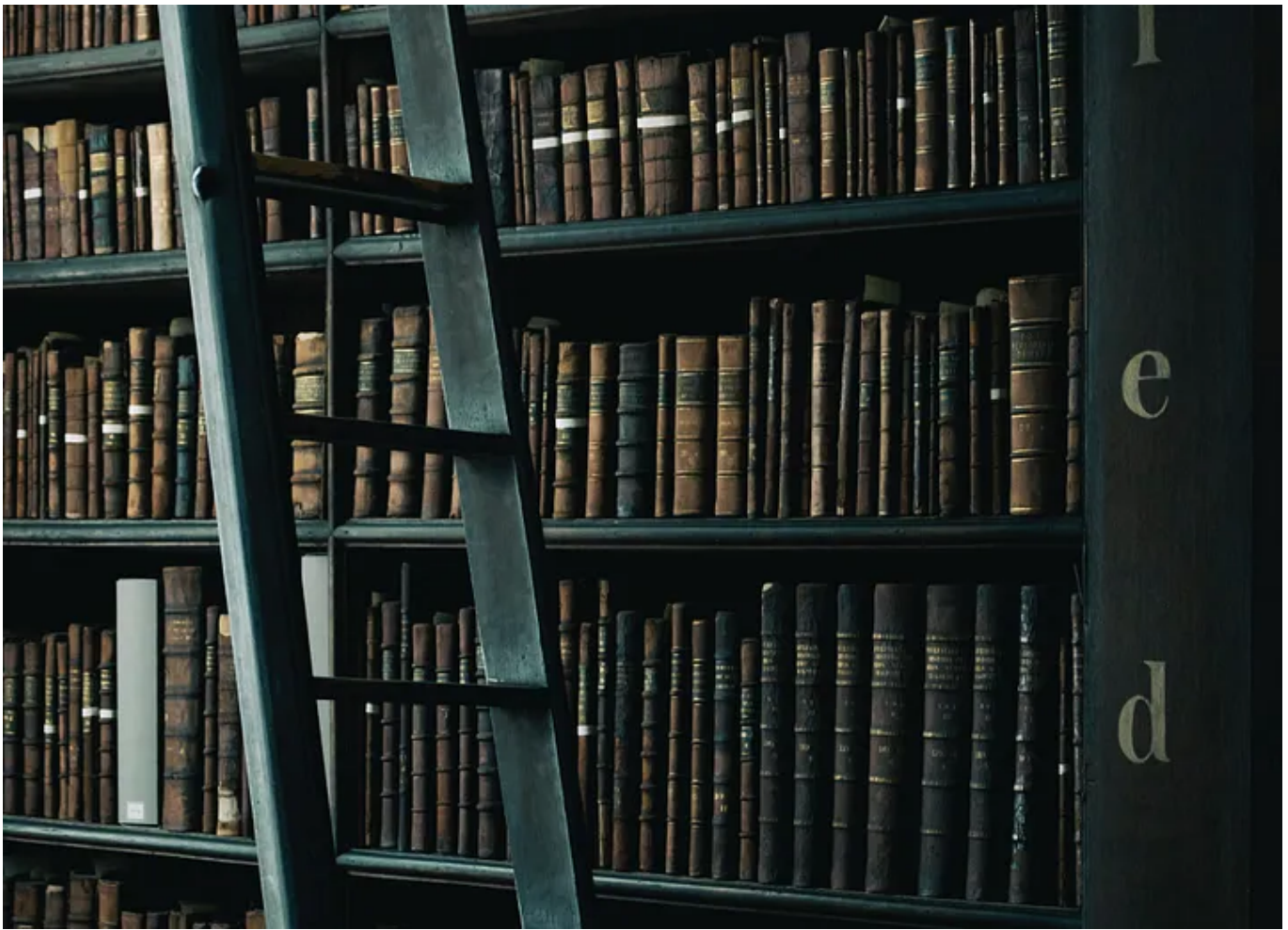


Photo by [Henry Be](#) on [Unsplash](#)

Introduction

I was developing a web application for chatting with PDF files, capable of processing large documents, above 1000 pages. But before starting a conversation with the document, I wanted the application to give the user a brief summary of the main topics, so it would be easier to start the interaction.

One way to do it is by summarizing the document using [LangChain](#), as showed in its [documentation](#). The problem, however, is the high computational cost and, by extension, the monetary cost. A thousand-page document contains roughly 250 000 words and each word needs to be fed into the LLM. Even more, the results must be further processed, as with the map-reduce method. A conservative estimate on the cost using gpt-3.5 Turbo

with 4k context is above 1\$ per document, just for the summary. Even when using free resources, such as the [Unofficial HuggingChat API](#), the sheer number of required API calls would be an abuse. So, I needed a different approach.

LDA to the Rescue

The Latent Dirichlet Allocation algorithm was a natural choice for this task. This algorithm takes a set of “documents” (in this context, a “document” refers to a piece of text) and returns a list of topics for each “document” along with a list of words associated with each topic. What is important for our case is the list of words associated with each topic. These lists of words encode the content of the file, so they can be fed to the LLM to prompt for a summary. I recommend [this article](#) for a detailed explanation of the algorithm.

There are two key considerations to address before we could get a high-quality result: selecting the hyperparameters for the LDA algorithm and determining the format of the output. The most important hyperparameter to consider is the number of topics, as it has the most significant on the final result. As for the format of the output, one that worked pretty well is the nested bulleted list. In this format, each topic is represented as a bulleted list with subentries that further describe the topic. As for why this works, I think that, by using this format, the model can focus on extracting content from the lists without the complexity of articulating paragraphs with connectors and relationships.

Implementation

I implemented the code in [Google Colab](#). The necessary libraries were gensim for LDA, pypdf for PDF processing, nltk for word processing, and LangChain for its prompt templates and its interface with the OpenAI API.

```
import gensim
import nltk
from gensim import corpora
from gensim.models import LdaModel
from gensim.utils import simple_preprocess
from nltk.corpus import stopwords
from pypdf import PdfReader
from langchain.chains import LLMChain
from langchain.prompts import ChatPromptTemplate
from langchain.llms import OpenAI
```

Next, I defined a utility function, *preprocess*, to assist in processing the input text. It removes stop words and short tokens.

```
def preprocess(text, stop_words):
    """
    Tokenizes and preprocesses the input text, removing stopwords and short
    tokens.

    Parameters:
        text (str): The input text to preprocess.
        stop_words (set): A set of stopwords to be removed from the text.
    Returns:
        list: A list of preprocessed tokens.
    """
    result = []
    for token in simple_preprocess(text, deacc=True):
        if token not in stop_words and len(token) > 3:
            result.append(token)
    return result
```

The second function, *get_topic_lists_from_pdf*, implements the LDA portion of the code. I accept the path to the PDF file, the number of topics, and the number of words per topic, and it returns a list. Each element in this list

contains a list of words associate with each topic. Here, we are considering each page from the PDF file to be a “document”.

```
def get_topic_lists_from_pdf(file, num_topics, words_per_topic):  
    """  
    Extracts topics and their associated words from a PDF document using the  
    Latent Dirichlet Allocation (LDA) algorithm.  
  
    Parameters:  
        file (str): The path to the PDF file for topic extraction.  
        num_topics (int): The number of topics to discover.  
        words_per_topic (int): The number of words to include per topic.  
  
    Returns:  
        list: A list of num_topics sublists, each containing relevant words  
        for a topic.  
    """  
    # Load the pdf file  
    loader = PdfReader(file)  
  
    # Extract the text from each page into a list. Each page is considered a doc  
    documents= []  
    for page in loader.pages:  
        documents.append(page.extract_text())  
  
    # Preprocess the documents  
    nltk.download('stopwords')  
    stop_words = set(stopwords.words(['english', 'spanish']))  
    processed_documents = [preprocess(doc, stop_words) for doc in documents]  
  
    # Create a dictionary and a corpus  
    dictionary = corpora.Dictionary(processed_documents)  
    corpus = [dictionary.doc2bow(doc) for doc in processed_documents]  
  
    # Build the LDA model  
    lda_model = LdaModel(  
        corpus,  
        num_topics=num_topics,  
        id2word=dictionary,  
        passes=15  
    )  
  
    # Retrieve the topics and their corresponding words  
    topics = lda_model.print_topics(num_words=words_per_topic)
```

```
# Store each list of words from each topic into a list
topics_ls = []
for topic in topics:
    words = topic[1].split("+")
    topic_words = [word.split("*")[1].replace("'", '').strip() for word in words]
    topics_ls.append(topic_words)

return topics_ls
```

The next function, *topics_from_pdf*, invokes the LLM model. As stated earlier, the model was prompted to format the output as a nested bulleted list.

```
def topics_from_pdf(llm, file, num_topics, words_per_topic):
    """
    Generates descriptive prompts for LLM based on topic words extracted from a
    PDF document.

    This function takes the output of `get_topic_lists_from_pdf` function,
    which consists of a list of topic-related words for each topic, and
    generates an output string in table of content format.

    Parameters:
        llm (LLM): An instance of the Large Language Model (LLM) for generating
        responses.
        file (str): The path to the PDF file for extracting topic-related words.
        num_topics (int): The number of topics to consider.
        words_per_topic (int): The number of words per topic to include.

    Returns:
        str: A response generated by the language model based on the provided
        topic words.
    """

    # Extract topics and convert to string
    list_of_topicwords = get_topic_lists_from_pdf(file, num_topics,
                                                    words_per_topic)

    string_lda = ""
    for list in list_of_topicwords:
        string_lda += str(list) + "\n"

    # Create the template
    template_string = '''Describe the topic of each of the {num_topics}
```

double-quote delimited lists in a simple sentence and also write down three possible different subthemes. The lists are the result of an algorithm for topic discovery.

Do not provide an introduction or a conclusion, only describe the topics. Do not mention the word "topic" when describing the topics. Use the following template for the response.

```
1: <<<(sentence describing the topic)>>>
- <<<(Phrase describing the first subtheme)>>>
- <<<(Phrase describing the second subtheme)>>>
- <<<(Phrase describing the third subtheme)>>>

2: <<<(sentence describing the topic)>>>
- <<<(Phrase describing the first subtheme)>>>
- <<<(Phrase describing the second subtheme)>>>
- <<<(Phrase describing the third subtheme)>>>

...

n: <<<(sentence describing the topic)>>>
- <<<(Phrase describing the first subtheme)>>>
- <<<(Phrase describing the second subtheme)>>>
- <<<(Phrase describing the third subtheme)>>>
```

Lists: ""{string_lda}"" '''

```
# LLM call
prompt_template = ChatPromptTemplate.from_template(template_string)
chain = LLMChain(llm=llm, prompt=prompt_template)
response = chain.run({
    "string_lda" : string_lda,
    "num_topics" : num_topics
})

return response
```

In the previous function, the list of words is converted into a string. Then, a prompt is created using the *ChatPromptTemplate* object from LangChain; note that the prompt defines the structure for the response. Finally, the function calls chatgpt-3.5 Turbo model. The return value is the response given by the LLM model.

Now, it's time to call the functions. We first set the API key. *This article* offers instructions on how to get one.

```
openai_key = "sk-p..."  
llm = OpenAI(openai_api_key=openai_key, max_tokens=-1)
```

Next, we call the *topics_from_pdf* function. I choose the values for the number of topics and the number of words per topic. Also, I selected a **public domain** book, The Metamorphosis by Franz Kafka, for testing. The document is stored in my personal drive and downloaded by using the *gdown* library.

```
!gdown https://drive.google.com/uc?id=1mpXUmuLGzkVEqSTicQvBPcpPJW0aPqdL  
  
file = "./the-metamorphosis.pdf"  
num_topics = 6  
words_per_topic = 30  
  
summary = topics_from_pdf(llm, file, num_topics, words_per_topic)
```

The result is displayed below:

```
1: Exploring the transformation of Gregor Samsa and the effects on his  
family and lodgers  
- Understanding Gregor's metamorphosis  
- Examining the reactions of Gregor's family and lodgers  
- Analyzing the impact of Gregor's transformation on his family  
  
2: Examining the events surrounding the discovery of Gregor's  
transformation
```


- Investigating the initial reactions of Gregor's family and lodgers
 - Analyzing the behavior of Gregor's family and lodgers
 - Exploring the physical changes in Gregor's environment
- 3: Analyzing the pressures placed on Gregor's family due to his transformation
- Examining the financial strain on Gregor's family
 - Investigating the emotional and psychological effects on Gregor's family
 - Examining the changes in family dynamics due to Gregor's metamorphosis
- 4: Examining the consequences of Gregor's transformation
- Investigating the physical changes in Gregor's environment
 - Analyzing the reactions of Gregor's family and lodgers
 - Investigating the emotional and psychological effects on Gregor's family
- 5: Exploring the impact of Gregor's transformation on his family
- Analyzing the financial strain on Gregor's family
 - Examining the changes in family dynamics due to Gregor's metamorphosis
 - Investigating the emotional and psychological effects on Gregor's family
- 6: Investigating the physical changes in Gregor's environment
- Analyzing the reactions of Gregor's family and lodgers
 - Examining the consequences of Gregor's transformation
 - Exploring the impact of Gregor's transformation on his family

The output is pretty decent, and it just took seconds! It correctly extracted the main ideas from the book.

This approach works with technical books as well. For example, *The Foundations of Geometry* by David Hilbert (1899) (also in the public domain):

- 1: Analyzing the properties of geometric shapes and their relationships
- Exploring the axioms of geometry
 - Analyzing the congruence of angles and lines
 - Investigating theorems of geometry
- 2: Studying the behavior of rational functions and algebraic equations
- Examining the straight lines and points of a problem
 - Investigating the coefficients of a function
 - Examining the construction of a definite integral

3: Investigating the properties of a number system

- Exploring the domain of a true group
- Analyzing the theorem of equal segments
- Examining the circle of arbitrary displacement

4: Examining the area of geometric shapes

- Analyzing the parallel lines and points
- Investigating the content of a triangle
- Examining the measures of a polygon

5: Examining the theorems of algebraic geometry

- Exploring the congruence of segments
- Analyzing the system of multiplication
- Investigating the valid theorems of a call

6: Investigating the properties of a figure

- Examining the parallel lines of a triangle
- Analyzing the equation of joining sides
- Examining the intersection of segments

Conclusion

Combining the LDA algorithm with LLM for large document topic extraction produces good results while significantly reducing both cost and processing time. We've gone from hundreds of API calls to just one and from minutes to seconds.

The quality of the output depends greatly on its format. In this case, a nested bulleted list worked just fine. Also, the number of topics and the number of words per topic are important for the result's quality. I recommend trying different prompts, number of topics, and number of words per topic to find what works best for a given document.

The code could be found in [this link](#).

Thank you for reading. Please let me know how it resulted with your documents. I hope soon to write about the implementation of the application I mentioned at the beginning.

LinkedIn: [Antonio Jimenez Caballero](#)

GitHub: [a-jimenezc](#)

[Llm](#)[Lda](#)[Text Preprocessing](#)[Langchain](#)[AI](#)

More from the list: "NLP"

Curated by Himanshu Birla



Jon Gi... in Towards Data ...

Characteristics of Word Embeddings

★ · 11 min read · Sep 4, 2021



Jon Gi... in Towards Data ...

The Word2vec Hyperparameters

★ · 6 min read · Sep 3, 2021



Jon Gi... in

The Word2vec

★ · 15 min rea



[View list](#)



Written by Antonio Jimenez Caballero


[Follow](#)

50 Followers · Writer for Towards Data Science

Lecturer and Professional Engineer currently developing Data Science web apps as personal projects. Passionate about learning, teaching, and mathematics.

More from Antonio Jimenez Caballero and Towards Data Science



 Antonis Makropoulos in Towards Data Science

How to Build a Multi-GPU System for Deep Learning in 2023

This story provides a guide on how to build a multi-GPU system for deep learning and...

10 min read · Sep 17



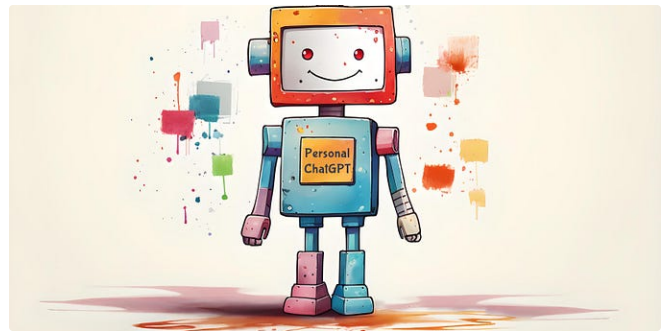
549



11



...



 Robert A. Gonsalves in Towards Data Science

Your Own Personal ChatGPT

How you can fine-tune OpenAI's GPT-3.5 Turbo model to perform new tasks using you...

🌟 · 15 min read · Sep 8



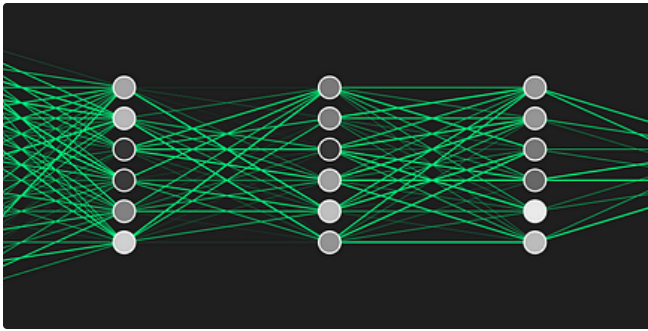
595



7



...



Callum Bruce in Towards Data Science

How to Program a Neural Network

A step-by-step guide to implementing a neural network from scratch

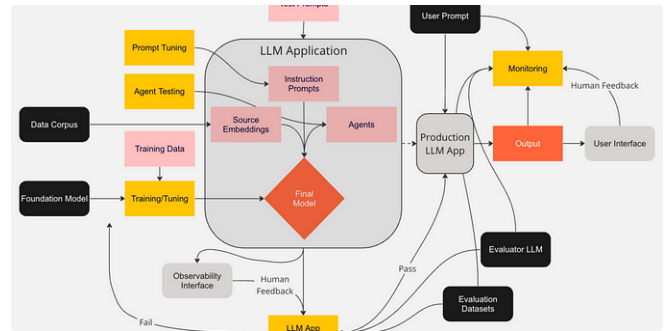
★ · 14 min read · Sep 24



470



458



Josh Poduska in Towards Data Science

LLM Monitoring and Observability

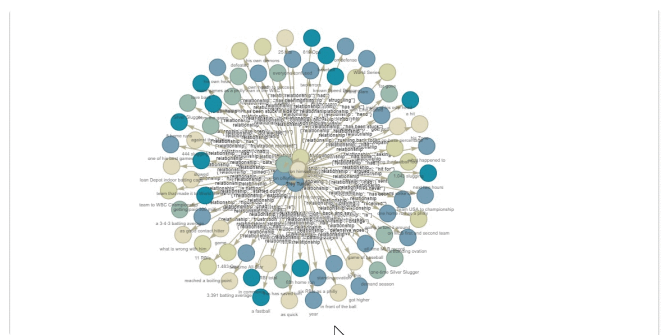
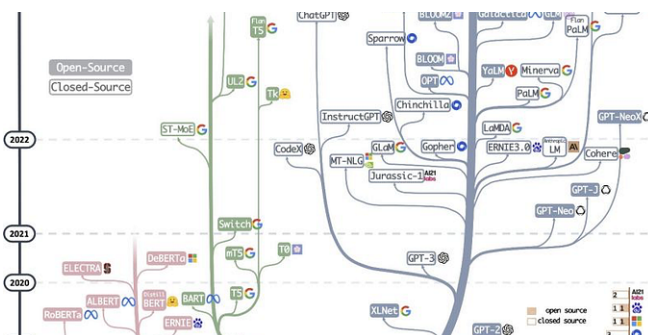
A Summary of Techniques and Approaches for Responsible AI

10 min read · Sep 15

See all from Antonio Jimenez Caballero

See all from Towards Data Science

Recommended from Medium





Haifeng Li

A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

15 min read · Sep 14



372



Wenqi Glantz in Better Programming

7 Query Strategies for Navigating Knowledge Graphs With...

Exploring NebulaGraph RAG Pipeline with the Philadelphia Phillies



17 min read · 4 days ago



501



4



Lists



Natural Language Processing

669 stories · 283 saves



Staff Picks

465 stories · 317 saves



Han HELOIR, Ph.D. in Artificial Corner

MongoDB and Langchain Magic: Your Beginner's Guide to Setting...

Introduction:



7 min read · Sep 12



1.4K



12



Ankit

Generating Summaries for Large Documents with Llama2 using...

Introduction

11 min read · Aug 28

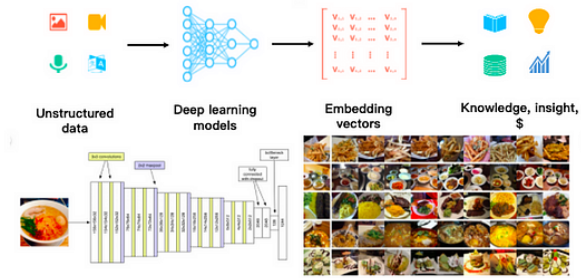
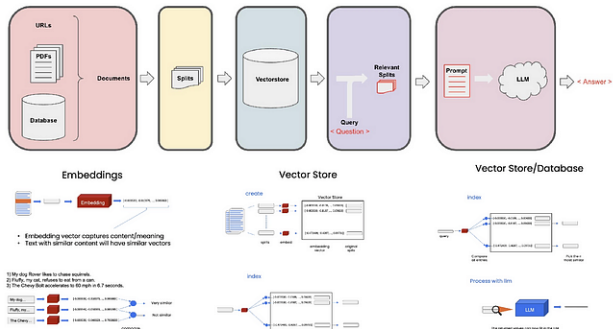


103



3





TeeTracker

Chat with your PDF (Streamlit Demo)

Conversation with specific files

4 min read · Sep 15

56

...

Jayita Bhattacharyya in GoPenAI

Primer on Vector Databases and Retrieval-Augmented Generation...

Vector Databases Generation (RAG)
Langchain Pinecone HuggingFace Large...

9 min read · Aug 16

228

1

...

See more recommendations