大模型（LLMs）RAG 版面分析——文本分块面

来自： AiGC面试宝典

宁静致远 2024年03月19日 22:30

- 大模型（LLMs）RAG 版面分析——文本分块面
  - 一、为什么需要对文本分块?
  - 二、能不能介绍一下常见的文本分块方法?
    - 2.1 一般的文本分块方法
    - 2.2 正则拆分的文本分块方法
    - 2.3 Spacy Text Splitter 方法
    - 2.4 基于 langchain 的 CharacterTextSplitter 方法
    - 2.5 基于 langchain 的 递归字符切分 方法
    - 2.6 HTML 文本拆分 方法
    - 2.7 Mrrkdown 文本拆分 方法
    - 2.8 Python代码拆分 方法
    - 2.9 LaTex 文本拆分 方法
  - 致谢

一、为什么需要对文本分块？

使用大型语言模型（LLM）时，切勿忽略文本分块的重要性，其对处理结果的好坏有重大影响。

> 考虑以下场景：你面临一个几百页的文档，其中充满了文字，你希望对其进行摘录和问答式处理。在这个流程中，最初的一步是提取文档的嵌入向量，但这样做会带来几个问题：

- **信息丢失的风险**：试图一次性提取整个文档的嵌入向量，虽然可以捕捉到整体的上下文，但也可能会忽略掉许多针对特定主题的重要信息，这可能会导致生成的信息不够精确或者有所缺失。
- **分块大小的限制**：在使用如 OpenAI 这样的模型时，分块大小是一个关键的限制因素。例如，GPT-4 模型有一个 32K 的窗口大小限制。尽管这个限制在大多数情况下不是问题，但从一开始就考虑到分块大小是很重要的。

因此，恰当地实施文本分块不仅能够提升文本的整体品质和可读性，还能够预防由于信息丢失或不当分块引起的问题。这就是为何在处理长篇文档时，采用文本分块而非直接处理整个文档至关重要的原因。

二、能不能介绍一下常见的文本分块方法?

2.1 一般的文本分块方法

如果不借助任何包，直接按限制长度切分方案：

```
text = "我是一个名为 ChatGLM3-6B 的人工智能助手，是基于清华大学 KEG 实验室和智谱 AI 公司于 2023 年共同训练的语言模型开发的。我的目标是通过回答用户提出的问题来帮助他们解决问题。由于我是一个计算机程序，所以我没有实际的存在，只能通过互联网来与用户交流。"
chunks = []
chunk_size = 128

for i in range(0, len(text), chunk_size):
    chunk = text[i:i + chunk_size]
```

```
    chunks.append(chunk)
chunks
>>>
[
    '我是一个名为 ChatGLM3-6B 的人工智能助手，是基于清华大学 KEG 实验室和智谱 AI 公司于 2023 年共同训练的语言模型开发的。我的目标是通过回答用户提出的问题来帮助他们解决问题。由于我是一个计算机程序，所以我没有实际的存在，只能通过互联网',
    '来与用户交流。'
]
```

2.2 正则拆分的文本分块方法

- 动机：【一般的文本分块方法】能够按长度进行分割，但是对于一些长度偏长的句子，容易从中间切开；
- 方法：在中文文本分块的场景中，正则表达式可以用来识别中文标点符号，从而将文本拆分成单独的句子。这种方法依赖于中文句号、“问号”、“感叹号”等标点符号作为句子结束的标志。
- 特点：虽然这种基于模式匹配的方法可能不如基于复杂语法和语义分析的方法精确，但它在大多数情况下足以满足基本的句子分割需求，并且实现起来更为简单直接。

```
import re

def split_sentences(text):
    # 使用正则表达式匹配中文句子结束的标点符号
    sentence_delimiters = re.compile(u'[。？！；]|\n')
    sentences = sentence_delimiters.split(text)
    # 过滤掉空字符串
    sentences = [s.strip() for s in sentences if s.strip()]
    return sentences

text ="文本分块是自然语言处理（NLP）中的一项关键技术，其作用是将较长的文本切割成更小、更易于处理的片段。这种分割通常是基于单词的词性和语法结构，例如将文本拆分为名词短语、动词短语或其他语义单位。这样做有助于更高效地从文本中提取关键信息。"
sentences = split_sentences(text)
print(sentences)
>>>
#output
[
    '文本分块是自然语言处理（NLP）中的一项关键技术，其作用是将较长的文本切割成更小、更易于处理的片段',
    '这种分割通常是基于单词的词性和语法结构，例如将文本拆分为名词短语、动词短语或其他语义单位',
    '这样做有助于更高效地从文本中提取关键信息'
]
```

在上面例子中，我们并没有采用任何特定的方式来分割句子。另外，还有许多其他的文本分块技术可以使用，例如词汇化（tokenizing）、词性标注（POS tagging）等。

2.3 Spacy Text Splitter 方法

- 介绍：Spacy是一个用于执行自然语言处理（NLP）各种任务的库。它具有文本拆分器功能，能够在进行文本分割的同时，保留分割结果的上下文信息。

```
import spacy
input_text = "文本分块是自然语言处理（NLP）中的一项关键技术，其作用是将较长的文本切割成更小、更易于处理的片段。这种分割通常是基于单词的词性和语法结构，例如将文本拆分为名词短语、动词短语或其他语义单位。这样做有助于更高效地从文本中提取关键信息。"

nlp = spacy.load( "zh_core_web_sm" )
```

```
doc = nlp(input_text)
for s in doc.sents:
    print (s)
>>>
[
    '文本分块是自然语言处理（NLP）中的一项关键技术，其作用是将较长的文本切割成更小、更易于处理的片段。',
    "这种分割通常是基于单词的词性和语法结构，例如将文本拆分为名词短语、动词短语或其他语义单位。",
    "这样做有助于更高效地从文本中提取关键信息。"
]
```

2.4 基于 langchain 的 CharacterTextSplitter 方法

使用CharacterTextSplitter，一般的设置参数为：chunk_size、 chunk_overlap、separator和strip_whitespace。

```
from langchain.text_splitter import CharacterTextSplitter
text_splitter = CharacterTextSplitter(chunk_size = 35, chunk_overlap=0, separator='',
strip_whitespace=False)
text_splitter.create_documents([text])
>>>
[
    Document(page_content='我是一个名为 ChatGLM3-6B 的人工智能助手，是基于清华大学 '),
    Document(page_content='KEG 实验室和智谱 AI 公司于 2023 年共同训练的语言模型开发'),
    Document(page_content='的。我的目标是通过回答用户提出的问题来帮助他们解决问题。由于我是一个计'),
    Document(page_content='算机程序，所以我没有实际的存在，只能通过互联网来与用户交流。')
]
```

2.5 基于 langchain 的 递归字符切分 方法

使用RecursiveCharacterTextSplitter，一般的设置参数为：chunk_size、 chunk_overlap。

```
#input text
input_text = "文本分块是自然语言处理（NLP）中的一项关键技术，其作用是将较长的文本切割成更小、更易于处理的片段。这种分割通常是基于单词的词性和语法结构，例如将文本拆分为名词短语、动词短语或其他语义单位。这样做有助于更高效地从文本中提取关键信息。"

from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = 100 , #设置所需的文本大小
    chunk_overlap = 20 )

chunks = text_splitter.create_documents([input_text])
print (chunks)
>>>
[
    Document(page_content='文本分块是自然语言处理（NLP）中的一项关键技术，其作用是将较长的文本切割成更小、更易于处理的片段。这种分割通常是基于单词的词性和语法结构，例如将文本拆分为名词短语、动词短语或其他语义单位。这样做有助'),
    Document(page_content='短语、动词短语或其他语义单位。这样做有助于更高效地从文本中提取关键信息。')]
```

与CharacterTextSplitter不同，RecursiveCharacterTextSplitter不需要设置分隔符，默认的几个分隔符如下：

```
"\n\n" - 两个换行符，一般认为是段落分隔符
"\n" - 换行符
" " - 空格
"" - 字符
```

拆分器首先查找两个换行符（段落分隔符）。一旦段落被分割，它就会查看块的大小，如果块太大，那么它会被下一个分隔符分割。如果块仍然太大，那么它将移动到下一个块上，以此类推。

2.6 HTML 文本拆分 方法

- 介绍：HTML文本拆分器是一种结构感知的文本分块工具。它能够在HTML元素级别上进行文本拆分，并且会为每个分块添加与之相关的标题元数据。
- 特点：对HTML结构的敏感性，能够精准地处理和分析HTML文档中的内容。

```
#input html string
 html_string = """
<!DOCTYPE html>
<html>
<body>
    <div>
        <h1>Mobot</h1>
        <p>一些关于Mobot的介绍文字。</p>
        <div>
            <h2>Mobot主要部分</h2>
            <p>有关Mobot的一些介绍文本。</p>
            <h3>Mobot第1小节</h3>
            <p>有关Mobot第一个子主题的一些文本。</p>
            <h3>Mobot第2小节</h3>
            <p>关于Mobot的第二个子主题的一些文字。</p>
        </div>
        <div>
            <h2>Mobot</h2>
            <p>关于Mobot的一些文字</p>
        </ div>
        <br>
        <p>关于Mobot的一些结论性文字</p>
    </div>
</body>
</html>
"""

 headers_to_split_on = [
    ( "h1" , "Header 1" ),
    ( "h2" , "标题 2" ),
    ( "h3" , "标题 3" ),
]

from langchain.text_splitter import HTMLHeaderTextSplitter
html_splitter = HTMLHeaderTextSplitter(headers_to_split_on=headers_to_split_on)

html_header_splits = html_splitter.split_text(html_string)
print(html_header_split)
>>>
[
    Document(page_content='Mobot'),
    Document(page_content='一些关于Mobot的介绍文字。\nMobot主要部分 Mobot第1小节 Mobot第2小节', metadata={'Header 1': 'Mobot'}),
    Document(page_content='有关Mobot的一些介绍文本。', metadata={'Header 1': 'Mobot', '标题 2': 'Mobot主要部分'}),
    Document(page_content='有关Mobot第一个子主题的一些文本。', metadata={'Header 1': 'Mobot', '标题 2': 'Mobot主要部分', '标题 3': 'Mobot第1小节'}),
```

```
    Document(page_content='关于Mobot的第二个子主题的一些文字。', metadata={'Header 1': 'Mobot', '标题 2': 'Mobot主要部分', '标题 3': 'Mobot第2小节'}),
    Document(page_content='Mobot div>', metadata={'Header 1': 'Mobot'}),
    Document(page_content='关于Mobot的一些文字  \n关于Mobot的一些结论性文字', metadata={'Header 1': 'Mobot', '标题 2': 'Mobot'})
]
```

仅提取在header_to_split_on参数中指定的HTML标题。

2.7 Mrrkdown 文本拆分 方法

- 介绍：Markdown文本拆分是一种根据Markdown的语法规则（例如标题、Bash代码块、图片和列表）进行文本分块的方法。
- 特点：具有对结构的敏感性，能够基于Markdown文档的结构特点进行有效的文本分割。

```
markdown_text = '# Mobot\n\n ## Stone\n\n这是python  \n这是\n\n ## markdown\n\n 这是中文文本拆分'

from langchain.text_splitter import MarkdownHeaderTextSplitter
headers_to_split_on = [
    ("#", "Header 1"),
    ("##", "Header 2"),
    ("###", "Header 3"),
]

markdown_splitter = MarkdownHeaderTextSplitter(headers_to_split_on=headers_to_split_on)
md_header_splits = markdown_splitter.split_text(markdown_text)
print(md_header_splits)
>>>
[
    Document(page_content='这是python\n这是', metadata={'Header 1': 'Mobot', 'Header 2': 'Stone'}),
    Document(page_content='这是中文文本拆分', metadata={'Header 1': 'Mobot', 'Header 2': 'markdown'})
]
```

MarkdownHeaderTextSplitter 能够根据设定的 headers_to_split_on 参数，将 Markdown 文本进行拆分。这一功能使得用户可以便捷地根据指定的标题将 Markdown 文件分割成不同部分，从而提高编辑和管理的效率。

2.8 Python代码拆分 方法

```
python_text = """
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

for i in range(10):
    print (i)
"""

from langchain.text_splitter import PythonCodeTextSplitter
python_splitter = PythonCodeTextSplitter(chunk_size=100, chunk_overlap=0)
python_splitter.create_documents([python_text])
>>>
[
    Document(page_content='class Person:\n  def __init__(self, name, age):\n    self.name = name\n self.age = age'),
```

```
    Document(page_content='p1 = Person("John", 36)\n\nfor i in range(10):\n    print (i)')
]
```

2.9 LaTex 文本拆分 方法

LaTex文本拆分工具是一种专用于代码分块的工具。它通过解析LaTex命令来创建各个块，这些块按照逻辑组织，如章节和小节等。这种方式能够产生更加准确且与上下文相关的分块结果，从而有效地提升LaTex文档的组织和处理效率。

```
#input Latex string
latex_text = """documentclass{article}begin{document}maketitlesection{Introduction}大型语言模型 (LLM)是一种机器学习模型，可以在大量文本数据上进行训练，以生成类似人类的语言。近年来，法学硕士在各种自然语言处理任务中取得了重大进展，包括语言翻译、文本生成和情感分析。subsection{法学硕士的历史}最早的法学硕士是在 20 世纪 80 年代开发的和 20 世纪 90 年代，但它们受到可处理的数据量和当时可用的计算能力的限制。然而，在过去的十年中，硬件和软件的进步使得在海量数据集上训练法学硕士成为可能，从而导致subsection{LLM 的应用}LLM 在工业界有许多应用，包括聊天机器人、内容创建和虚拟助理。它们还可以在学术界用于语言学、心理学和计算语言学的研究。end{document}"""

from langchain.text_splitter import LatexTextSplitter
Latex_splitter = LatexTextSplitter(chunk_size= 100 , chunk_overlap= 0 )

latex_splits = Latex_splitter.create_documents([latex_text])
print (latex_splits)
>>>
[
    Document(page_content='documentclass{article}begin{document}maketitlesection{Introduction}大型语言模型 (LLM)'),
    Document(page_content='是一种机器学习模型，可以在大量文本数据上进行训练，以生成类似人类的语言。近年来，法学硕士在各种自然语言处理任务中取得了重大进展，包括语言翻译、文本生成和情感分析。subsection{法学硕士的历史'),
    Document(page_content='}最早的法学硕士是在'),
    Document(page_content='20 世纪 80 年代开发的和 20 世纪 90'),
    Document(page_content='年代，但它们受到可处理的数据量和当时可用的计算能力的限制。然而，在过去的十年中，硬件和软件的进步使得在海量数据集上训练法学硕士成为可能，从而导致subsection{LLM 的应用}LLM'),
    Document(page_content='在工业界有许多应用，包括聊天机器人、内容创建和虚拟助理。它们还可以在学术界用于语言学、心理学和计算语言学的研究。end{document}')
]
```

在上述示例中，我们注意到代码分割时的重叠部分设置为0。这是因为在处理代码分割过程中，任何重叠的代码都可能完全改变其原有含义。因此，为了保持代码的原始意图和准确性，避免产生误解或错误，设置重叠部分为0是必要的。当你决定使用哪种分块器处理数据时，重要的一步是提取数据嵌入并将其存储在向量数据库（Vector DB）中。上面的例子中使用文本分块器结合 LanceDB 来存储数据块及其对应的嵌入。LanceDB 是一个无需配置、开源且无服务器的向量数据库，其数据持久化在硬盘驱动器上，允许用户在不超出预算的情况下实现扩展。此外，LanceDB 与 Python 数据生态系统兼容，因此你可以将其与现有的数据工具（如 pandas、pyarrow 等）结合使用。