XMind
试用模式

LangChain学习框架

Prompts（提示）

- 基本概念
  - 定义
    - 1. 一种可重复生成提示的方法。它包含以下内容：文本字符串（“模板”），可以接收输入并返回一个给LLM的提示。
    - 2. 它还可以包含：对LLM的指令；一组少量示例，以帮助语言模型生成更好的回复；针对语言模型的问题。
  - 作用
    - 1. 有时候，您可能需要创建依赖于多个输入的提示。
    - 2. Prompt Template使得代码更简洁、更易于管理。
- Example Selector（示例选择器）
  - 作用
    - 允许选择在提示中包含的示例。
    - 在提示中提供示例可以改善LLM返回的回答。
  - 选择
    - 基于长度的示例选择器
      - 基于长度进行选择示例，用于确定使用哪些示例（当您担心提示过长时很有用）。
    - 最大边际相关性示例选择器
      - 基于示例与输入之间的相似性以及多样性的组合来选择示例（使用嵌入向量）。
    - n-gram重叠示例选择器
      - 根据n-gram重叠得分选择和排序示例，使得示例与输入最相似。
    - 相似性示例选择器
      - 基于示例与输入之间的相似性选择示例（使用嵌入向量）。
- Output Parsers（输出解析器）
  - 作用
    - 你希望从LLM获取的输出是结构化的。
  - 方法
    - get_format_instructions()
      - 返回一个包含指示语言模型输出格式的字符串。
    - parse(str)
      - 接收一个字符串并将其解析为某种结构。
  - 其他类型
    - CommaSeparatedListOutputParser（逗号分隔列表输出解析器）
    - OutputFixingParser（输出修复解析器）
    - Structured Output Parser（结构化输出解析器）

Index（索引）

- 基本概念
  - 索引最常见的使用方式是什么
    - 索引在链式方法中通常用于“检索”步骤。这一步骤涉及接收用户的查询并返回最相关的文档。
  - 定义
    - 索引指的是对文档进行结构化处理，以便LLM（语言模型）能够更好地与其交互。
  - 其他
    - 1. 索引除了用于检索之外，还可以用于其他用途。
    - 2. 检索可以使用除索引之外的其他逻辑来查找相关的文档。
- Document Loaders（文档加载器）
  - 定义
    - 文档加载器使得从各种来源轻松创建文档成为可能。然后可以将这些文档加载到向量存储中，以从源加载文档。
  - 方法
    - 加载
      - load方法将从源加载文档，并将它们作为文档数组返回。
    - 加载并分割
      - loadAndSplit方法将从源加载文档，并使用提供的TextSplitter对它们进行拆分，然后将它们作为文档数组返回。
- Text Splitters（文本切割器）
  - 为什么需要文本切割器
    - 语言模型通常受到可以传递给它们的文本数量的限制。因此，有必要将它们拆分成较小的块。
    - 使用文本拆分器还可以提高向量存储搜索的结果，因为较小的块有时更容易匹配查询。
  - 主要方法
    - 创建文档
      - 接受一个原始文本字符串列表，并返回一个文档列表。
    - 切分文档
      - 接受一个文档列表，并返回一个文档列表。
- VectorStore（向量存储）
  - 定义
    - 一种特定类型的数据库，针对存储文档及其嵌入进行优化，以便为特定查询获取最相关的文档。
    - 您可以从文档列表或文本列表及其相应元数据创建向量存储。您还可以从现有索引创建向量存储。
  - 示例
    - Pinecone
    - Chroma
    - FAISS
- Retrievers（检索器）
  - 定义
    - 一种存储数据的方式，可以通过语言模型进行查询。该对象必须暴露的唯一接口是一个getRelevantDocuments方法，该方法接受一个查询字符串，并返回一个文档列表。
    - 该接口暴露一个get_relevant_documents方法，该方法接受一个查询（一个字符串）并返回一个文档列表。

Language model（语言模型）

- Emeddings（嵌入）
  - 定义
    - 嵌入（Embeddings）将文本转换为一组数字。
    - 这使我们能够通过比较这些数字组的相似性来找到具有相似含义的文本片段。
  - 提供商
    - OpenAI
    - Cohere
    - HuggingFace
  - 主要方法
    - embed_documents
    - embed_query
- LLM（大语言模型）
  - 定义
    - 一个超级智能的计算机程序，能够理解和生成文本。
    - 它可以根据所学到的信息回答问题或完成任务。
  - LangChain提供LLM了吗
    - Langchain不提供LLM（语言模型）。Langchain提供了一种与您选择的LLM轻松交互的方式。
  - LLM的类型
    - 闭源
      - OpenAI GPT
    - 开源
      - LLaMA
      - BLOOM
      - StableLM
      - 20+其他
- Chat Model（聊天模型）
  - 消息类型
    - 系统消息
      - 设置AI应遵循的目标的信息
    - 人类消息
      - 从人类的角度发送的信息
    - AI消息
      - 从人类正在交互的AI的角度发送的信息
  - 常规LLM和Chat模型之间的差异是什么呢？
    - Chat模型在底层使用了LLM（大型语言模型），但输入/输出却有所不同。
    - 他们不是提供“文本输入，文本输出”的API，而是暴露了一个接口，其中“聊天消息”是输入和输出。
  - 什么是不同的聊天消息模板？
    - 人类消息提示模板
      - 一个公开方法的类，用于接收用户输入并返回HumanMessage（人类消息）。
    - AI消息提示模板
      - 一个公开方法的类，用于接收用户输入并返回AIMessage（AI消息）。
    - 系统消息提示模板
      - 一个公开方法的类，用于接收用户输入并返回SystemMessage（系统消息）。
    - 聊天消息提示模板
      - 一个公开方法的类，用于接收用户输入并返回ChatMessage（聊天消息）。

Memory（记忆）

- 如何给代理添加记忆
  - 1. 创建带有记忆的LLMChain
  - 2. 使用LLMChain创建自定义代理。
- 如何给链条添加记忆
  - 1. 设置提示和记忆
  - 2. 初始化LLMChain
  - 3. 调用LLMChain
- 基于VectorStore的记忆
  - 将记忆存储在VectorDB中
  - 查询与前K个最相关文档
  - 优点：不需要显式追踪消息的顺序；AI可以记住之前告诉它的相关信息
- ConversationBufferMemory（对话缓冲记忆）
  - Memory允许存储消息
  - 将消息提取到变量中
  - 优点：易于理解和使用
- ConversationBufferWindowMemory（对话缓冲窗口记忆）
  - 仅使用最近的K个消息
  - 优点：有助于保持记忆历史的简洁性
- ConversationSummaryMemory（对话摘要记忆）
  - 随着时间的推移，创建对话摘要
  - 优点：有助于在时间上压缩对话中的信息

Chains（链）

- 基本信息
  - 什么是Chain?
    - 链式方法是易于重复使用的组件，可以相互链接在一起。
    - 组件可以是提示（prompts）、模型（models）、任意函数或其他链式方法
  - 为什么需要Chain
    - 将语言模型与其他信息源或第三方API结合在一起通常是很有用的
- LLM Chain（大模型链）
  - 定义
    - 1. 最常见的链式方法类型。
    - 2. 接受输入，将格式化字符串传递给LLM并返回LLM的输出。
  - 组成
    - 1. PromptTemplate（提示模板）
    - 2. LLM模型
    - 3. 可选的输出解析器
- 其他Chain
  - ConversationalRetrievalChain
    - 与带有聊天历史的对话文档进行交流的链式方法（与检索型问答不同之处在于该链式方法包含聊天历史）
  - RetrivalQA
    - 在一系列文档上进行问答
  - Vector DB Text Generation
    - 这对于以下情况非常有用：
    - 1. 从向量数据库中获取文档
    - 2. 使用LLMChain生成文本，以从第一步中获取的文档作为输入。
  - BashChain
    - 用于运行文件系统命令的链式方法。
  - LLM Math
    - 用于解决复杂的数学问题的链式方法
  - LLMRequestsChain
    - 1. 使用Python的requests库从URL获取HTML结果。
    - 2. 使用LLM解析结果
  - 还有更多...
- Sequential Chain（顺序链）
  - 定义
    - 一系列链式方法，按照一定的顺序调用
  - 类型
    - SimpleSequentialChain（简单顺序链式方法）
      - 每个步骤具有单一的输入/输出，一个步骤的输出是下一个步骤的输入
    - SequentialChain（顺序链式方法）
      - 允许多个输入/输出。
- Index Chains（索引链）
  - 定义
    - 将您自己的数据（存储在索引中）与LLM（语言模型）相结合。其中最好的例子是利用LLM在您自己的文档上进行问答。
  - 将多个文档传递给语言模型的方法
    - Stuffing
      - 最简单的方法是将所有相关数据直接作为上下文填充到提示中，以传递给语言模型。
      - 优势
        - 在生成文本时，只需对LLM进行单次调用，LLM将同时访问所有的数据。
      - 劣势
        - 大多数LLM（语言模型）都有上下文长度的限制，对于大型文档（或多个文档），直接将其全部作为上下文会导致提示超过上下文长度的限制，从而无法正常工作。
    - Map Reduce
      - 这种方法涉及对每个数据块运行初始提示（对于摘要任务，可以是该块的摘要；对于问答任务，可以是仅基于该块的答案）。然后，运行另一个提示来组合所有初始输出。
      - 优势
        - 这种方法可以扩展到比“StuffDocumentsChain”更大的文档（以及更多的文档）。对各个文档的LLM调用是独立的，因此可以进行并行处理。这意味着可以同时处理多个文档，提高处理效率。
      - 劣势
        - 相比于“StuffDocumentsChain”方法，这种方法需要更多对LLM的调用。在最终的组合调用过程中会丢失一些信息。
    - Refine
      - 这种方法涉及在第一个数据块上运行初始提示，生成一些输出。对于剩余的文档，将该输出与下一个文档一起传递给LLM，并要求LLM根据新文档对输出进行改进。
      - 优势
        - 这种方法可以获取更相关的上下文，并且可能比“MapReduceDocumentsChain”方法损失更少的信息。
      - 劣势
        - 这种方法需要比“StuffDocumentsChain”更多对LLM的调用。这些调用也不是独立的，意味着无法像“MapReduceDocumentsChain”那样进行并行处理。此外，文档的顺序可能存在一些潜在的依赖性。
    - Map-Rerank
      - 这种方法涉及在每个数据块上运行初始提示，该提示不仅尝试完成任务，还给出了对其答案的确定程度得分。然后，根据这个得分对响应进行排名，并返回最高得分的答案。
      - 优势
        - 与“MapReduceDocumentsChain”方法相比，这种方法具有类似的优点。优势在于需要较少的调次数。
      - 劣势
        - 这种方法无法在文档之间进行信息的组合。这意味着当您预期在单个文档中存在单个答案时，它会非常有用。

Agent（代理）

- 基本概念
  - 定义
    - 代理（Agents）通常指的是利用语言模型作为推理引擎，并将其与两个关键组成部分连接起来：工具和记忆。
- Tools（工具）
  - 定义
    - 工具（Tools）帮助将语言模型与其他数据源或计算连接起来。
  - 为什么需要工具
    - 由于LLM只具有其训练数据的知识，我们可以使用工具来填补这一差距。
  - 工具可以做什么
    - 工具帮助获取最新的数据，并将其作为上下文插入到提示中。
    - 工具还可以用于执行操作（例如运行代码、修改文件等），并将其输出用于做出更多决策的代理。
  - 示例
    - Google搜索、Wolfram Alpha、Zapier等是30多种可用的工具之一。此外，您还可以创建自定义工具。
- Memory（记忆）
  - 定义
    - 记忆（Memory）帮助代理记住先前的交互。它使代理能够具有更多上下文来处理下一个请求。
- Agent不同类型
  - ReAct（Resoning and Acting）
    - 1. 用户给代理一个任务
    - 2. 思考：代理“思考”应该做什么
    - 3. 动作/动作输入：代理决定采取什么行动（也就是使用什么工具），以及该工具的输入应该是什么
    - 4. 观察：工具的输出。
    - 5. 重复步骤2-4，直到代理认为任务完成
  - AutoGPT
    - AutoGPT和普通的Langchain代理有什么区别
      - 记忆模式
        - 在AutoGPT中，如何处理记忆？
          - 1. AutoGPT是一个更长期运行的过程，无法将完整的代理步骤列表传递给LLM调用。
          - 2. 它在中间代理步骤上使用基于检索的记忆。
          - 3. 在底层，这种基于检索的记忆使用向量存储在嵌入中进行语义搜索。
        - 在Langchain中，如何处理记忆？
          - 1. 代理步骤的记忆是通过保留与该任务相关的中间代理步骤的列表，并将完整列表传递给LLM调用来实现的。
          - 2. 系统的记忆是通过记住最终的输入和输出（但遗忘中间的代理步骤）来实现的。
      - 执行
      - 目的
        - 在AutoGPT中，目标通常更加开放且持续时间较长
  - BabyAGI
    - 这个项目的创新之处是什么？
      - 与传统的Langchain代理相比，BabyAGI具有独立的规划和执行步骤，它一次性规划了一系列动作（而不仅仅是下一个动作）。
    - 它与AutoGPT有何相似之处？
      - 1. BabyAGI将基于检索的记忆应用于中间的代理工具步骤。
      - 2. BabyAGI设计用于处理更长期运行的任务。
    - BabyAGI是如何工作的？
      - 1. BabyAGI明确规划了一系列动作
      - 2. 然后，它执行第一个动作，然后使用该结果进行另一个规划步骤并更新任务列表。
    - 它如何比传统的Langchain代理表现更好？
      - 传统的LangChain代理有时在几个步骤之后会忘记其最初的目标，因此提前规划所有步骤可以帮助解决这个问题。
- Agent Executor（代理执行器）
  - 在一个while循环中运行代理的逻辑，直到满足某个停止条件。

原文：TW：@pwang_szn / 翻译：WB：PotatO的小号嘻嘻

@PotatO的小号嘻嘻