## 3.2 提示

一切都始于提示。

事实上,Langchain 的一切都是围绕提示构建的,提示对于您使用 Langchain 所做的一切至关重要。

让我们稍微谈谈 提示是什么以及在 Langchain 中的提示模板的概念。

我们跟聊天机器人沟通的唯一方式是输入 Prompts。

什么是 提示？提示指的是模型的输入, 我们发给大语言模型的内容。

在 Langchain 中,提示基本上是让模型在某些输入文本的**条件限制**下生成文本的过程。这些模型通常只是继续预测下一个标记,通常是下一个单词,然后通过这种方式组装文本。

提示本身对输出结果有很大的影响,以至于这导致了一个全新的领域,人们现在称之为 提示工程。

早期的提示非常简单,您可能会问一些简单的问题,比如乾隆是什么时候皇帝? 希望大型语言模型在其权重中学到足够的知识来回答这个问题,并生成一个输出结果。

随着模型的不断改进, 提示从这些简单的形式转变为更加复杂的形式,现在我们看到人们给出了包含整个上下文和各种信息的提示。

从GPT论文中,我们可以看到其中有不同类型的提示,论文中包含了许多不同的提示。

您还可以看到这个示例中我们有所谓的样本（few shot）模板,即我们不仅仅告诉提示一些内容,还会给它一些示例,然后让它生成一个新的答案或示例。

在 Langchain 中,所有这些都由提示模板来处理,因此了解提示的不同部分非常重要,因为您将在提示中构建不同的部分。

在 Langchain 中,您将构建一个提示,并在其中注入不同的内容。例如,这是一个简单的提示,以一个上下文开始,告诉模型您是一个起名字助手,可以帮助用户根据所提供的业务描述创建业务名称,然后给出一些示例,以特殊的方式给出这些示例,这样模型就可以学习生成与示例相似的内容。然后,我们可以传递问题或任务,并为输出结果进行预处理。

不仅 Langchain 中经常使用, 也在许多其他提示系统中使用。这些 提示系统, 基本上是在占位符注入一些用户输入的信息,这是用于回复的一些要点,再次注入一些信息。把 提示发给语言模型,语言模型给我们一个回答。

这样可以针对许多不同的任务进行操作,可以用于分类,也可以用于其他各种不同的任务。

在实际的应用开发中, 开发者一定是想要动态插入提示,而不是硬编码的。还想要有多个灵活的值,而不只有一句提示。LangChain 提供了 PromptTemplate 基本类对象来构建使用多个值的提示。

LangChain 还提供了多个类和函数,构建了很多不同实例的 PromptTemplate, 让开发者使用提示更加容易,满足这些实际开发的多样需求。

这一章的重要概念包括提示模板、输出解析器、示例选择器和控制提示长度。

我们首先来理解提示模板 PromptTemplate。

## 3.2.1 提示模板

实际上, 提示模板是一个 “未完成” 的输入示例。Langchain 提供了我们通过告诉它要替换占位符的值来创建格式化之后的提示。Langchain 内置的提示模板对象 (PromptTemplate), 使我们能够将用户输入插入到提示文本的占位符中，并且可以增加额外的知识,额外的上下文, 将用户输入进行格式化，最终提供给语言模型高质量的提示,从而获得更好的输出结果。

提示模板的定义

在Langchain中，提示模板是一种可复制生成提示的方式。它包含一个文本字符串（即”模板”），该模板可以接收用户的一系列参数，从而生成一个提示。比如说，我们要给公司的产品取一个好听的名字，用户输入的就是产品的品类名字，例如“袜子”或“毛巾”。然而，我们并不需要为每一个品类都编写一个提示，而是使用这个模板，根据用户输入的不同品类，生成对应的提示。

模板的三个组成部分

提示模板主要由三个部分组成：语言模型的说明，一组少数示例，以及一个问题或任务。这三部分中，语言模型的说明和示例是固定的常量，而问题或任务是可以由用户改变的变量。Langchain将这三个部分组合起来，并为输出结果进行格式化的处理，以生成一个完整的提示。

提示模板在Langchain中得到了广泛的应用，但也在许多其他的大语言模型提示系统中使用。我们常常能看到类似的情况。Langchain的提示模板在其他模型中是通用的。

提示模板有许多实际应用场景。比如，它在做摘要或分类时非常实用。以新闻摘要为例，我们可以构建一个模板，模型的任务就是生成新闻的摘要。用户的输入可能是复制黏贴了一篇完整的新闻文章，给语言模型的说明可能是“生成这篇新闻的摘要”，示例则可以是一些已经生成过的新闻摘要。然后，Langchain将这三个部分组合起来，生成一个完整的提示，如：“请为这篇新闻生成一个摘要。” 根据这个提示，模型就能生成一个符合要求的新闻摘要。这样，我们就可以利用提示模板，轻松地为任意一篇新闻生成摘要。

一个最简单的示例

这是最基本的示例,这个提示模板没有包含示例,但它使我们能够动态更改带有用户输入的提示。

from langchain import PromptTemplate  
  
template = """  
You are an expert data scientist with an expertise in building deep learning models.   
Explain the concept of {concept} in a couple of lines  
"""  
  
prompt = PromptTemplate(  
 input\_variables=["concept"],  
 template=template,  
)  
  
prompt.format(concept="NLP")

当我们传入 concept="NLP" 时, PromptTemplate 方法,将 “NLP” 注入到模板中

我们得到的结果是:

'\nYou are an expert data scientist with an expertise in building deep learning models. \nExplain the concept of NLP in a couple of lines\n'

这个代码示例将用户输入变量 input\_variables和开发者定义的 template 传参给 PromptTemplate对象, 然后调用 format 方法，格式化为语言模型可用的字符串提示。

构造提示模板的步骤

构造模板包含取出模板和格式化两个步骤。from\_template即取出模板，format即格式化用户输入。Langchain, 内置了很多模板，比如聊天模型的提示模板，角色的提示模板。格式化可以是普通字符串用作通用语言模型输入，也可以是用作聊天模型输入的Message对象。

上面这个代码还可以简化为：

# 导入PromptTemplate和template定义同上  
  
prompt = PromptTemplate.from\_template(template).format(concept="NLP")  
prompt

这段代码可以用自然语言描述为：取出内置模式给template使用，传参用户的输入后格式化输出完整提示。无论是哪一种提示模板，都是基于 PromptTemplate 对象，我们基于这个对象构造自己的提示模板。

内置提示模板，是Langchain简化开发的方式。即使你不指定 input\_variables , Langchain 会自行推断，只要你告诉了它取出哪个模式。

专用聊天模型的提示模板

专用聊天模型的提示模板的不同之处，输入的是消息列表, 支持输出 Message 对象。Langchain 的优势在于提供了聊天提示模板 （ChatPromptTemplate）和角色消息提示模板。角色消息提示模板包含AIMessagePromptTemplate, SystemMessagePromptTemplate 和 HumanMessagePromptTemplate 三种角色消息提示模板。

无论看起来这些多么复杂，他们都遵循构造模板的步骤，取模式和格式化。聊天模型的提示模板取是消息提示模式，格式化多了消息列表的选择。

我们将上一个示例代码改造为聊天提示模板，下面是代码示例：

先导入消息提示模板：

from langchain.prompts import (  
 ChatPromptTemplate,  
 PromptTemplate,  
 SystemMessagePromptTemplate,  
 AIMessagePromptTemplate,  
 HumanMessagePromptTemplate,  
)

改造思路是生成人类和系统的消息提示，通过ChatPromptTemplate整合为消息列表提示： 先使用 from\_template 取出消息模式，传入定义的template模板字符串，生成人类和系统两种模板。

from langchain import PromptTemplate  
  
template = """  
You are an expert data scientist with an expertise in building deep learning models.   
"""  
  
system\_message\_prompt = SystemMessagePromptTemplate.from\_template(template)  
human\_template="Explain the concept of {concept} in a couple of lines"  
human\_message\_prompt = HumanMessagePromptTemplate.from\_template(human\_template)

然后转化为聊天提示模板类型：

chat\_prompt = ChatPromptTemplate.from\_messages(system\_message\_prompt, human\_message\_prompt])

最后指定输出为消息类型的格式, 字符串的输出则是这样：

chat\_prompt.format(concept="NLP")

打印输出的是：

'System: \nYou are an expert data scientist with an expertise in building deep learning models. \n\nHuman: Explain the concept of NLP in a couple of lines'

指定格式化为消息对象类型则是这样：

chat\_prompt.format\_prompt(concept="NLP").to\_messages()

打印输出的是：

[SystemMessage(content='\nYou are an expert data scientist with an expertise in building deep learning models. \n', additional\_kwargs={}),  
HumanMessage(content='Explain the concept of 强化学习 in a couple of lines', additional\_kwargs={}, example=False)]

尽管上述示例并没有展示如何加入示例，但请不要忧虑，接下来我们即将进入下一章节：少样本提示模板。这个章节将全面展示如何利用示例来优化我们的模板。

事实上，所有的提示模板都基于示例选择这一概念。为了从大语言模型中获取更加完整和准确的结果，少量示例学习（Few Shot Learning）就显得尤为重要。下一章少样本提示模板，就是在构造提示时，我们提供一些示例来指导模型的输出。这些示例可以帮助模型更好地理解任务要求，从而生成更加贴合用户预期的答案。这也正是FewShotPromptTemplate的价值所在：通过使用少量示例学习，我们能够有效提升模型的输出质量，让模型给出的回答更加准确和具有针对性。

## 3.2.2 少样本提示模板

少样本提示模板 (FewShotPromptTemplate)已经被证明是很有用的。通常用户只要输入关键信息，就可以获得预期满意的回答。这些示例可以硬编码，但如果动态选择，会更强大。Langchain 提供了少样本提示模板接受用户输入，然后返回包含示例列表的提示。

在这篇文章中，我们将详细探讨如何通过调整和增强提示，特别是加入一些具体的例子，让模型通过学习这些例子来提升自身的输出质量。你会惊讶地发现，这种微小的调整会对模型输出的结果产生巨大的影响。

首先，我们需要明白一点，这与我们之前所做的工作并无太大区别，其核心仍然是制作一个优秀的提示模板。现在，让我们通过一个具体的例子来理解这个过程。

假设我们现在的任务是让模型进行反义词接龙。在这个任务中，我们会给模型一个词，然后期望模型返回这个词的反义词。因此，我们需要提供一些示例，例如 “happy” 对应的反义词是 “sad”，“tall” 的反义词是 “short”，以此类推。然后，我们通过LangChain来设置我们的提示模板。

首先，我们会设置一个前缀，这样可以方便我们逐步构建提示的各个部分，以确保它们能完整地结合在一起。然后，我们会为模型设置一些标准的示例，以帮助模型理解任务需求。接下来，我们会设置一个Few Shot提示模板，然后在这里传入我们的示例。我们还会为此设置一个前缀和一个后缀，这样可以帮助模型更好地理解任务。

然后，我们运行代码，看看模型能否正确地生成我们期望的结果。例如，如果我们输入”big”，模型就应该返回”small”。这就是我们期望看到的反义词。

实际上，我们并没有看到整个提示，因为在大多数情况下，我们并不希望向用户展示完整的提示。因此，这就是使用提示模板制作提示的一种简单方法。

在这个过程中，标准提示模板和Few Shot提示模板都发挥了重要的作用。要注意的是，Few Shot提示模板的关键在于示例，因此你可以在这里使用一系列示例。这个示例只需要展示输入是什么，以及你期望的输出是什么。

我们通过下面代码，创建一个包含几个示例的少样本提示模板,来解释 FewShotPromptTemplate 的作用。

pip install openai langchain

导入 PromptTemplate, FewShotPromptTemplate 对象

from langchain.prompts.few\_shot import FewShotPromptTemplate  
from langchain.prompts.prompt import PromptTemplate

创建一个包含几个示例的列表

examples = [  
 {"word": "happy", "antonym": "sad"},  
 {"word": "tall", "antonym": "short"},  
]

实例化 PromptTemplate 对象

example\_prompt\_format = """Word: {word}  
Antonym: {antonym}  
"""  
example\_prompt = PromptTemplate(input\_variables=["word","antonym"], template=example\_prompt\_format)  
  
print(example\_prompt.format(\*\*examples[0]))

实例化 FewShotPromptTemplate 对象

我们创建 FewShotPromptTemplate 对象，传入示例、示例格式化器、前缀、命令和后缀，这些都在指导 LLM 的输出。

此外，我们还可以提供输入变量 examples, example\_prompt 和分隔符 example\_separator=“”，用于将示例与前缀 prefix 和后缀 suffix 分开。

few\_shot\_prompt = FewShotPromptTemplate(  
 examples=examples,  
 example\_prompt=example\_prompt, # 上一步实例化PromptTemplate对象  
 prefix="Give the antonym of every input\n",  
 suffix="Word: {input}\nAntonym: ",  
 input\_variables=["input"],  
 example\_separator="\n",  
)

我们可以生成一个提示，它看起来像这样。

print(few\_shot\_prompt.format(input="big"))  
  
Give the antonym of every input  
  
Word: happy  
Antonym: sad  
  
Word: tall  
Antonym: short  
  
Word: big  
Antonym:

FewShotPromptTemplate 是一种非常有用的范例，可以控制 LLMs 的输出并引导其响应。

在实际的应用开发中,我们的示例相当复杂, 现实是我们在提示中还要加入大量的历史聊天信息,可是大型语言模型可以传递的标记数量是有限的，大多数模型最多可以传递1024到16000个标记。

Langchain 提供了控制提示长度的方法, 有兴趣的读者可以在官方文档中查看 LengthBasedExampleSelector 和 SemanticSimilarityExampleSelector 两种高阶的示例选择器, Langchain 还支持自定义示例选择器, 自己动手做一个示例选择器也是个不错的选择。

## 3.2.3 扩展提示模板

LangChain提供了极其灵活的提示模板方法和组合提示的方式，满足各种开发需求。在所有的这些方法中，基础模板和少样本提示模板是最基础的，其他所有的方法都在此基础上进行扩展。

LangChain提供了一套默认的提示模板，可以生成适用于各种任务的提示。然而，可能会出现默认提示模板无法满足你的需求的情况。例如，你可能需要创建一个带有特定动态指令的提示模板。在这种情况下，LangChain 支持你可以创建一个自定义的提示模板。

为了个性化LLM应用，你可能需要将LLM与特定用户的最新信息进行组合。特征库可以很好地保持这些数据的新鲜度，而LangChain提供了一种方便的方式，可以将这些数据与LLM进行组合。做法是从提示模板内部调用特征库，检索值，然后将这些值格式化为提示。

针对聊天模型需求，LangChain提供了不同类型的消息提示模板。最常用的是AIMessagePromptTemplate，SystemMessagePromptTemplate和HumanMessagePromptTemplate，它们分别创建一个AI消息、系统消息和人类消息。

此外，LangChain还支持”部分”提示模板，也就是说，传入一部分所需的值，以创建一个只期望剩余子集值的新提示模板。LangChain以两种方式支持这一点：部分格式化字符串值和部分格式化返回字符串值的函数。这两种不同的方式支持不同的用例。

我们可以通过PipelinePrompt来组合多个提示。这在你希望重用部分提示时非常有用。

通常，将提示存储为文件，而不是Python代码，更为方便。这样可以方便地共享、存储和版本控制提示。LangChain中支持的文件类型有JSON 和 YAML, 框架理论上 LangChain 想要支持一切的文件类型。

## 3.2.4 示例提示选择器

示例提示对模型输出结果产生的影响是非常显著的。这点在实际操作中的反馈是明确无误的。但问题在于，我们可能有大量这样的示例，我们不可能全部输入给模型。而且，尝试适应所有示例可能会很快变得非常昂贵，尤其是在计算资源和时间上。这就是示例选择器发挥作用的地方，它帮助我们选择最适合的示例来提示模型。

以金融财报的摘要为例，如果举例的摘要过长，成本会变得很昂贵。大量、冗长的例子可能会占用模型可处理的 Token 数量，这使得模型无法充分理解和处理用户的真正输入，从而影响输出质量。

幸运的是，LangChain 的示例提示选择器（Example Selector）提供了一套工具，来解决这个问题。这些工具能基于策略选择合适的例子，如根据例子的长度、输入与例子之间的n-gram重叠分数来评估其相似度打分、找到与输入具有最大余弦相似度的例子, 以及多样性等因素来选择例子, 从而保持提示成本的相对稳定。

根据长度选择示例，是很普遍和现实的需求，以下是根据长度选择示例的代码：

from langchain.prompts import PromptTemplate  
from langchain.prompts import FewShotPromptTemplate  
from langchain.prompts.example\_selector import LengthBasedExampleSelector  
  
examples = [  
 {"word": "happy", "antonym": "sad"},  
 {"word": "tall", "antonym": "short"},  
 {"word": "energetic", "antonym": "lethargic"},  
 {"word": "sunny", "antonym": "gloomy"},  
 {"word": "windy", "antonym": "calm"},  
]  
  
example\_prompt = PromptTemplate(  
 input\_variables=["word", "antonym"],  
 template="Word: {word}\nAntonym: {antonym}",  
)  
  
example\_selector = LengthBasedExampleSelector(  
 examples=examples,   
 example\_prompt=example\_prompt,   
 max\_length=25,  
)  
  
dynamic\_prompt = FewShotPromptTemplate(  
 example\_selector=example\_selector,  
 example\_prompt=example\_prompt,  
 prefix="Give the antonym of every input",  
 suffix="Word: {adjective}\nAntonym:",   
 input\_variables=["adjective"],  
)  
  
print(dynamic\_prompt.format(adjective="big"))