Custom prompt template

Let's suppose we want the LLM to generate English language explanations of a function given its name. To achieve this task, we will create a custom prompt template that takes in the function name as input, and formats the prompt template to provide the source code of the function.

Why are custom prompt templates needed?

LangChain provides a set of default prompt templates that can be used to generate prompts for a variety of tasks. However, there may be cases where the default prompt templates do not meet your needs. For example, you may want to create a prompt template with specific dynamic instructions for your language model. In such cases, you can create a custom prompt template.

Take a look at the current set of default prompt templates here.

Creating a Custom Prompt Template

There are essentially two distinct prompt templates available - string prompt templates and chat prompt templates. String prompt templates provides a simple prompt in string format, while chat prompt templates produces a more structured prompt to be used with a chat API.

In this guide, we will create a custom prompt using a string prompt template.

To create a custom string prompt template, there are two requirements:

1. It has an input_variables attribute that exposes what input variables the prompt template expects.
2. It exposes a format method that takes in keyword arguments corresponding to the expected input_variables and returns the formatted prompt.

We will create a custom prompt template that takes in the function name as input and formats the prompt to provide the source code of the function. To achieve this, let's first create a function that will return the source code of a function given its name.

```
import inspect


def get_source_code(function_name):
    # Get the source code of the function
    return inspect.getsource(function_name)
```

Next, we'll create a custom prompt template that takes in the function name as input, and formats the prompt template to provide the source code of the function.

```
from langchain.prompts import StringPromptTemplate
from pydantic import BaseModel, validator

PROMPT = """\
Given the function name and source code, generate an English language
explanation of the function.
Function Name: {function_name}
Source Code:
{source_code}
Explanation:
"""



class FunctionExplainerPromptTemplate(StringPromptTemplate, BaseModel):
    """A custom prompt template that takes in the function name as input,
and formats the prompt template to provide the source code of the
function."""

    @validator("input_variables")
    def validate_input_variables(cls, v):
        """Validate that the input variables are correct."""
        if len(v) != 1 or "function_name" not in v:
            raise ValueError("function_name must be the only
input_variable.")
        return v

    def format(self, **kwargs) -> str:
        # Get the source code of the function
        source_code = get_source_code(kwargs["function_name"])

        # Generate the prompt to be sent to the language model
        prompt = PROMPT.format(
            function_name=kwargs["function_name"].__name__,
source_code=source_code
        )
        return prompt

    def _prompt_type(self):
        return "function-explainer"
```

API Reference:

- StringPromptTemplate from `langchain.prompts`

Use the custom prompt template

Now that we have created a custom prompt template, we can use it to generate prompts for our task.

```
fn_explainer = FunctionExplainerPromptTemplate(input_variables=
["function_name"])

# Generate a prompt for the function "get_source_code"
prompt = fn_explainer.format(function_name=get_source_code)
print(prompt)
```

```
    Given the function name and source code, generate an English language
explanation of the function.
    Function Name: get_source_code
    Source Code:
    def get_source_code(function_name):
        # Get the source code of the function
        return inspect.getsource(function_name)

    Explanation:
```