LLM

An LLMChain is a simple chain that adds some functionality around language models. It is used widely throughout LangChain, including in other chains and agents.

An LLMChain consists of a PromptTemplate and a language model (either an LLM or chat model). It formats the prompt template using the input key values provided (and also memory key values, if available), passes the formatted string to LLM and returns the LLM output.

Get started

```
from langchain import PromptTemplate, OpenAI, LLMChain

prompt_template = "What is a good name for a company that makes {product}?"

llm = OpenAI(temperature=0)
llm_chain = LLMChain(
    llm=llm,
    prompt=PromptTemplate.from_template(prompt_template)
)
llm_chain("colorful socks")
```

```
    {'product': 'colorful socks', 'text': '\n\nSocktastic!'}
```

Additional ways of running LLM Chain

Aside from `__call__` and `run` methods shared by all `Chain` object, `LLMChain` offers a few more ways of calling the chain logic:

- `apply` allows you run the chain against a list of inputs:

```
input_list = [
    {"product": "socks"},
    {"product": "computer"},
    {"product": "shoes"}
]

llm_chain.apply(input_list)
```

```
    [{'text': '\n\nSocktastic!'},
     {'text': '\n\nTechCore Solutions.'},
     {'text': '\n\nFootwear Factory.'}]
```

- `generate` is similar to `apply`, except it return an `LLMResult` instead of string. `LLMResult` often contains useful generation such as token usages and finish reason.

```
llm_chain.generate(input_list)
```

```
    LLMResult(generations=[[Generation(text='\n\nSocktastic!', generation_info={'finish_reason': 'stop',
'logprobs': None})], [Generation(text='\n\nTechCore Solutions.', generation_info={'finish_reason': 'stop',
'logprobs': None})], [Generation(text='\n\nFootwear Factory.', generation_info={'finish_reason': 'stop',
```

```
'logprobs': None})]], llm_output={'token_usage': {'prompt_tokens': 36, 'total_tokens': 55,
'completion_tokens': 19}, 'model_name': 'text-davinci-003'})
```

- `predict` is similar to `run` method except that the input keys are specified as keyword arguments instead of a Python dict.

```
# Single input example
llm_chain.predict(product="colorful socks")
```

```
    '\n\nSocktastic!'
```

```
# Multiple inputs example

template = """Tell me a {adjective} joke about {subject}."""
prompt = PromptTemplate(template=template, input_variables=["adjective", "subject"])
llm_chain = LLMChain(prompt=prompt, llm=OpenAI(temperature=0))

llm_chain.predict(adjective="sad", subject="ducks")
```

```
    '\n\nQ: What did the duck say when his friend died?\nA: Quack, quack, goodbye.'
```

Parsing the outputs

By default, `LLMChain` does not parse the output even if the underlying `prompt` object has an output parser. If you would like to apply that output parser on the LLM output, use `predict_and_parse` instead of `predict` and `apply_and_parse` instead of `apply`.

With `predict`:

```
from langchain.output_parsers import CommaSeparatedListOutputParser

output_parser = CommaSeparatedListOutputParser()
template = """List all the colors in a rainbow"""
prompt = PromptTemplate(template=template, input_variables=[], output_parser=output_parser)
llm_chain = LLMChain(prompt=prompt, llm=llm)

llm_chain.predict()
```

```
    '\n\nRed, orange, yellow, green, blue, indigo, violet'
```

With `predict_and_parser`:

```
llm_chain.predict_and_parse()
```

```
    ['Red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
```

Initialize from string

You can also construct an LLMChain from a string template directly.

```
template = """Tell me a {adjective} joke about {subject}."""
llm_chain = LLMChain.from_string(llm=llm, template=template)
```

```
llm_chain.predict(adjective="sad", subject="ducks")
```

```
    '\n\nQ: What did the duck say when his friend died?\nA: Quack, quack, goodbye.'
```