Output Parsers

Language models output text. But many times you may want to get more structured information than just text back. This is where output parsers come in.

Output parsers are classes that help structure language model responses. There are two main methods an output parser must implement:

- `get_format_instructions() -> str`: A method which returns a string containing instructions for how the output of a language model should be formatted.
- `parse(str) -> Any`: A method which takes in a string (assumed to be the response from a language model) and parses it into some structure.

And then one optional one:

- `parse_with_prompt(str) -> Any`: A method which takes in a string (assumed to be the response from a language model) and a prompt (assumed to the prompt that generated such a response) and parses it into some structure. The prompt is largely provided in the event the OutputParser wants to retry or fix the output in some way, and needs information from the prompt to do so.

Below we go over the main type of output parser, the `PydanticOutputParser`. See the `examples` folder for other options.

```
from langchain.prompts import PromptTemplate, ChatPromptTemplate,
HumanMessagePromptTemplate
from langchain.llms import OpenAI
from langchain.chat_models import ChatOpenAI

from langchain.output_parsers import PydanticOutputParser
from pydantic import BaseModel, Field, validator
from typing import List
```

```
model_name = 'text-davinci-003'
temperature = 0.0
model = OpenAI(model_name=model_name, temperature=temperature)
```

Skip to main content

```
# Define your desired data structure.
class Joke(BaseModel):
    setup: str = Field(description="question to set up a joke")
    punchline: str = Field(description="answer to resolve the joke")

    # You can add custom validation logic easily with Pydantic.
    @validator('setup')
    def question_ends_with_question_mark(cls, field):
        if field[-1] != '?':
            raise ValueError("Badly formed question!")
        return field
```

```
# Set up a parser + inject instructions into the prompt template.
parser = PydanticOutputParser(pydantic_object=Joke)
```

```
prompt = PromptTemplate(
    template="Answer the user query.\n{format_instructions}\n{query}\n",
    input_variables=["query"],
    partial_variables={"format_instructions": parser.get_format_instructions()}
)
```

```
# And a query intented to prompt a language model to populate the data structure.
joke_query = "Tell me a joke."
_input = prompt.format_prompt(query=joke_query)
```

```
output = model(_input.to_string())
```

```
parser.parse(output)
```

```
Joke(setup='Why did the chicken cross the road?', punchline='To get to the other
side!')
```