```
print(input_text)
print(f"\nCorrect response:\n>> {entry['output']}")
print(f"\nModel response:\n>> {response_text.strip()}")
print("---------------------------------------")
```

As mentioned earlier, the `generate` function returns the combined input and output text, so we use slicing and the `.replace()` method on the `generated_text` contents to extract the model's response. The instructions, followed by the given test set response and model response, are shown next.

Below is an instruction that describes a task. Write a response that appropriately completes the request.

Instruction:

Rewrite the sentence using a simile.

Input:

The car is very fast.

Correct response:

>> The car is as fast as lightning.

Model response:

>> The car is as fast as a bullet.

Below is an instruction that describes a task. Write a response that appropriately completes the request.

Instruction:

What type of cloud is typically associated with thunderstorms?

Correct response:

>> The type of cloud typically associated with thunderstorms is cumulonimbus.

Model response:

>> The type of cloud associated with thunderstorms is a cumulus cloud.

Below is an instruction that describes a task. Write a response that appropriately completes the request.

Licensed to Jason Walsh <manning@jwalsh.net>

Instruction:

Name the author of 'Pride and Prejudice.'

Correct response:

>> Jane Austen.

Model response:

>> The author of 'Pride and Prejudice' is Jane Austen.

As we can see based on the test set instructions, given responses, and the model's responses, the model performs relatively well. The answers to the first and last instructions are clearly correct, while the second answer is close but not entirely accurate. The model answers with "cumulus cloud" instead of "cumulonimbus," although it's worth noting that cumulus clouds can develop into cumulonimbus clouds, which are capable of producing thunderstorms.

Most importantly, model evaluation is not as straightforward as it is for classification fine-tuning, where we simply calculate the percentage of correct spam/non-spam class labels to obtain the classification's accuracy. In practice, instruction-fine-tuned LLMs such as chatbots are evaluated via multiple approaches:

- Short-answer and multiple-choice benchmarks, such as Measuring Massive Multitask Language Understanding (MMLU; https://arxiv.org/abs/2009.03300), which test the general knowledge of a model.
- Human preference comparison to other LLMs, such as LMSYS chatbot arena (https://arena.lmsys.org).
- Automated conversational benchmarks, where another LLM like GPT-4 is used to evaluate the responses, such as AlpacaEval (https://tatsu-lab.github.io/alpaca_eval/).

In practice, it can be useful to consider all three types of evaluation methods: multiple-choice question answering, human evaluation, and automated metrics that measure conversational performance. However, since we are primarily interested in assessing conversational performance rather than just the ability to answer multiple-choice questions, human evaluation and automated metrics may be more relevant.

Conversational performance

Conversational performance of LLMs refers to their ability to engage in human-like communication by understanding context, nuance, and intent. It encompasses skills such as providing relevant and coherent responses, maintaining consistency, and adapting to different topics and styles of interaction.

Licensed to Jason Walsh <manning@jwalsh.net>

Human evaluation, while providing valuable insights, can be relatively laborious and time-consuming, especially when dealing with a large number of responses. For instance, reading and assigning ratings to all 1,100 responses would require a significant amount of effort.

So, considering the scale of the task at hand, we will implement an approach similar to automated conversational benchmarks, which involves evaluating the responses automatically using another LLM. This method will allow us to efficiently assess the quality of the generated responses without the need for extensive human involvement, thereby saving time and resources while still obtaining meaningful performance indicators.

Let's employ an approach inspired by AlpacaEval, using another LLM to evaluate our fine-tuned model's responses. However, instead of relying on a publicly available benchmark dataset, we use our own custom test set. This customization allows for a more targeted and relevant assessment of the model's performance within the context of our intended use cases, represented in our instruction dataset.

To prepare the responses for this evaluation process, we append the generated model responses to the `test_set` dictionary and save the updated data as an `"instruction-data-with-response.json"` file for record keeping. Additionally, by saving this file, we can easily load and analyze the responses in separate Python sessions later on if needed.

The following code listing uses the `generate` method in the same manner as before; however, we now iterate over the entire `test_set`. Also, instead of printing the model responses, we add them to the `test_set` dictionary.

Listing 7.9 Generating test set responses

```
from tqdm import tqdm

for i, entry in tqdm(enumerate(test_data), total=len(test_data)):
    input_text = format_input(entry)

    token_ids = generate(
        model=model,
        idx=text_to_token_ids(input_text, tokenizer).to(device),
        max_new_tokens=256,
        context_size=BASE_CONFIG["context_length"],
        eos_id=50256
    )
    generated_text = token_ids_to_text(token_ids, tokenizer)

    response_text = (
        generated_text[len(input_text):]
        .replace("### Response:", "")
        .strip()
    )
    test_data[i]["model_response"] = response_text

with open("instruction-data-with-response.json", "w") as file:
    json.dump(test_data, file, indent=4)
```

indent for pretty-printing

Licensed to Jason Walsh <manning@jwalsh.net>