Learning Objectives

Learners will be able to...

- **Generate a Response Using `openai.Completion.create`**
- **Define Large Language Models**
- **Learn about Classification Design**
- **Practice Creating Prompts**
- **Run Different Prompts with OpenAI**

info

Make Sure You Know

You are familiar with Python.

Limitations

This is a gentle introduction. So there is a little bit of Python programming. The information might not be the most up to date as OpenAI releases new features.

Large Language Models

Refining the Results

Up until now, it should be clear just how powerful large language models like GPT-3 are. **Large language models (LLMs)** are machine learning algorithms that can recognize, summarize, translate, predict, and generate human languages on the basis of very large text-based datasets.

In short, you can apply models like GPT-3 virtually any task that involves understanding or generating natural language or code. Add the following prompt to the file on the left. Then click the `TRY IT` button. The model should return a working JavaScipt function.

```
write a recursive function in javascript that calculates the
        first n numbers of the fibonacci sequence
```

Given the vast capabilities of large language models, we should stop asking what they can do. Instead, a better question would be, “How do we get better results?” Here are three basic guidelines to creating better results:

- **Show and tell**
- **Provide quality data**
- **Check your settings**

We will explore these guidelines over the course of this assignment.

Show and Tell

Constructing a Better Prompt

One of the more impressive aspects of the GPT-3 model is that it can give quality answers to vague prompts. If we want to increase the quality of the responses, we need to improve the prompt. Use **show and tell** as a guideline when developing a prompt.

For show and tell we want to focus on 3 things:

1. Giving the model clear instructions (tell)
2. Giving the model an example (show)
3. Giving the model clear instructions and an example (show and tell)

Let’s start with a simple prompt that does not make use of the show and tell principles.

```
recommend 10 movies
```

If you want to try other prompts. You can use the `RESET` button below to clear the text in the file on the left. Then use the `TRY IT` button to generate another response.

Most likely, you see movies that are popular, critically acclaimed, or maybe you have never heard of the film before. Since you are asking for ten titles, the model numbers each film. If you run the prompt a couple of times you may see the numbering like `1.` or even `1)`. Since we are not being specific, the model is not very consistent.

We can produce a better list of films by telling the model what features you are looking for. Instead of numbers, we want a list that uses numbers to identify each item.

```
recommend 10 movies. use letters when listing your movies.
```

The model now produces a list with letters instead of numbers. But the list can be just as inconsistent in terms of formatting. This is where the show principle comes into play. In addition to the prompt, show the model how you want the list formatted. Each line in the list should contain a letter followed by a closing parenthesis, and the film title should appear between quotes.

```
recommend 10 movies. use letters when listing your movies.
A) "Titanic"
B) "The Godfather"
```

For more precision, we can combine the show and tell principles. Tell the prompt that you want 10 horror movies in a list that uses letters. Then show the model how you want the list formatted.

```
recommend 10 horror movies. use letters when listing your
        movies.
A) "The Ring"
B) "The Exorcist"
```

Using the show and tell principles, you can dramatically change the type and format of the model's response. With the right combination, you can improve the quality of the results, which is more desirable than the standard GPT-3 response.

Provide Quality Data

Intentionality

You may be familiar with the expression, “Garbage in, garbage out.” Believe it or not, this holds true for models like GPT-3. The model is sophisticated enough to produce coherent output even with terrible prompts, so GPT-3 rarely provides “garbage out”. However, the idea is to produce the best results possible, so we need to give the model the best data we can. We are going to ask the model to perform sentiment analysis, which determines how positive or negative a sentence is.

```
what is the overall sentiment of the following sentences:
["i am happy","i am happy to be sad","I am sad"]
```

The model will say something along the lines of that the overall sentiment is “happy”. Is that true? The third sentence clearly does not portray a happy sentiment. What happened? We gave GPT-3 three distinct sentences, but we did not specify how the sentiment analysis should be performed. Do we want an average score for all three sentences, or do we want scores for each individual sentence? We never told the model what to do, so it calculated the average sentiment.

If we want sentiment analysis for each sentence in the list, then we need to explicitly tell that to the model. Let’s modify the original prompt above by adding the words “for each” to it.

```
what is the overall sentiment for each of the following
        sentences:
["i am happy","i am happy to be sad","I am sad"]
```

We can see by giving it better directions, it was more clear on the assignment it was given. Feel free to reset and try different ways you can get the AI to generate and present new information.

challenge

Try this variation:

Modify the prompt above so that it performs sentiment analysis on each sentence. The results should display each sentence followed by its sentiment analysis.

▼ **Solution**

Here is one possible solution:

```
what is the overall sentiment for each of the following
        sentences:
["i am happy","i am happy to be sad","I am sad"]


sentence: i am happy
sentiment:
```

Finally, be sure to proofread your examples. The model is usually smart enough to see through basic spelling mistakes and give you a response. However, it also might assume this is intentional (think of companies like eBay, Tumblr, and Reddit), which can affect the response. Be intentional about the instructions you give to the model.

Check Your Settings

Keyword Arguments

As important as the prompt is, it is only one component of generating a response from the GPT-3 model. Let's leave the OpenAI playground and return to the Python language. We pass the `openai.Completion.create` method a variety of keyword arguments. The `prompt` is but one of these keyword arguments. By focusing on the interplay between these keyword arguments, we can improve the quality of the responses.

```
response = openai.Completion.create(
  model="text-davinci-002",
  prompt="",
  temperature=0,
  max_tokens=25,
  top_p=0,
  frequency_penalty=0,
  presence_penalty=0
)
```

In particular, keep a close eye on `temperature` and `top_p`. These keyword arguments control how deterministic the model is when generating a response. That is, adjusting the values of these keyword arguments can create a result that does not change much (if at all) each time the model runs.

Let's set the temperature to `0.1` and try it with the prompt `when did dadism start`. This value increases the model's confidence in its top choice.
The closer your temperature is to 0, the more deterministic the model will become. This means you may see very little variance or "creativity" in the response. Instead, you should see a rather matter-of-fact statement.

```
prompts ="when did dadism start?"

response = openai.Completion.create(
  model="text-davinci-002",
  prompt=prompts,
  temperature=0.1)

print(response['choices'][0]['text'].strip())
```

challenge

Try this variation:

- Try running the code multiple times when you set `temperature` to `1`. You should now see more variance or “creativity” in the responses.

```
response = openai.Completion.create(
  model="text-davinci-002",
  prompt=prompts,
  temperature=1)
```

Now let’s take a look at `top_p`. This keyword argument sets the scope of the potential results. The larger the value, the greater number of potential responses the model will consider the best result. Set the value of `top_p` to `1` and run the code a few times. You should see a variety in responses each time the code runs.

```
response = openai.Completion.create(
  model="text-davinci-002",
  prompt=prompts,
  top_p=1)
```

challenge

Try this variation:

- Try running the code multiple times when you set `top_p` to `0.1`. In this example, you are limiting the responses to the 10% of the possible answers. This means the potential responses are limited in nature and therefore more deterministic.

```
response = openai.Completion.create(
  model="text-davinci-002",
  prompt=prompts,
  top_p=0.1)
```

Combining `top_p` and `temperature`

Both the `top_p` and `temperature` are correlated, which means using one affects the results of the other. Be careful in how you use these two keyword arguments. Take a look at the example below. The `temperature` keyword argument should maximize creativity. However, `top_p` limits the results to only the top 10%. That means the `top_p` value counteracts the `temperature` value, and the results will be more deterministic rather than creative.

```
response = openai.Completion.create(
  model="text-davinci-002",
  prompt=prompts,
  temperature=1,
  top_p=0.1)
```

Keep in mind how these two keyword arguments work together. Starting out, it might be best to use either `top_p` or `temperature` to control variance in responses. This way you will not “undo” one keyword argument with the other.

top_p is a hyperparameter that controls the cumulative probability cutoff for the set of next possible words the model can choose during generation. If top_p is set to 0.95, for example, the model will consider the smallest set of next possible words whose combined probability exceeds 0.95.

Troubleshooting Tips

If you’re having trouble getting the API to perform as expected, follow this checklist:

- Is it clear what the intended generation should be?
- Are there enough examples?
- Did you check your examples for mistakes? (The API won’t tell you directly)
- Are you using `temperature` and `top_p` correctly?
- Are your other settings being used correctly?

The last tip may seem a bit vague. To illustrate this point, change the prompt so that it is asking for the sentiment of each sentence.

```
prompts ="what is the overall sentiment for each of the
        following sentences:['i am happy','i am happy to be
        sad','I am sad']"
```

Run the script once more. You should notice that the response is incomplete. The response ends before describing the sentiment of each sentence. That is because we did not specify a value for the `max_tokens` keyword argument. The model uses the default value 16, which is insufficient for the response.

To remedy this problem, set max_tokens to 100. Run the program again.

```
response = openai.Completion.create(
  model="text-davinci-002",
  prompt=prompts,
  top_p=1,
  max_tokens=100)
```

You should see a complete response that lists the sentiment for each sentence. Generating a good response is a balancing act between all of the different factors that affect how the model works.

Classification Design

A classifier is any algorithm that sorts data into different classes. GPT-3 is a model which makes use of several different algorithms, including those used for classification, to generate its results. This means that GPT-3 can perform classification out of the box. However, you need to structure your prompts for successful results.

We use plain language to describe your inputs and outputs. In the code below, We use plain language for the input “tweet” and the expected output “sentiment.” As a best practice, start with plain language descriptions.

```
Classify the sentiment of these tweets:

1. "I had the worst day"
2. "I had a blast at the movies"
3. "I can't wait for christmas"
4. "My cat is adorable ❤❤"
5. "I hate chocolate ☹"
6. "My day was okay"
```

While you can often use shorthand or keys to indicate the input and output, it’s best to start by being as descriptive as possible. Then work backwards to remove extra words and see if performance stays consistent.

challenge

Try this variation:

- Reduce the clarity of the prompt by having only a single word. Compare these results from the prompt above.

```
sentiment

1. "I had the worst day"
2. "I had a blast at the movies"
3. "I can't wait for christmas"
4. "My cat is adorable ❤❤"
5. "I hate chocolate ☹"
6. "My day was okay"
```

Show the API how to respond to any case. In this example, we remove the prompt in plain language. Instead we provide examples for how the model should respond to the prompt. The model should be able to infer how to respond to each tweet in the prompt. Click the `TRY IT` button a few times and notice how the model responds to tweet #6.

```
1. "I had the worst day"
2. "I had a blast at the movies"
3. "I can't wait for christmas"
4. "My cat is adorable ❤❤"
5. "I hate chocolate ☹"
6. "My day was okay"

tweet 1: -
tweet 2: +
```

Because we did not specify how to respond to tweets with the neutral sentiment, the model will respond with `0` or `+/-` or sometimes a `+`. If we want to have a specific label for a neutral tweet, then we need to provide an example for how to respond. A neutral label is important because there will be many cases where even a human would have a hard time determining if something is positive or negative.

Add a key with the expected label for each sentiment. Click the `TRY IT` button a few times and compare the output from the example above.

```
1. "I had the worst day"
2. "I had a blast at the movies"
3. "I can't wait for christmas"
4. "My cat is adorable ❤❤"
5. "I hate chocolate ☹"
6. "My day was okay"

if positive: +
if negative: -
if neutral: ?

tweet 1: -
tweet 2: +
```

Important, the model already understands the concepts of sentiment and a tweet. You need fewer examples for familiar tasks such as this. If you're building a classifier for something the model might not be familiar with, it might be necessary to provide more examples.

Coding Exercise

Write a function called `generate_response` that takes a prompt and model as input and returns the generated response using the `openai.Completion.create method`. The function should have the following signature:

```
def generate_response(prompt, model):
  #your code should be below this line
    return response.choices[0].text.strip()
```