Homework 7: Evaluating Generative AI Models

Due November 16th at 10pm

In this homework, you will explore evaluating large language models on two tasks: 1) solving plain-text Murdle puzzles and 2) generating new Murdle puzzles.

Submit your Python files along with your question responses on Gradescope. **Please do not rename your Python files.**

Sending Model Requests

For this assignment, you will work with a large language model called GPT-5-nano. You will query the model using the OpenAI Python library (openai). First, install this library using pip:

pip install openai

Next, in the terminal, configure your API key before running your program.

For Mac/Linux:

export OPENAI_API_KEY=‘‘your−api−key’’

For Windows:

setx OPENAI_API_KEY ‘‘your−api−key’’

IMPORTANT: Your API is a secret key! Do not upload it anywhere online or share it with anyone! It will cost actual money if your key leaks and someone else uses it!

I have given you each a limited but sufficient amount of money per API key for this assignment. However, **please don't be wasteful!** If you know a run of your program is pointless, please do kill it. You can kill a running program using CTRL + C.

Hint: to check if your API key is correctly configured, you can start a Python shell session and run the following:

```
import os
print(bool(os.environ.get("OPENAI_API_KEY"))) # should be True

import query_openai
print(query_openai.send_query('Say hello.'))
```

1 Task 1: Solving Plain-Text Murdles (30 pts)

In this task, you will explore whether GPT-5-nano can solve plain-text Murdle puzzles. I have given you a dataset of 30 puzzles in **murdle_text**. The suspects have short IDs such as `S1` and `S2`. Your program will ask the model to identify the murderer using one of these IDs.

1.1 Reading the data

I have given you a function called **read_murdle_data**. It reads in all of the Murdle puzzles in the folder, plus the answers in **answer_key.csv**.

It return two aligned lists: 1) a list of puzzles represented as dictionaries with `date` and `text` fields, and 2) a dictionary of date-murderer pairs.

1.2 Crafting a basic prompt

Next, write a function called **make_prompt**. This function should take in a puzzle's text, prepare a prompt, and return the full prompt. Your task is to craft prompts to feed to the model so that it solves the Murdle task.

First, try just a basic description of the task. I have included a FORMAT string that should be added to the end of your instructions, to tell the model how to format its response. Your prompt should end with the text of the puzzle to solve.

This function should take a puzzle and a condition ("basic" or "improved") and return your prompt.

1.3 Running one prompt

Next, **write a function** called **run_one_prompt**. This function will take a puzzle, a label, and a condition. It should call **make_prompt** to construct a prompt for the puzzle, and then use **query_model** to send it to the model. For now, just print out the model's response.

1.4 Extracting and scoring answers

When you run a prompt through the LLM, you will need to do some additional string processing to get back an answer. The model may include an explanation before its final answer.

Write a function called **get_answer**. It should take the response text and search for a suspect ID within it.

Your function should search for a suspect id within the final nonblank line in the format `ANSWER: S<number>`. Suspect ids start with capital S and end in a number. If the answer is missing, return `None`; this prediction will receive a score of zero.

Your function should handle variations in text formatting like capitalization and punctuation.

Now that you've written this scoring function, return your **run_one_prompt** function. **Update it to call get_answer and compare the result with the correct label.**

Return the result as a dictionary and 1 or 0 indicating if the answer was correct.

1.5 Evaluating the dataset

Now add lines of code to your **main** function so that it loops over the dataset of puzzles and evaluates each of them using **run_one_prompt**. Compute and print the model's accuracy on the full dataset. You should also call **export_results** to save the results of each condition.

1.6 Improving your prompt

Next, write an Improved prompt. This will be used in the "improved" condition.

You can experiment with different ways of describing the task and formatting your prompts. You might add more details about the task, or hints about how to approach solving the puzzle. You can also use the miniature puzzle example that I've given you at the top of the program. Just make sure you don't include the target puzzle's solution in your prompt.

Extend your **make_prompt** function so that it runs the "improved" condition with your new prompt.

You should try out different prompts and submit your assignment with your best version stored in IMPROVED at the topic of your file.

1.7 Analysis questions

Answer the following questions in your write-up:

1. **How well did the model perform on the task?** Report accuracy for each of the three prompting approaches, and describe some of the errors the model made.
2. **Describe the different strategies you tried out for prompting.** What worked well? What didn't work?

2 Task 2: Murdle Generation (30 points)

In this task, you will explore how well models are able to generate plain-text Murdle puzzles.

2.1 Writing your benchmarking program

You can structure your benchmarking program similar to the last one.

I have given you a JSON schema, which can be used to explain the puzzle format to the model and test that the generated puzzles are structurally correct.

I have provided functions for loading the schema and an example Murdle puzzle; for testing if the puzzle is valid; and for exporting the generated puzzles.

You will need to finish writing **make_prompt** and **run_one_puzzle**.

You should submit your program with two prompt versions:

1. Plain prompt: minimal description of the task and the required format.
2. Special prompt: your best prompt, which can use the example puzzle I've given you, or any other instructions you think might improve the quality of the generated puzzles.

2.2 Generating some puzzles

Once your generation program is working, **use it to generate 10 puzzles using each version of your prompt.**

2.3 Analysis questions

Answer the following questions in your write-up:

1. **How well did the model perform on the task?** Report how many times the model failed in each condition, and describe some of the errors the model made.
2. **Describe the different strategies you tried out for prompting.** What worked well? What didn't work?

3 Task 3: Human Evaluation (20 points)

A Murdle puzzle not only needs to be parseable, but fun. Judging this task relies on human taste: what makes a Murdle puzzle clever, funny, or satisfying?

In this section, you will do a small-scale human analysis of your generated puzzles. You will collect ratings for **at least 7 puzzles in each condition.**

I have written you a program called **annotate.py** that runs a web application to make it easier to collect ratings.

The web application uses the **inventory.csv** file in **generated-murdles**. (If you want to exclude any puzzles from the rating process, remove them from the inventory.)

You can run the web application as follows:

```
python annotate.py
```

Once it is running, you can open it up in a web browser at **http://127.0.0.1:8000**.

This web page will allow you to view a puzzle and give it a rating. It stores the ratings as a CSV file called **ratings.csv**.

3.1 Obtaining Human Judgments

Go through and rate each of your puzzles. Now, you are not an entirely fair annotator; you may know which puzzles were Special and which were Plain.

To get fairer scores, recruit a friend to rate the puzzles. Don't tell your friend which puzzles were created by which prompt.

3.2 Analysis questions

Answer the following questions in your write-up:

1. **How often do you and your friend agree?** You can simply report your average disagreement (for each item, take the absolute value of the difference of your ratings), or, if you'd like to be more thorough, you can compute Weighted Cohen's Kappa.
2. **Do your Special puzzles receive higher ratings on average?**

4 Task 4: Reading Reflections

These questions ask you to reflect on pages 170-215 of *AI Needs You* ("Trust and Terror").

Question 1 (10 points)

Harding writes: *Members of the national security apparatus began to highlight that Google or Facebook knew just as much, or more, about you as the government, and that you should fear them more than the state. The companies countered that an individual could choose to opt out of using Facebook or Google but could not opt out of government surveillance.*

What do you think about the argument that you can opt out of using all major tech companies? What effects would this have on your opportunities and lifestyle?

Question 2 (10 points)

Harding frames the Snowden leaks as a key turning point because the US and UK "unintentionally ceded the higher moral ground." **Compare and contrast this with the approach taken by the US in the Space Race as discussed in the earlier chapter "Peace and War."**