12.12.23

OPENCAMPUS.sh

Practical Engineering with LLMs

Evaluation of LLM outputs and structured output

Quiz

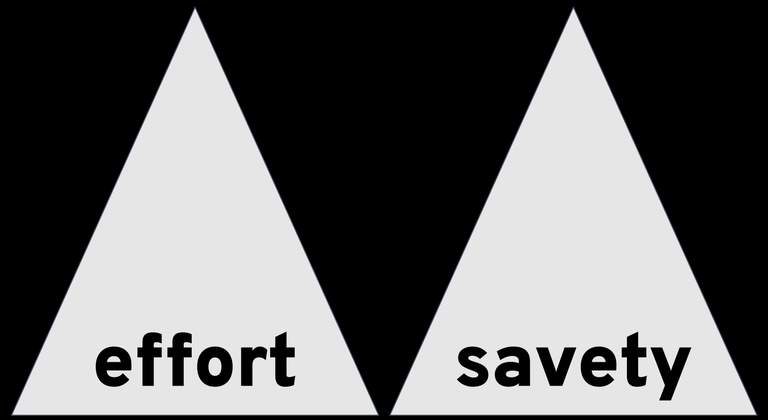
https://forms.office.com/r/jnDyZvjEKu

TODAY’S SCHEDULE

- Quiz
- Short Recap
- Short presentation of current status
- Breakout Session ‘Next Steps’
- Homework for next week

Steps to evaluate LLM outputs

- Tune prompts on handful of examples
- Add additional "tricky" examples opportunistically
- Develop metrics to measure performance on examples
- Collect randomly sampled set of examples to tune to (development set/hold-out cross validation set)
- Collect and use a hold-out test set



Evaluation methods - Rubric

Write a rubric to let a LLM evaluate answers based on context!

```
You are evaluating a submitted answer to a question based on the context \
that the agent uses to answer the question.
Here is the data:
[BEGIN DATA]
************
[Question]: {cust_msg}
************
[Context]: {context}
************
[Submission]: {completion}
************
[END DATA]
```

Evaluation methods - Rubric

```
Compare the factual content of the submitted answer with the context. \
Ignore any differences in style, grammar, or punctuation.
Answer the following questions:
    - Is the Assistant response based only on the context provided? (Y or N)
    - Does the answer include information that is not provided in the context? (Y or N)
    - Is there any disagreement between the response and the context? (Y or N)
    - Count how many questions the user asked. (output a number)
    - For each question that the user asked, is there a corresponding answer to it?
      Question 1: (Y or N)
      Question 2: (Y or N)
      ...
      Question N: (Y or N)
    - Of the number of questions asked, how many of these questions were addressed by the answer
```

Evaluation methods - Rubric

```
Compare the factual content of the submitted answer with the context. \
Ignore any differences in style, grammar, or punctuation.
Answer the following questions:
    - Is the Assistant response based only on the context provided? (Y or N)
    - Does the answer include information that is not provided in the context? (Y or N)
    - Is there any disagreement between the response and the context? (Y or N)
    - Count how many questions the user asked. (output a number)
    - For each question that the user asked, is there a corresponding answer to it?
      Question 1: (Y or N)
      Question 2: (Y or N)
      ...
      Question N: (Y or N)
    - Of the number of questions asked, how many of these questions were addressed by the answer?
```

Evaluation methods - Compare with ideal/human answers

- Create a test set with ideal / human-generated answers
- Let the LLM judge whether the generated answer is semantically consistent with the provided ideal answer
- Provide the LLM options to choose from (subset, superset, semantically equal, etc.)

Evaluation methods - Compare with ideal/human answers

```
You are comparing a submitted answer to an expert answer on a given question. Here is the data:
[BEGIN DATA]
************
[Question]: {cust_msg}
************
[Expert]: {ideal}
************
[Submission]: {completion}
************
[END DATA]
```

```
Compare the factual content of the submitted answer with the expert answer. Ignore any differences in style, grammar, or pu
    The submitted answer may either be a subset or superset of the expert answer, or it may conflict with it. Determine whi
    (A) The submitted answer is a subset of the expert answer and is fully consistent with it.
    (B) The submitted answer is a superset of the expert answer and is fully consistent with it.
    (C) The submitted answer contains all the same details as the expert answer.
    (D) There is a disagreement between the submitted answer and the expert answer.
    (E) The answers differ, but these differences don't matter from the perspective of factuality.
  choice_strings: ABCDE
```

Evaluation methods - LangChain

LangChain: Evaluation

Outline:

- Example generation
- Manual evaluation (and debuging)
- LLM-assisted evaluation
- LangChain evaluation platform

Evaluation methods - QA Evaluation chains

Use QAGenerateChain to generate RAG evaluation examples from documents.

LLM-Generated examples

In [11]:
```
from langchain.evaluation.qa import QAGenerateChain
```

In [12]:
```
example_gen_chain = QAGenerateChain.from_llm(ChatOpenAI(model=llm_m
```

In [13]:
```
# the warning below can be safely ignored
```

In [14]:
```
new_examples = example_gen_chain.apply_and_parse(
    [{"doc": t} for t in data[:5]]
)
```

In [15]:
```
new_examples[0]
```

```
{'query': "What is the weight of the Women's Campside Oxfords per p
air?",
 'answer': "The Women's Campside Oxfords weigh approximately 1 lb.
1 oz. per pair."}
```

Evaluation methods - Evaluation chain

Use QAEvalChain to evaluate the output of a QA chain with the given examples

```
In [23]: from langchain.evaluation.qa import QAEvalChain

In [24]: llm = ChatOpenAI(temperature=0, model=llm_model)
         eval_chain = QAEvalChain.from_llm(llm)

In [25]: graded_outputs = eval_chain.evaluate(examples, predictions)

In [26]: for i, eg in enumerate(examples):
             print(f"Example {i}:")
             print("Question: " + predictions[i]['query'])
             print("Real Answer: " + predictions[i]['answer'])
             print("Predicted Answer: " + predictions[i]['result'])
             print("Predicted Grade: " + graded_outputs[i]['text'])
             print()
```

```
Example 0:
Question: Do the Cozy Comfort Pullover Set        have side pocket
s?
Real Answer: Yes
Predicted Answer: The Cozy Comfort Pullover Set, Stripe has side po
ckets on the pull-on pants.
Predicted Grade: CORRECT
```

Output Parsing - StructuredOutputParser

1. Define reponse schemas
2. Generate a StructuredOutputParser
3. Generate format instructions based on the response schemas
4. Format a prompt with the generated format instructions added
5. Generate a response
6. Let the output parser parse the output based on the given format instructions

Works good for JSON output with string values

Output Parsing - StructuredOutputParser

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

llm = ChatOpenAI(temperature=0.0)

prompt = ChatPromptTemplate(
    messages=[
        HumanMessagePromptTemplate.from_template(template_string)
    ],
    input_variables=["brand_description"],
    partial_variables={"format_instructions": format_instructions},
    output_parser=output_parser # here we add the output parser to the Prompt template
)
```

Output Parsing - CommaSeperatedListOutputParser

- Similar to the StructuredOutputParser
- Works good for comma separated lists instead of JSON

Output Parsing - PydanticOutputParser

- Allows to define the structure of the output in the Pydantic format (defined as class)
- Different data types can be used
- Output can be validated with the defined output format

```python
class BrandInfo(BaseModel):
    brand_name: str = Field(description="This is the name of the brand")
    reasoning: str = Field(description="This is the reasons for the score")
    likelihood_of_success: int = Field(description="This is an integer score between 1-10")

    # You can add custom validation logic easily with Pydantic.
    @validator('likelihood_of_success')
    def check_score(cls, field):
        if field >10:
            raise ValueError("Badly formed Score")
        return field
```

Output Parsing - OutputFixingParser

- Parsers can be used to create an OutputFixingParser that will try to correct the incorrect output

```
from langchain.output_parsers import OutputFixingParser

new_parser = OutputFixingParser.from_llm(parser=parser,
                                         llm=ChatOpenAI())
```

Output Parsing - RetryWithErrorOutputParser

- Parsers can be used to create an RetryWithErrorOutputParser that will retry the parsing
- Also takes the original prompt into account

```
1  from langchain.output_parsers import RetryWithErrorOutputParser
```

```
1  retry_parser = RetryWithErrorOutputParser.from_llm(parser=parser, llm=OpenAI(temperature=0))
```

```
1  retry_parser.parse_with_prompt(bad_response, prompt_value)
```

```
Action(action='search', action_input='who is leo di caprios gf?')
```

Evaluation methods - Evaluation frameworks

OpenAI Eval framework

https://github.com/openai/evals
https://medium.com/@sergioli/evaluating-chatgpt-using-openai-evals-7ca85c0ad139

RAGAS

https://github.com/explodinggradients/ragas

LangSmith

https://www.langchain.com/langsmith

Short Project Presentation

Quickly explain to the audience in a few sentences what your current project status is, what your next steps are and if you feel that you need any help or run into problems.

Breakout Rooms

Where do you want to go?

- Remind yourselves what the goal is, e.g. have a look at your project template! Are all participants on the same page? All hands on board?

Where are you now?

- What have you tried? What did not work (yet)? What worked? What have you already accomplished?

How do you get to the finish line?

- What steps are necessary to finish your project until week 9? Who will be working on which part? How do you put it all together? What is realistic?

Homework for Next Week

- Take a look at this article: https://blog.n8n.io/open-source-llm/
- Take a look at some open-source / open-access LLM frameworks:
  - Ollama (Mac, Linux):
    - https://www.youtube.com/watch?v=Ox8hhpgrUi0
    - https://www.youtube.com/watch?v=k_1pOF1mj8k
  - LM Studio (Windows, Mac, Linux):
    https://medium.com/@genebernardin/running-llms-locally-using-lm-studio-38070f286413
  - GPT4All
- Test some open-source/open-access LLMs either locally downloaded or in HuggingFace spaces or any other test environment (e.g. HuggingChat, H2O.ai, etc.)
  - Do you see significant differences in the output of proprietary and open-source/open-access LLMs?