Lab 3: LLMs, Prompting and RAG

June 13, 2024

Welcome to Lab 3 of our course on Natural Language Processing. Today, we will be diving deep into the fourth and most recent paradigm in NLP teased in the previous Lab, i.e. Pre-train, Prompt and Predict. The core idea behind the paradigm is that once we train a big enough language model (pre-training + instruction tuning), we do not really need to train these models further to solve any specific taks, but instead can directly prompt the model to solve a task by specifying instructions, task descriptions and in some cases a few examples.

Like last time we will be working on the with the SocialIQA dataset, and demonstrating how to work with LLMs to solve such tasks.

Along with the SocialIQA dataset, we will also delve into the fascinating world of Retrieval Augmented Generation (RAG). RAG is a powerful technique that combines the strengths of pre-trained language models and information retrieval systems to generate contextually relevant responses.

For building the RAG system, why not build something which might be useful for plaksha students. We will build a Question Answering system which will be able to answer questions about the Plaksha Professors. For this task, the dataset is already generated by scraping the plaksha full-time faculty page to get information about various professors, their areas of expertise, research interests, and more. Our goal is to convert this data into embeddings.

Once we have these embeddings, we can use them to retrieve contextually relevant information based on a given query. For instance, if a student wants to know which professor specializes in Natural Language Processing, our RAG system should be able to retrieve the relevant professor details.

The final part of our system is a Language Model (LM). Once we have the relevant context from our retrieval system, we pass it to a pre-trained LM. The LM then generates a coherent and contextually appropriate response.

By the end of this lab, you will have a hands-on understanding of how to build a RAG system. You will learn how to convert text data into embeddings, how to retrieve relevant context based on a query, and how to generate responses using a pre-trained LM.

This Lab doesn't require any GPU, since we will be heavily using APIs from various third party sources.

For the embeddings we will be utilizing Voyage AI's latest Embedding model via their API. The api provdies free access to embeddings upto 50 Million tokens, which are

plenty for our assignments and even for your final projects if needed.
Note: The Voyage API has very low rate limits when you don't add payment details, with 3 RPM(requests per minute)

For Language Models, we have two options, the first one being groq, which has various models like LLaMa 3 8b/70b, Mixtral 8x7b and Gemma 7b which are free to use and have very high throughput.
Another option is to use Open Router, where there are plenty of free model options available.
Both service providers are compatible with OpenAI package, and hence can be used interchangebly by just changing `base_url`, `api_key` and `model_name`.

Learning Outcomes of the Lab:

- **Mastering Prompting Techniques:** Learn how to effectively prompt large language models to solve specific tasks and to work with the SocialIQA dataset, demonstrating the practical use of LLMs in solving real-world NLP tasks.
- **Embedding Creation and Utilization:** Gain hands-on experience in converting text data into embeddings using embedding model and utilizing these embeddings for information retrieval.
- **Understanding Retrieval-Augmented Generation (RAG):** Learn how to combine pre-trained language models with information retrieval systems to generate contextually relevant responses.

Recommended Reading:

- Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, Graham Neubig. *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing*. https://arxiv.org/abs/2107.13586

- Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, Haofen Wang. *Retrieval-Augmented Generation for Large Language Models: A Survey*. https://arxiv.org/abs/2312.10997

Let's get started!

In [ ]:
```
%pip install -U voyageai
%pip install openai
```

```
Collecting voyageai
  Downloading voyageai-0.2.3-py3-none-any.whl (19 kB)
Collecting aiohttp<4.0,>=3.5 (from voyageai)
  Downloading aiohttp-3.9.5-cp311-cp311-macosx_11_0_arm64.whl (390 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 390.2/390.2 kB 9.6 MB/s eta
0:00:00m eta 0:00:01
Collecting aiolimiter<2.0.0,>=1.1.0 (from voyageai)
  Downloading aiolimiter-1.1.0-py3-none-any.whl (7.2 kB)
Requirement already satisfied: numpy>=1.11 in /Users/rajatjacob/.pyenv/ver
sions/3.11.4/envs/nlp/lib/python3.11/site-packages (from voyageai) (1.26.
4)
Requirement already satisfied: requests<3.0,>=2.20 in /Users/rajatjacob/.p
yenv/versions/3.11.4/envs/nlp/lib/python3.11/site-packages (from voyageai)
(2.31.0)
Collecting tenacity>=8.0.1 (from voyageai)
  Downloading tenacity-8.3.0-py3-none-any.whl (25 kB)
Collecting aiosignal>=1.1.2 (from aiohttp<4.0,>=3.5->voyageai)
  Using cached aiosignal-1.3.1-py3-none-any.whl (7.6 kB)
Requirement already satisfied: attrs>=17.3.0 in /Users/rajatjacob/.pyenv/v
ersions/3.11.4/envs/nlp/lib/python3.11/site-packages (from aiohttp<4.0,>=
3.5->voyageai) (23.2.0)
Collecting frozenlist>=1.1.1 (from aiohttp<4.0,>=3.5->voyageai)
  Downloading frozenlist-1.4.1-cp311-cp311-macosx_11_0_arm64.whl (53 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.4/53.4 kB 9.6 MB/s eta 0:
00:00
Collecting multidict<7.0,>=4.5 (from aiohttp<4.0,>=3.5->voyageai)
  Downloading multidict-6.0.5-cp311-cp311-macosx_11_0_arm64.whl (30 kB)
Collecting yarl<2.0,>=1.0 (from aiohttp<4.0,>=3.5->voyageai)
  Downloading yarl-1.9.4-cp311-cp311-macosx_11_0_arm64.whl (81 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.2/81.2 kB 13.7 MB/s eta
0:00:00
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/rajatjac
ob/.pyenv/versions/3.11.4/envs/nlp/lib/python3.11/site-packages (from requ
ests<3.0,>=2.20->voyageai) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /Users/rajatjacob/.pyenv/ve
rsions/3.11.4/envs/nlp/lib/python3.11/site-packages (from requests<3.0,>=
2.20->voyageai) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/rajatjacob/.py
env/versions/3.11.4/envs/nlp/lib/python3.11/site-packages (from requests<
3.0,>=2.20->voyageai) (2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in /Users/rajatjacob/.py
env/versions/3.11.4/envs/nlp/lib/python3.11/site-packages (from requests<
3.0,>=2.20->voyageai) (2024.2.2)
Installing collected packages: tenacity, multidict, frozenlist, aiolimite
r, yarl, aiosignal, aiohttp, voyageai
Successfully installed aiohttp-3.9.5 aiolimiter-1.1.0 aiosignal-1.3.1 froz
enlist-1.4.1 multidict-6.0.5 tenacity-8.3.0 voyageai-0.2.3 yarl-1.9.4
WARNING: There was an error checking the latest version of pip.
Note: you may need to restart the kernel to use updated packages.
Collecting openai
  Downloading openai-1.34.0-py3-none-any.whl (325 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 325.5/325.5 kB 4.7 MB/s eta
0:00:00[31m3.4 MB/s eta 0:00:01
Requirement already satisfied: anyio<5,>=3.5.0 in /Users/rajatjacob/.pyen
v/versions/3.11.4/envs/nlp/lib/python3.11/site-packages (from openai) (4.
3.0)
Collecting distro<2,>=1.7.0 (from openai)
  Using cached distro-1.9.0-py3-none-any.whl (20 kB)
Requirement already satisfied: httpx<1,>=0.23.0 in /Users/rajatjacob/.pyen
v/versions/3.11.4/envs/nlp/lib/python3.11/site-packages (from openai) (0.2
```

```
7.0)
Collecting pydantic<3,>=1.9.0 (from openai)
  Downloading pydantic-2.7.4-py3-none-any.whl (409 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 409.0/409.0 kB 21.0 MB/s eta
0:00:00
Requirement already satisfied: sniffio in /Users/rajatjacob/.pyenv/version
s/3.11.4/envs/nlp/lib/python3.11/site-packages (from openai) (1.3.1)
Requirement already satisfied: tqdm>4 in /Users/rajatjacob/.pyenv/version
s/3.11.4/envs/nlp/lib/python3.11/site-packages (from openai) (4.66.4)
Requirement already satisfied: typing-extensions<5,>=4.7 in /Users/rajatja
cob/.pyenv/versions/3.11.4/envs/nlp/lib/python3.11/site-packages (from ope
nai) (4.11.0)
Requirement already satisfied: idna>=2.8 in /Users/rajatjacob/.pyenv/versi
ons/3.11.4/envs/nlp/lib/python3.11/site-packages (from anyio<5,>=3.5.0->op
enai) (3.7)
Requirement already satisfied: certifi in /Users/rajatjacob/.pyenv/version
s/3.11.4/envs/nlp/lib/python3.11/site-packages (from httpx<1,>=0.23.0->ope
nai) (2024.2.2)
Requirement already satisfied: httpcore==1.* in /Users/rajatjacob/.pyenv/v
ersions/3.11.4/envs/nlp/lib/python3.11/site-packages (from httpx<1,>=0.23.
0->openai) (1.0.5)
Requirement already satisfied: h11<0.15,>=0.13 in /Users/rajatjacob/.pyen
v/versions/3.11.4/envs/nlp/lib/python3.11/site-packages (from httpcore==1.
*->httpx<1,>=0.23.0->openai) (0.14.0)
Collecting annotated-types>=0.4.0 (from pydantic<3,>=1.9.0->openai)
  Downloading annotated_types-0.7.0-py3-none-any.whl (13 kB)
Collecting pydantic-core==2.18.4 (from pydantic<3,>=1.9.0->openai)
  Downloading pydantic_core-2.18.4-cp311-cp311-macosx_11_0_arm64.whl (1.8
MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 11.6 MB/s eta 0:0
0:0031m37.8 MB/s eta 0:00:01
Installing collected packages: pydantic-core, distro, annotated-types, pyd
antic, openai
Successfully installed annotated-types-0.7.0 distro-1.9.0 openai-1.34.0 py
dantic-2.7.4 pydantic-core-2.18.4
WARNING: There was an error checking the latest version of pip.
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
# from google.colab import drive
# drive.mount('/content/gdrive')
siqa_data_dir = "./data/socialiqa-train-dev/"
plaksha_data_dir = "./data/"
```

In [ ]:
```
# We start by importing libraries that we will be making use of in the as
import json
import os
import random
import re
import time

from collections import Counter
from functools import partial
from pprint import pprint

import numpy as np
import pandas as pd
import tqdm
```

```
import openai
import voyageai
```

Preperation

In [ ]:
```
OPENROUTER_API_KEY = "sk-or-v1-241745dd57230b2a9fcafe079dbf986daced368d63
client = openai.OpenAI(
  base_url="https://openrouter.ai/api/v1",
  api_key=OPENROUTER_API_KEY
)

VOYAGE_API_KEY = "pa-QwlJU1Eaqa5jYXtnfF6YD89NcA8WSBnIjhLPJ6S6tpw" # Your
vo = voyageai.Client(VOYAGE_API_KEY)

GROQ_API_KEY = "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"
client = openai.OpenAI(
  base_url="https://api.groq.com/openai/v1",
  api_key=GROQ_API_KEY
)
```

In [ ]:
```
# Loading the SocialIQA dataset

def load_siqa_data(split):

    # We first load the file containing context, question and answers
    with open(f"data/socialiqa-train-dev/{split}.jsonl") as f:
        data = [json.loads(jline) for jline in f.read().splitlines()]

    # We then load the file containing the correct answer for each questi
    with open(f"data/socialiqa-train-dev/{split}-labels.lst") as f:
        labels = f.read().splitlines()

    return data, labels


train_data, train_labels = load_siqa_data("train")
dev_data, dev_labels = load_siqa_data("dev")

print(f"Number of Training Examples: {len(train_data)}")
print(f"Number of Validation Examples: {len(dev_data)}")
```

```
Number of Training Examples: 33410
Number of Validation Examples: 1954
```

In [ ]:
```
train_data[0]
```

Out[ ]:
```
{'context': 'Cameron decided to have a barbecue and gathered her friends together.',
 'question': 'How would Others feel as a result?',
 'answerA': 'like attending',
 'answerB': 'like staying home',
 'answerC': 'a good friend to have'}
```

In [ ]:
```
train_labels[0]
```

Out[ ]:
```
'1'
```

Task 1: Prompting Basics (30 minutes)

In this task, you will be learning how create standard NLP problems into text prompts which can then be fed to an LLM for its prediction. Mainly there are 2 concepts that are important to understand while creating prompts:

- Prompt Template or Function: a textual string that has two slots: an input slot [X] for input x and an answer slot [Z] for an intermediate generated answer text z that will later be mapped into y.
- Answer verbalizer: A mapping between the task labels to words or phrases that converts the more artificial looking labels to natural language that fits with the prompt. eg. for sentiment analysis we can define Z = {"excellent", "good", "OK", "bad", "horrible"} to represent each of the classes in Y = {++, +, ~, -, --}.

Name	Notation	Example	Description
Input	$\boldsymbol{x}$	I love this movie.	One or multiple texts
Output	$\boldsymbol{y}$	++ (very positive)	Output label or text
Prompting Function	$f_{\text{prompt}}(\boldsymbol{x})$	[X] Overall, it was a [Z] movie.	A function that converts the input into a specific form by inserting the input $\boldsymbol{x}$ and adding a slot [Z] where answer $\boldsymbol{z}$ may be filled later.
Prompt	$\boldsymbol{x}'$	I love this movie. Overall, it was a [Z] movie.	A text where [X] is instantiated by input $\boldsymbol{x}$ but answer slot [Z] is not.
Filled Prompt	$f_{\text{fill}}(\boldsymbol{x}', \boldsymbol{z})$	I love this movie. Overall, it was a bad movie.	A prompt where slot [Z] is filled with any answer.
Answered Prompt	$f_{\text{fill}}(\boldsymbol{x}', \boldsymbol{z}^*)$	I love this movie. Overall, it was a good movie.	A prompt where slot [Z] is filled with a true answer.
Answer	$\boldsymbol{z}$	"good", "fantastic", "boring"	A token, phrase, or sentence that fills [Z]

Table 2: Terminology and notation of prompting methods. $\boldsymbol{z}^*$ represents answers that correspond to true output $\boldsymbol{y}^*$.

We can also include more interesting stuff like instruction of the task in the template and explanation of the answer in the verbalizer to make more powerful prompts, as we will see a bit later.

Task 1.1 Defining prompt function and verbalizer for SocialIQA.

For the purpose of this excercise, we ask you to implement this prompt function:

```
Context: {{context}}
    Question: {{question}}
    Which one of these answers best answers the question
according to the context?
    AnswerA: {{answerA}}
    AnswerB: {{answerB}}
    AnswerC: {{answerC}}
```

and verbalizer:

```
{"1": "The answer is A", "2": "The answer is B", "3": "The
answer is C"}
```

This prompt was obtained from PromptSource, an awesome resource for finding prompts for hundreds of NLP tasks!

In [ ]:
```
def social_iqa_prompting_fn(siqa_example: dict[str, str]):
    """
    Takes an example from the SocialIQA dataset, fills in the prompt temp

    Inputs:
        siqa_example: A dictionary containing the context, question and a

    Outputs:

    """
    prompt = "\n    ".join(
        [
            f"Context: {siqa_example['context']}",
            f"Question: {siqa_example['question']}",
            "Which one of these answers best answers the question accordi
            f"AnswerA: {siqa_example['answerA']}",
            f"AnswerB: {siqa_example['answerB']}",
            f"AnswerC: {siqa_example['answerC']}",
        ]
    )

    return prompt
```

In [ ]:
```
# Sample Test Case 1
print("Running Sample Test Case 1")
siqa_example = train_data[0]
prompt = social_iqa_prompting_fn(siqa_example)
expected_prompt = """Context: Cameron decided to have a barbecue and gath
    Question: How would Others feel as a result?
    Which one of these answers best answers the question according to the
    AnswerA: like attending
    AnswerB: like staying home
    AnswerC: a good friend to have"""
print(f"Input Example:\n{siqa_example}")
print(f"Prompt:\n{prompt}")
print(f"Expected Prompt:\n{expected_prompt}")
assert prompt == expected_prompt

# Sample Test Case 2
print("Running Sample Test Case 2")
siqa_example = train_data[100]
prompt = social_iqa_prompting_fn(siqa_example)
expected_prompt = """Context: Jordan's dog peed on the couch they were se
    Question: How would Jordan feel afterwards?
    Which one of these answers best answers the question according to the
    AnswerA: selling a couch
    AnswerB: Disgusted
    AnswerC: Relieved"""
print(f"Input Example:\n{siqa_example}")
print(f"Prompt:\n{prompt}")
print(f"Expected Prompt:\n{expected_prompt}")
assert prompt == expected_prompt
```

```
Running Sample Test Case 1
Input Example:
{'context': 'Cameron decided to have a barbecue and gathered her friends t
ogether.', 'question': 'How would Others feel as a result?', 'answerA': 'l
ike attending', 'answerB': 'like staying home', 'answerC': 'a good friend
to have'}
Prompt:
Context: Cameron decided to have a barbecue and gathered her friends toget
her.
    Question: How would Others feel as a result?
    Which one of these answers best answers the question according to the
context?
    AnswerA: like attending
    AnswerB: like staying home
    AnswerC: a good friend to have
Expected Prompt:
Context: Cameron decided to have a barbecue and gathered her friends toget
her.
    Question: How would Others feel as a result?
    Which one of these answers best answers the question according to the
context?
    AnswerA: like attending
    AnswerB: like staying home
    AnswerC: a good friend to have
Running Sample Test Case 2
Input Example:
{'context': "Jordan's dog peed on the couch they were selling and Jordan r
emoved the odor as soon as possible.", 'question': 'How would Jordan feel
afterwards?', 'answerA': 'selling a couch', 'answerB': 'Disgusted', 'answe
rC': 'Relieved'}
Prompt:
Context: Jordan's dog peed on the couch they were selling and Jordan remov
ed the odor as soon as possible.
    Question: How would Jordan feel afterwards?
    Which one of these answers best answers the question according to the
context?
    AnswerA: selling a couch
    AnswerB: Disgusted
    AnswerC: Relieved
Expected Prompt:
Context: Jordan's dog peed on the couch they were selling and Jordan remov
ed the odor as soon as possible.
    Question: How would Jordan feel afterwards?
    Which one of these answers best answers the question according to the
context?
    AnswerA: selling a couch
    AnswerB: Disgusted
    AnswerC: Relieved
```

In [ ]:

```python
def social_iqa_verbalizer(label: str):
    """
    Takes in the label and coverts it into a natural language phrase as s

    Inputs:
        label: A string containing the correct answer for a SocialIQA exa

    Outputs:
        A string containing the natural language phrase corresponding to
    """
```

```
    verbalized_label = {"1": "The answer is A", "2": "The answer is B", "

    return verbalized_label[label]
```

In [ ]:
```
# Sample Test Case 1
print("Running Sample Test Case 1")
siqa_example = train_labels[0]
output = social_iqa_verbalizer(siqa_example)
expected_output = """The answer is A"""
print(f"Input Example:\n{siqa_example}")
print(f"output:\n{output}")
print(f"Expected output:\n{expected_output}")
assert output == expected_output

# Sample Test Case 2
print("\nRunning Sample Test Case 2")
siqa_example = train_labels[100]
output = social_iqa_verbalizer(siqa_example)
expected_output = """The answer is B"""
print(f"Input Example:\n{siqa_example}")
print(f"output:\n{output}")
print(f"Expected output:\n{expected_output}")
assert output == expected_output
```

```
Running Sample Test Case 1
Input Example:
1
output:
The answer is A
Expected output:
The answer is A

Running Sample Test Case 2
Input Example:
2
output:
The answer is B
Expected output:
The answer is B
```

Let's now obtain the prompts and verbalized labels for each of the the examples in the dataset

In [ ]:
```
train_prompts = None
train_verbalized_labels = None
val_prompts = None
val_verbalized_labels = None

# YOUR CODE HERE
train_prompts = [social_iqa_prompting_fn(example) for example in train_da
train_verbalized_labels = [social_iqa_verbalizer(label) for label in trai
val_prompts = [social_iqa_prompting_fn(example) for example in dev_data]
val_verbalized_labels = [social_iqa_verbalizer(label) for label in dev_la
```

In [ ]:
```
# Sample Test Case 1
print("Running Sample Test Case 1")
idx = 10
siqa_example = train_data[idx]
prompt = train_prompts[idx]
```

```python
expected_prompt = """Context: Sydney was a school teacher and made sure t
    Question: How would you describe Sydney?
    Which one of these answers best answers the question according to the
    AnswerA: As someone that asked for a job
    AnswerB: As someone that takes teaching seriously
    AnswerC: Like a leader"""
print(f"Input Example:\n{siqa_example}")
print(f"Prompt:\n{prompt}")
print(f"Expected Prompt:\n{expected_prompt}")
assert prompt == expected_prompt

# Sample Test Case 2
print("\nRunning Sample Test Case 2")
idx = 10
siqa_label = train_labels[idx]
output = social_iqa_verbalizer(siqa_label)
verbalized_label = "The answer is B"
print(f"Input Example:\n{siqa_label}")
print(f"Verbalized Label:\n{verbalized_label}")
print(f"Expected Verbalized Label:\n{verbalized_label}")
assert output == verbalized_label
```

```
Running Sample Test Case 1
Input Example:
{'context': 'Sydney was a school teacher and made sure their students lear
ned well.', 'question': 'How would you describe Sydney?', 'answerA': 'As s
omeone that asked for a job', 'answerB': 'As someone that takes teaching s
eriously', 'answerC': 'Like a leader'}
Prompt:
Context: Sydney was a school teacher and made sure their students learned
well.
    Question: How would you describe Sydney?
    Which one of these answers best answers the question according to the
context?
    AnswerA: As someone that asked for a job
    AnswerB: As someone that takes teaching seriously
    AnswerC: Like a leader
Expected Prompt:
Context: Sydney was a school teacher and made sure their students learned
well.
    Question: How would you describe Sydney?
    Which one of these answers best answers the question according to the
context?
    AnswerA: As someone that asked for a job
    AnswerB: As someone that takes teaching seriously
    AnswerC: Like a leader

Running Sample Test Case 2
Input Example:
2
Verbalized Label:
The answer is B
Expected Verbalized Label:
The answer is B
```

It is often useful to have a reverse verbalizer as well that converts the verbalized labels back to the structured and consistent labels in the dataset. For example, "The answer is A" is mapped back to "1" and so on.

In [ ]:
```
def social_iqa_reverse_verbalizer(verbalized_label: str):
    """
    Reverses the verbalized label into the label
    Inputs:
        verbalized_label: A string containing the natural language phrase
    Outputs:
        label: A string containing the correct answer for a SocialIQA exa

    Important Note: We will be using this function to map LLM's output to
    For example, it can be "The answer is A" or "The answer is A." or or
    When you reverse the verbalized label, make sure you handle these cas

    Important Note 2: If the resulting text doesn't have the answer, then

    HINT: use regex pattern matching.
    """

    matches = re.search(r'answer is ([ABC])', verbalized_label)
    if not matches:
        return ""
    answer, = matches.groups()
    label = str("ABC".find(answer)+1)

    return label
```

In [ ]:
```
# Sample Test Case 1
print("Running Sample Test Case 1")
example_verbalized_label = "The answer is C"
output = social_iqa_reverse_verbalizer(example_verbalized_label)
expected_output = "3"
print(f"Input Example:\n{example_verbalized_label}")
print(f"output:\n{output}")
print(f"Expected output:\n{expected_output}")
assert output == expected_output

# Sample Test Case 2
print("\nRunning Sample Test Case 2")
example_verbalized_label = "The answer is B"
output = social_iqa_reverse_verbalizer(example_verbalized_label)
expected_output = "2"
print(f"Input Example:\n{example_verbalized_label}")
print(f"output:\n{output}")
print(f"Expected output:\n{expected_output}")
assert output == expected_output

# Sample Test Case 3
print("\nRunning Sample Test Case 3")
example_verbalized_label = "some explanation before the actual answer, Th
output = social_iqa_reverse_verbalizer(example_verbalized_label)
expected_output = "1"
print(f"Input Example:\n{example_verbalized_label}")
print(f"output:\n{output}")
print(f"Expected output:\n{expected_output}")
assert output == expected_output

# Sample Test Case 4
print("\nRunning Sample Test Case 4")
example_verbalized_label = "some text here the answer is C, some more tex
output = social_iqa_reverse_verbalizer(example_verbalized_label)
```

```python
expected_output = "3"
print(f"Input Example:\n{example_verbalized_label}")
print(f"output:\n{output}")
print(f"Expected output:\n{expected_output}")
assert output == expected_output

# Sample Test Case 5
print("\nRunning Sample Test Case 5")
example_verbalized_label = "none of the options is the correct answer"
output = social_iqa_reverse_verbalizer(example_verbalized_label)
expected_output = ""
print(f"Input Example:\n{example_verbalized_label}")
print(f"output:\n{output}")
print(f"Expected output:\n{expected_output}")
assert output == expected_output
```

```
Running Sample Test Case 1
Input Example:
The answer is C
output:
3
Expected output:
3

Running Sample Test Case 2
Input Example:
The answer is B
output:
2
Expected output:
2

Running Sample Test Case 3
Input Example:
some explanation before the actual answer, The answer is A
output:
1
Expected output:
1

Running Sample Test Case 4
Input Example:
some text here the answer is C, some more text
output:
3
Expected output:
3

Running Sample Test Case 5
Input Example:
none of the options is the correct answer
output:

Expected output:
```

Task 1.2: Choose Few-Shot examples

Often we can get better performance on a task by providing a few examples of the task as part of the prompt. This is also known as in-context learning, where the model learns to solve a task based on the examples provided in the context (and no updates to the model's weights!). One of the easiest way that works reasonably well in practice is to simply choose `k` examples randomly for each class from the entire training dataset, such that we have n_classes * k few-shot examples where n_classes = 3 for SocialIQA dataset. Implement the `choose_few_shot` function below that does that.

In [ ]:
```
from collections import defaultdict
```

In [ ]:
```
def choose_few_shot(train_prompts, train_verbalized_labels, k = 1, seed =
    """
    Randomly chooses k examples from the training set for few-shot in-con
    Inputs:
        train_prompts: A list of prompts for the training set.
        train_verbalized_labels: A list of labels for the training set.
        k: The number of examples per class to choose.
        n_classes: The number of classes in the dataset.
        seed: The random seed to use, to ensure reproducible outputs

    Outputs:
        - List[Dict[str, str]]: A list of 3k examples from the training s

    Example Output: [
        {
            "prompt": <Example Prompt 1>,
            "label": <Example Label_1>
        },
        ...,
        {
            "prompt": <Example Prompt 3k>,
            "label": <Example Label_3k>
        }
    ]
    """

    random.seed(seed)
    np.random.seed(seed)

    indexes = defaultdict(set)
    for i in range(len(train_prompts)):
        indexes[train_labels[i]].add(i)

    fs_examples = []

    for label, idx in indexes.items():
        indexes = np.random.choice(list(idx), size=k)

        for i in indexes:
            fs_examples.append({
                'prompt': train_prompts[i],
                'label': train_verbalized_labels[i]
            })

    # Shuffle the examples to ensure there is no bias in the order of the
    random.shuffle(fs_examples)
```

```
    return fs_examples
```

In [ ]:
```
# Sample Test Case 1
print("Running Sample Test Case 1. Checking if the output length is corre
k = 1
seed = 42
output = choose_few_shot(train_prompts, train_verbalized_labels, k, seed)
output_len = len(output)
expected_output_len = k * len(set(train_labels))
print(f"k: {k}")
print(f"Output Length:\n{output_len}")
print(f"Expected Output Length:\n{expected_output_len}")
assert output_len == expected_output_len

# Sample Test Case 2
print("\nRunning Sample Test Case 2. Checking if all labels are predicted
output_labels = sorted(list(set([example["label"] for example in output])
expected_output_labels = ["The answer is A", "The answer is B", "The answ
print(f"Output Labels:\n{output_labels}")
print(f"Expected Output Labels:\n{expected_output_labels}")
assert output_labels == expected_output_labels

# Sample Test Case 3
print("\nRunning Sample Test Case 3. Checking if count of labels are corr
k = 3
output = choose_few_shot(train_prompts, train_verbalized_labels, k, seed)
output_label_counter = Counter(list(([example["label"] for example in out
expected_output_counter = {"The answer is A": k, "The answer is B": k, "T
print(f"For k = {k}")
print(f"Output Label Counter:\n{output_label_counter}")
print(f"Expected Output Label Counter:\n{expected_output_counter}")
assert output_label_counter == expected_output_counter
```

```
Running Sample Test Case 1. Checking if the output length is correct
k: 1
Output Length:
3
Expected Output Length:
3

Running Sample Test Case 2. Checking if all labels are predicted
Output Labels:
['The answer is A', 'The answer is B', 'The answer is C']
Expected Output Labels:
['The answer is A', 'The answer is B', 'The answer is C']

Running Sample Test Case 3. Checking if count of labels are correct
For k = 3
Output Label Counter:
Counter({'The answer is B': 3, 'The answer is C': 3, 'The answer is A':
3})
Expected Output Label Counter:
{'The answer is A': 3, 'The answer is B': 3, 'The answer is C': 3}
```

In [ ]:
```
# Choose 3 few-shot examples from training data
few_shot_examples = choose_few_shot(train_prompts, train_verbalized_label
```

Few-shot examples with explanations

So far above we have been constructing label verbalizer to provide the answer directly. Often it can be useful to prompt the model to first generate an explanation before the answer. For eg.

```
"prompt": "Context: Tracy didn't go home that evening and resisted Riley's attacks.
            Question: What does Tracy need to do before this?
            Options:
            (A) make a new plan
            (B) Go home and see Riley
            (C) Find somewhere to go"
"label": "Tracy found somewhere to go and didn't come home because she wanted to resist Riley's attacks. Hence, the answer is C"
```

One way to prompt the model to generate such explanations is to provide the explanations for the few-shot examples, which will ground the model to first generate an explanation and then the answer. This helps both improve the performance of the model as well as have more interpretable outputs from LLM.

Below we provide a few examples with explanations for SocialIQA task obtained from Super-NaturalInstructions, an amazing resource for prompts, instructions and explanations for around 1600 NLP tasks.

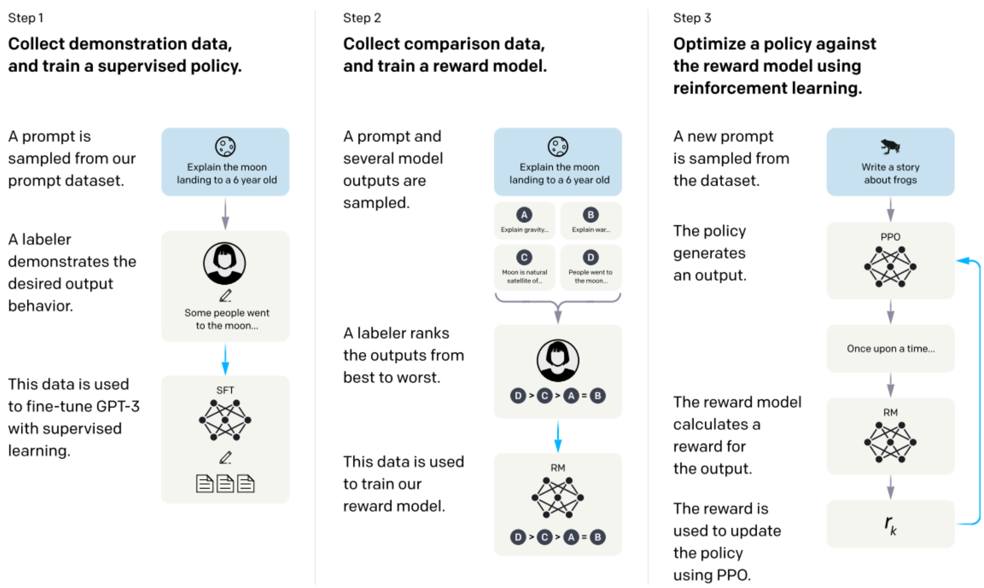
In [ ]:
```
fs_examples_w_explanations = [
    {
        "prompt": "Context: Tracy didn't go home that evening and resiste
        "label": "Tracy found somewhere to go and didn't come home becaus
    },
    {
        "prompt": "Context: Sydney walked past a homeless woman asking fo
        "label": "Sydney is a sympathetic person because she felt bad for
    },
    {
        "prompt": "Context: Taylor gave help to a friend who was having t
        "label": "The friend should thank Taylor for the generosity she s
    }
]
```

In [ ]:
```
fs_examples_w_explanations
```

Out[ ]:
```
[{'prompt': "Context: Tracy didn't go home that evening and resisted Ril
ey's attacks.\nQuestion: What does Tracy need to do before this?\nWhich
one of these answers best answers the question according to the context?
\nAnswerA: make a new plan\nAnswerB: Go home and see Riley\\AnswerC: Fin
d somewhere to go",
  'label': "Tracy found somewhere to go and didn't come home because she
wanted to resist Riley's attacks. Hence, the correct answer is C."},
 {'prompt': 'Context: Sydney walked past a homeless woman asking for cha
nge but did not have any money they could give to her. Sydney felt bad a
fterwards.\nQuestion: How would you describe Sydney?\nWhich one of these
answers best answers the question according to the context?\nAnswerA: sy
mpathetic\nAnswerB: like a person who was unable to help\nAnswerC: incre
dulous',
  'label': "Sydney is a sympathetic person because she felt bad for some
one who needed help, and she couldn't help her. Hence, the correct answe
r is A."},
 {'prompt': 'Context: Taylor gave help to a friend who was having troubl
e keeping up with their bills.\nQuestion: What will their friend want to
do next?\nWhich one of these answers best answers the question according
to the context?\nAnswerA: help the friend find a higher paying job\nAnsw
erB: thank Taylor for the generosity\nAnswerC: pay some of their late em
ployees',
  'label': 'The friend should thank Taylor for the generosity she showed
by helping him pay bills. Hence, the correct answer is B.'}]
```

Task 2: Evaluating ChatGPT (GPT-3.5-Turbo) on SocialIQA (45 minutes)

Today we will be working with OpenAI's GPT family of models. ChatGPT (or GPT-3.5) was built on top of GPT-3, which is a pre-trained Large Language Model (LLM) with 175 Billion parameters, trained on a huge amount of unlabelled data using the language modelling objective (i.e. given k tokens, generate (k+1)th token). While this forms the basis of all GPT family of models, GPT-3.5 and later models are based on InstructGPT, which further adds an Instruction Tuning step that learns from human feedback to follow provided instructions.



From the Ouyang et al. 2022

As a consequence of the pre-training with language modeling objective and instruction tuning, we can use GPT-3.5 to complete a given piece of text and provide specific instructions about how to go about completing the text. We achieve this by defining a text prompt which is to be given as the input to the LLM which then generates a completion of the provided text.

In [ ]:
```
response = client.chat.completions.create(
  model="llama3-70b-8192",
  messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the
        {"role": "user", "content": "Where was it played?"}
    ],
  max_tokens=20,
  temperature=0.0,
)
```

Let's try to wrap our head around different parameters to this function call.

First we have `model`, where we specify which OpenAI model to use. We have used `"gpt-3.5-turbo"` here, which is similar to ChatGPT like you would have used online. You can find the list of other models here.

Next, we have `messages`, which contains the conversation between the user and assistant that is to be completed. Notice that the first message is what we call a "system prompt", which is used to set the behavior of the assistant.

`max_tokens` is used to specify the maximum number of response tokens that the model should generate. This can be useful when you know how long the response is typically going to be, and can help reduce cost.

`temperature`, helps in controlling the variability in the output. Lower values for temperature result in more consistent outputs, while higher values generate more diverse and creative results. Setting temperature to 0 will make the outputs mostly deterministic, but a small amount of variability will remain.

In [ ]:
```
response
```

Out[ ]:
```
ChatCompletion(id='chatcmpl-edd67d7d-7b53-4f5c-ac25-2ca23d543a9d', choic
es=[Choice(finish_reason='length', index=0, logprobs=None, message=ChatC
ompletionMessage(content='The 2020 World Series was played at Globe Life
Field in Arlington, Texas. It was a', role='assistant', function_call=No
ne, tool_calls=None))], created=1718272910, model='llama3-70b-8192', obj
ect='chat.completion', system_fingerprint='fp_2f30b0b571', usage=Complet
ionUsage(completion_tokens=20, prompt_tokens=59, total_tokens=79, prompt
_time=0.01430469, completion_time=0.053011911, total_time=0.067316601),
x_groq={'id': 'req_01j08gqryyeqevf3w9d4pzgmqk'})
```

Now let's look at the response. The assistant's reply can be extracted with `response['choices'][0]['message']['content']`. Every response will include a finish_reason. The possible values for finish_reason are:

- stop: API returned complete message, or a message terminated by one of the stop sequences provided via the stop parameter
- length: Incomplete model output due to max_tokens parameter or token limit
- function_call: The model decided to call a function
- content_filter: Omitted content due to a flag from our content filters
- null: API response still in progress or incomplete

Depending on input parameters (like providing functions as shown below), the model response may include different information.

In [ ]:
```
model_output = response.choices[0].message.content
print(model_output)
```

```
The 2020 World Series was played at Globe Life Field in Arlington, Texas.
It was a
```

Task 2.1: Using ChatGPT to solve SocialIQA problems

Now we have an understanding of how to work with OpenAI API, we can go ahead and call the api with the prompts that we just created and check how well does the model perform the task. We promt the model with the test example for which we want the prediction and provide few-shot examples as part of the context. This can be done by simply providing the example prompt and labels as user-assistant conversation history and test example as the most recent query of the user. Implement the function `get_social_iqa_pred_gpt` that receives a test prompt to be answered, few-shot examples, and some api specific hyperparameters to predict the answer.

In [ ]:
```
def get_social_iqa_pred_gpt(
    test_prompt,
    few_shot_examples,
    model_name="llama3-70b-8192",
    max_tokens=20,
    temperature=0.0,
):
    """
    Calls the OpenAI API with test_prompt and few-shot examples to genera
    Inputs:
        test_prompt: The prompt for the test example
        few_shot_examples: A list of few-shot examples
        model_name: The name of the model to use
        max_tokens: The maximum number of tokens to generate
        temperature: The temperature to use for the model

    Outputs:
        model_output: The model's output

    Hint: Your messages to be sent should be in the following format:
        [
            {"role": "user", "content": <fs-example-1-promot>},
            {"role": "assistant", "content": <fs-example-1-label>},
            ...,
```

```
            {"role": "user", "content": <fs-example-3k-promot>},
            {"role": "assistant", "content": <fs-example-3k-label>},
            {"role": "user", "content": <test-prompt>},
        ]
    """

    messages_prompt = [
        {
            "role": "user",
            "content": "You are an expert of Human Social Common Sense. Y
        }
    ]
    for ex in few_shot_examples:
        messages_prompt.extend(
            [
                {"role": "user", "content": ex["prompt"]},
                {"role": "assistant", "content": ex["label"]},
            ]
        )
    messages_prompt.append({"role": "user", "content": test_prompt})

    while True:
        try:
            model_output = client.chat.completions.create(
                model=model_name,
                messages=messages_prompt,
                max_tokens=max_tokens,
                temperature=temperature,
            )
            return model_output.choices[0].message.content
        except (
            openai.APIConnectionError,
            openai.RateLimitError,
            openai.Timeout,
            openai.InternalServerError,
        ) as e:
            # Sleep and try again
            print(f"Couldn't get response due to {e}. Trying again!")
            time.sleep(20)
            continue
```

In [ ]:

```
test_example = val_prompts[0]
test_example_label = val_verbalized_labels[0]
model_output = get_social_iqa_pred_gpt(test_example, few_shot_examples,
                                       model_name = "llama3-70b-8192",
                                       max_tokens = 20, temperature = 0.0
print(test_example)
print(f"Model's response: ", model_output)
print(f"Correct answer: ", test_example_label)
```

```
Context: Tracy didn't go home that evening and resisted Riley's attacks.
    Question: What does Tracy need to do before this?
    Which one of these answers best answers the question according to the
context?
    AnswerA: make a new plan
    AnswerB: Go home and see Riley
    AnswerC: Find somewhere to go
Model's response:  The answer is C
Correct answer:  The answer is C
```

As you can see the model didn't quite get the answer right. Let's try providing examples with explanations i.e. `fs_examples_w_explanations` and see the output. Note that we will need to give a higher value of `max_tokens`, since the model is also expected to generate explanation now.

In [ ]:
```
test_example = val_prompts[0]
test_example_label = val_verbalized_labels[0]
model_output = get_social_iqa_pred_gpt(test_example, fs_examples_w_explan
                                        model_name = "llama3-70b-8192",
                                        max_tokens = 50, temperature = 0.
print(test_example)
print(f"Model's response: ", model_output)
print(f"Correct answer: ", test_example_label)
```

```
Context: Tracy didn't go home that evening and resisted Riley's attacks.
    Question: What does Tracy need to do before this?
    Which one of these answers best answers the question according to the
context?
    AnswerA: make a new plan
    AnswerB: Go home and see Riley
    AnswerC: Find somewhere to go
Model's response:  Tracy didn't go home that evening, which implies that s
he was avoiding Riley's attacks. Therefore, she must have found somewhere
to go instead of going home. Hence, the correct answer is C.
Correct answer:  The answer is C
```

As you can see the output is correct and the explanation also makes sense.

Let's do a full fledged evaluation now. Due to API limits, we will only be evaluating first 32 examples of the validation set and not the whole but that should give us some idea of how good our LLM (LLaMa3-70b) is at solving social common-sense reasoning problems

In [ ]:
```
def get_model_predictions(
        test_prompts,
        few_shot_examples,
        model_name = "llama3-70b-8192",
        max_tokens = 20,
        temperature = 0.0,
):
    """
    Get predictions for all test prompts using the `get_social_iqa_pred_g

    Inputs:
        test_prompts: A list of test prompts
        few_shot_examples: A list of few-shot examples
        model_name: The name of the model to use
        max_tokens: The maximum number of tokens to generate
        temperature: The temperature to use for the model

    Outputs:
        model_preds: A list of model predictions for each test prompt
    """

    model_preds = []
    for prompt in test_prompts:
        pred = get_social_iqa_pred_gpt(prompt, few_shot_examples=few_shot
```

```
        model_preds.append(social_iqa_reverse_verbalizer(pred))

    return model_preds

def evaluate_model_preds(
        model_preds,
        test_labels
):
    """
    Evaluates the prediction of the model by performing string match betw

    Inputs:
        model_preds: A list of model predictions for each test prompt
        test_labels: A list of test labels. Note that these are not verba

    Outputs:
        accuracy: The accuracy of the model i.e. #correct_predictions / #
    """

    matches = 0
    for pred, test in zip(model_preds, test_labels):
        if pred==test:
            matches+=1
    accuracy = matches/len(test_labels)
    return accuracy*100
```

In [ ]:

```
# To test if things are working fine
k = 5
test_prompts = val_prompts[:k]
test_labels = dev_labels[:k]
model_preds = get_model_predictions(test_prompts, few_shot_examples,
                                    model_name = "llama3-70b-8192",
                                    max_tokens = 20, temperature = 0.0)

accuracy = evaluate_model_preds(model_preds, test_labels)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 80.0
```

In [ ]:

```
# Evaluate on 32 validation examples
k = 32
test_prompts = val_prompts[:k]
test_labels = dev_labels[:k]
model_preds = get_model_predictions(test_prompts, few_shot_examples,
                                    model_name = "llama3-70b-8192",
                                    max_tokens = 20, temperature = 0.0)

accuracy = evaluate_model_preds(model_preds, test_labels)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 65.625
```

In [ ]:

```
# Evaluate on 32 validation examples with explanations
k = 32
test_prompts = val_prompts[:k]
test_labels = dev_labels[:k]
model_preds = get_model_predictions(test_prompts,
                                    fs_examples_w_explanations,
                                    model_name = "llama3-70b-8192",
                                    max_tokens = 150, temperature = 0.0)
```

```
accuracy = evaluate_model_preds(model_preds, test_labels)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 75.0
```

Doing 128 examples will take some time of around 30 minutes

In [ ]:
```
# Evaluate on 128 validation examples with explanations
k = 128
test_prompts = val_prompts[:k]
test_labels = dev_labels[:k]
model_output, model_preds = get_model_predictions(test_prompts,
                                    fs_examples_w_explanations,
                                    model_name = "llama3-70b-8192",
                                    max_tokens = 150, temperature = 0.0)
accuracy = evaluate_model_preds(model_preds, test_labels)
print(f"Accuracy: {accuracy}")
```

As you can see we get slightly better performance on prompting the model with explanations than without 78.9% vs 71.875%. We can do more prompt-engineering and better type of explanations to improve the performance further. Also, there may be instances where the answer was correct but our pattern matching didn't catch the correct answer and marked it incorrect.

But we hope with this you would have gotten some idea on how to use these models to solve NLP tasks like this. Also, notice that common sense reasoning remains an open problem for the models we have today, as even with LLMs like LLaMa3, ChatGPT, which are fairly strong LLMs, the accuracy remains isn't as high as we wanted.

Task 3: Retrieval Augmented Generation (RAG)

Now, lets move to another task, where our goal is to create a question answering system for students to ask questions about professors from Plaksha University.

The data for professors has been scrapped from the university full-time faculty webpage and stored in a csv file. The csv file has the following columns:

- name: The name of the professor
- expertise: The area of expertise of the professor
- interest: The research interests of the professor
- about: A short bio/about of the professor

In [ ]:
```
df = pd.read_csv("./data/Plaksha.csv")
df.fillna("N/A", inplace=True)
df.head()
```

Out[ ]:

	expertise	interest	about	name
0	Sustainable Carbon Efficient Energy Systems, S...	Managing renewable uncertainties via battery s...	Dr. Vivek Deulkar is a faculty member at Plaks...	Dr. Vivek Deulkar
1	Energy Efficient and Smart Buildings	Building Energy Informatics,Smart Energy Homes...	Vishal Garg is a University Chair Professor an...	Dr. Vishal Garg
2	Quantum Computing and Cryptography	Quantum Computing and Cryptography,Post-Quantu...	Dr. Tapas Pandit recently joined Plaksha as an...	Dr. Tapas Pandit
3	Financial Economics, Labor Economics, Macroeco...	Financial Economics,Macrofinance,Financial Dis...	Dr. Tanmoy is a financial and labor economist,...	Dr. Tanmoy Majilla
4	DNA repair and DNA damage response	DNA repair and DNA damage response,DNA-protein...	Dr. Swagata Halder aims to pursue his research...	Dr. Swagata Halder

Task 3.1 Creating Embeddings

Create an embedding prompt by combing multiple columns which will be sent to the embedding model to get the final embedding.

Template: `Professor <professor_name>'s Area of Expertise is in <expertise> and some of the professor's Research Interests are <interest>. Here is a short Bio/About of the Professor: <about>`

In [ ]:
```
def embedding_prompt(x):
    """
    This function takes a row of the dataframe as input and returns a str
    The string is a combination of the professor's name, expertise, resea
    """
    return f"Professor {x['name']}'s Area of Expertise is in {x['expertis
```

Apply the embedding prompt to the dataframe to create a new column `prompt` which will be used to get the embeddings.

In [ ]:
```
df['prompt'] = df.apply(embedding_prompt, axis=1)
assert 'prompt' in df.columns, "Prompt column not found"
```

Getting embeddings from the model `voyage-large-2-instruct` and save it as a new column `embedding`

In [ ]:
```
def get_embedding(x, input_type='document'):
    """
    This function takes a prompt string as input and returns the embeddin
    The embedding is obtained using the voyage-large-2-instruct model.
    """
    # time.sleep(25)
```

```
    emb_obj = vo.embed([x], model="voyage-large-2-instruct", input_type=i

    return np.array(emb_obj.embeddings[0])
```

Load the embeddings dataset csv file

In [ ]:
```
df = pd.read_csv("./data/Plaksha_with_embeddings.csv")
df.fillna("N/A", inplace=True)
df.head()
```

Out[ ]:

	expertise	interest	about	name	prompt
0	Sustainable Carbon Efficient Energy Systems, S...	Managing renewable uncertainties via battery s...	Dr. Vivek Deulkar is a faculty member at Plaks...	Dr. Vivek Deulkar	Professor Dr. Vivek Deulkar's Area of Expertis...
1	Energy Efficient and Smart Buildings	Building Energy Informatics,Smart Energy Homes...	Vishal Garg is a University Chair Professor an...	Dr. Vishal Garg	Professor Dr. Vishal Garg's Area of Expertise ...
2	Quantum Computing and Cryptography	Quantum Computing and Cryptography,Post-Quantu...	Dr. Tapas Pandit recently joined Plaksha as an...	Dr. Tapas Pandit	Professor Dr. Tapas Pandit's Area of Expertise...
3	Financial Economics, Labor Economics, Macroeco...	Financial Economics,Macrofinance,Financial Dis...	Dr. Tanmoy is a financial and labor economist,...	Dr. Tanmoy Majilla	Professor Dr. Tanmoy Majilla's Area of Experti...
4	DNA repair and DNA damage response	DNA repair and DNA damage response,DNA-protein...	Dr. Swagata Halder aims to pursue his research...	Dr. Swagata Halder	Professor Dr. Swagata Halder's Area of Experti...

Convert the embedding column which is a string of list to a numpy array

In [ ]:
```
def str2np(x):
    """
    This function takes a string representation of a list as input,
    removes the brackets, splits the string into a list, converts the ele
    and finally converts the list to a numpy array.

    HINT: Use the replace and split functions
    HINT: Remove brackets from string, and using split to convert to norm
    HINT: Covert the elements of list to float
    HINT: Convert list to numpy array
    """
    return np.array(list(map(float, str(x).replace('[', '').replace(']',
```

Apply the `str2np` function to the embedding column

In [ ]:
```
df['embedding'] = df['embedding'].apply(str2np)

assert 'embedding' in df.columns, "Embedding column not found"
assert isinstance(df['embedding'][0], np.ndarray), "Embedding column is n
assert df['embedding'][0].shape == (1024,), "Embedding column is not of s
assert df['embedding'][0].dtype == np.float64, "Embedding column is not o
```

Document In-Memory

Create a numpy 2D array from the embedding column

In [ ]:
```
embeddings = np.stack(df['embedding'])
assert embeddings.shape == (37, 1024), "Incorrect Embedding Shape"
```

Now, we will create a cosine similarity function which will be used to get the similarity between the query and the professors.

In [ ]:
```
def cosine_similarity(a, b):
    """
    This function calculates the cosine similarity between two vectors.

    Parameters:
    a (numpy array): The first vector.
    b (numpy array): The second vector.

    Returns:
    float: The cosine similarity between the two vectors.
    """
    similarity = np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))
    return similarity
```

In [ ]:
```
# Test cases for cosine similarity

a = [1, 2, 3]
b = [4, 5, 6]
print("Your cosine similarity is: ", cosine_similarity(a, b))
print("Expected cosine similarity is: ", 0.9746318461970762)
assert np.isclose(cosine_similarity(a, b), 0.9746318461970762, atol=0.000

a = [0.1, 0.6, 0.8, 0.6, 0.34, 0.78, 0.65, 0.88, 0.1, 0.98, 0.34, 0.77]
b = [0.8, 0.5, 0.44, 0.67, 0.4, 0.6, 0.7, 0.23, 0.87, 0.45, 0.78, 0.98]
print("\nYour cosine similarity is: ", cosine_similarity(a, b))
print("Expected cosine similarity is: ", 0.7814329877768034)
assert np.isclose(cosine_similarity(a, b), 0.7814329877768034, atol=0.000
```

```
Your cosine similarity is:  0.9746318461970762
Expected cosine similarity is:  0.9746318461970762

Your cosine similarity is:  0.7814329877768034
Expected cosine similarity is:  0.7814329877768034
```

Getting the N most similar professors for a given query

In [ ]:
```
def similarProfessor(query, df: pd.DataFrame, embeddings_matrix, k=3):
    """
    This function returns the N most similar professors for a given query

    Parameters:
    query (str): The query for which similar professors are to be found.
    df (pandas DataFrame): The DataFrame containing information about pro
    embeddings_matrix (numpy array): The matrix of embeddings for the pro

    Returns:
    pandas DataFrame: A DataFrame containing the N most similar professor

    # HINT: Get the embedding for the query first, input_type = 'query'
    # HINT: Use the cosine_similarity function to get the similarity betw
    # HINT: Use the np.argsort to get the indices of the most similar pro
    # IMPORTANT: argsort return indices in ascending order, but we need d
    """
    emb = get_embedding(query, input_type='query')
    df=df.copy()
    df['sim'] = df['embedding'].apply(lambda e: cosine_similarity(e, emb)
    return df.sort_values(by='sim', ascending=False).iloc[:k]
    # sim = np.apply_along_axis(lambda e: cosine_similarity(emb, e), arr=
    # idx = np.argsort(sim)[-1:-k-1:-1]
    # return df.iloc[idx]
```

Now, we will create a system prompt which will do cosine similarity with the existing embeddings and get top N most similar professors for a given query.

Template: `You are an helpful assistant, whose role is to help students with their queries. You will be given a context about one or more professors followed by a query from the student about which, what professor is better, or which topic does a particular professor is best at etc. Given the context you have to correctly answer the query and if there is not information in the context regarding the query then you have to answer 'No information available.'`

You can play with the system prompt

In [ ]:
```
system_prompt = "You are an helpful assistant, whose role is to help stud
```

Now for a given query, let's find the top N professor who are related to the query and create the context string for the LLM

In [ ]:
```
def context_template(name, expertise, interest, about):
    return f"""
{name}:
Area of Expertise: {expertise}
The Professor's Research Interests are {interest}
Here is the Bio/About the Professor:
{about}"""
```

In [ ]:
```
query = "I want to be part of a change in Indian education, working with
query = "I want to work with robotics in the healthcare industry."
```

In [ ]:
```
result_df = similarProfessor(query, df, embeddings)

context = ""
for index, row in result_df.iterrows():
    context += context_template(row['name'], row['expertise'], row['inter
    context += "\n\n"

print(context)
```

Dr. Sunita Chauhan:
Area of Expertise: Medical and Surgical Robotics, Robotics and Automated Systems, Cyber Physical Systems
The Professor's Research Interests are Medical/Surgical Robotics, Biomechatronics,Robotics, Mechatronics and AI - in Structural Healthcare, Agriculture, Smart Buildings, Sports Engineering etc.,Ultrasound - Imaging & Diagnostics (Medical/Industrial), Therapeutics and Surgical Ultrasound,Flexible and Soft Robotics - configuration system development for applications in Industrial and Healthcare systems,Intelligent/Smart Systems - Sensing, Monitoring, Manufacturing
Here is the Bio/About the Professor:
Sunita CHAUHAN (PhD, DIC, Medical Robotics, Imperial College of Science Technology and Medicine, London, UK -1999), has recently joined as Professor and Founding director - Center for Centre for Equitable & Personalized Healthcare at Plaksha University. She is Professor (adj.) at the Mechanical and Aerospace Department, Faculty of Engineering and held the positions of Professor and Director of the Robotics and Mechatronics Engineering at Monash University, Australia for almost a decade; Chief Investigator of the BmRAS (Bio- mechatronics, Robotics & Automated Systems) research group. Prior to joining Monash in 2012, she had worked at Nanyang Technological University (Singapore) 1999-2011, Newcastle University (UK) 2011-12 and several other industrial R&D and scientific positions. She had been on several academic/research visiting fellowship/roles at Klinikum Mannheim, Karl Ruprecht's University of Heidelberg, Germany; Kings College, London, UK; NTU, Singapore etc.

Her current research interests include: Medical/Surgical Robotics (comprising state of the art surgical assist technologies such as Computer Assisted and Integrated Surgery (CAS/CIS) systems including safety driven design, development (using both subtractive and additive manufacturing) and intelligent control of novel medical/surgical robotic systems for minimally invasive and non-invasive surgery, Robotic exoskeletons, Intern-replacement, system safety etc.); Surgical training and automated assessment using AI based deep-learning methodologies; Intelligent Diagnostics and Robotics in Infrastructural Healthcare for inspection and proactive maintenance (Railways, Aerospace, Defence, Agriculture, Buildings, Solar farms etc); Sports Eng.- high performance swimming, cycling, archery etc.

Professor Chauhan is a member of several prestigious professional organizations, such as senior member-IEEE and its Robotics & Automation Society, life member-IACAS, member of UIA and ISTU & SPIE (past). She had been an invited key-note speaker and panel member in various Intl’ conferences and many scientific and public events; invited to hold special conference sessions; general chair for Sports Eng. conference; invited for several expert Intl’ review panels: ASTAR (SG), NHMRC (Australia), FP7/8 and Horizon2020, ESF European grants. She serves on the boards of directors and advisory of several organizations/institutions globally.

She has been conferred various awards and accolades internationally, notably - IEEE Asia Pacific Most Inspiring Engineer of the Year Award -by IEEE R10 (Asia and Pacific region) for contribution towards advancing technology for humanity through her work on Surgical Robotic Systems for Management of Breast Cancers; ROAR & OASIS fellowship with professional attachments in Europe; the Public-sector Innovation award (TEC -The Entrepreneur Challenge), Prime minister office, SG) along with her enterprising students, 2006; Hind Rattan Award (Jewel of India), by NRI Soc., c/o Govt. of India, nominated for Sword of Honor (Pravasi Bhartiya Divas) and Mahatama Gandhi Samman (organized at the House of Commons London), Commonwealth scholarship/fellowship award to pursue PhD in Robotics & AI by British Council.

Her work caught Intl' media attention several times and published in leadi
ng newspapers and scientific magazines. She has supervised national and in
ternationally-funded research projects and is a sole/principal inventor of
several patents granted/pending related to her research. She delivered mor
e than 70 invited talks and participated in various short courses/workshop
s conducted by local as well as overseas organizations. She is a reviewer
of several international journals and conferences, and participates active
ly in their program and organizing committees as specialized session-chai
r, track-chair, publicity-chair etc.

Dr. Sandeep Manjanna:
Area of Expertise: Robotics and Applied Machine Learning
The Professor's Research Interests are Robotics for Marine and Agricultur
e,Path Planning,Reinforcement Learning,Adaptive Sampling for Environmental
Monitoring
Here is the Bio/About the Professor:
Dr. Sandeep Manjanna is a founding faculty at Plaksha University. His rese
arch is in applied machine learning and robotics. Before this, he was a Ja
mes McDonnell Postdoctoral Fellow at GRASP Labs in Computer and Informatio
n Science at University of Pennsylvania, Philadelphia, PA, USA. His resear
ch interests include robotic active sampling, multi-robot coordination, ro
botic sensor networks, reinforcement learning, and machine learning.

Dr. Manjanna received his MSc and PhD from McGill University advised by Pr
of. Gregory Dudek. His doctoral research focuses on designing algorithms f
or autonomous vehicles to sample, understand, and map challenging environm
ents. Dr Manjanna's PhD thesis was awarded CIPPRS John Barron Dissertation
Award in Robotics (2021) for its impact in the field of Robotics in Canad
a.

Dr. Shashank Tamaskar:
Area of Expertise: Robotic autonomy, Vision based robot navigation
The Professor's Research Interests are Multi-Agent Robotic Path Planning &
Control,Flexible Robotic Control & Manipulation
Here is the Bio/About the Professor:
Prior to joining Plaksha, Dr. Tamaskar worked as a Research Scientist & Pr
oject Manager in Siemens Corporation, where he led a team of robotics and
machine learning engineers to work on complex R&D projects related to adva
nced manufacturing and automation. The research topic areas included flexi
ble robotic control leveraging the digital twin of the manufacturing proce
ss, flexible pick and place operations and interoperability between roboti
c and manufacturing systems. These projects were funded by the Advanced Ro
botics in Manufacturing (ARM) Institute, which is DoD sponsored public-pri
vate partnership aimed at improving the adoption of robotics in manufactur
ing.

Dr. Tamaskar received his BTech from IIT Bombay in 2005 and completed his
MS and Ph.D. in Aeronautics & Astronautics from Purdue University in 2011
and 2014 respectively. After his Ph.D., he worked as a Technical Specialis
t in Advanced Controls Research Group at Cummins Inc. where he developed a
lgorithms for state estimation and utilized model predictive techniques fo
r internal combustion engines. His research led to significant improvement
s in engine fuel efficiency and reduction in emissions and the work is cur
rently being deployed in production engines and has been awarded several p
atents. He has received several patents and awards for his research such a
s the Boeing Excellence Award for outstanding research. He also founded an

```
d led the development of the first student satellite of IIT Bombay, Pratha
m, which was launched by Indian Space Research Organization in September 2
016.

At Plaksha, Dr. Tamaskar wants to pursue industry-focused research in Mult
i-Agent Robotic Path Planning and Control and Flexible Robotic Manipulatio
n with the long-term vision of improving the adoption of robotics in Indi
a. While countries like China and South Korea have heavily invested in rob
otics and automation, India is far behind. India's robot density is only 4
industrial robots per ten thousand employees, far below China (732)/South
Korea (2589). Some of the challenges associated with automation and roboti
cs for medium and small industries are lack of flexible technologies, high
acquisition and sustainment cost, dearth of technical support, interoperab
ility with legacy automation equipment, and paucity of indigenous solution
s and trained workforce. He is actively seeking partnership opportunities
with like-minded individuals in pursuit of this long-term vision and help
accelerate the adoption of robotics and automation.
```

In [ ]:

```
response = client.chat.completions.create(
    model= "llama3-70b-8192",
    messages=[
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": f"Context: \n\n {context}.\n\n Query:
    ],
)
```

In [ ]:

```
print(response.choices[0].message.content)
```

```
Based on the context, Dr. Sunita Chauhan is the professor who is best suit
ed to work with robotics in the healthcare industry. Her area of expertise
includes Medical and Surgical Robotics, Biomechatronics, and Robotics and
Automated Systems, which aligns with your interest in robotics in the heal
thcare industry.
```