Introduction

Welcome to this handbook where we embark on a classification task with the OpenAI API. Our journey begins with setting up all required libraries and diving into the dataset for sentiment classification. Drawing from our previous learnings, we'll employ prompt engineering to generate classified responses from the dataset. Lastly, we'll predict and evaluate the LLM's performance to assess its practical utility.

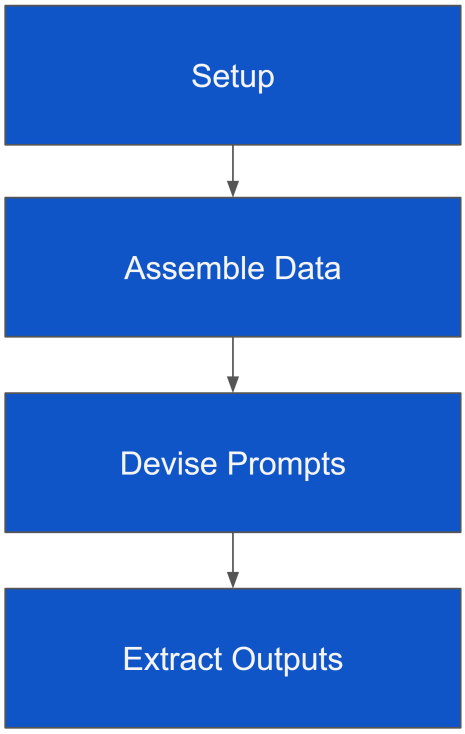


Figure 1

1. Setup

Section Overview: *To begin, we'll first set up the required code for the classification task. We'll start by installing and importing all the necessary libraries. Then, we'll proceed to authenticate with the OpenAI API. Finally, we'll create a token counter function to monitor the token usage via the API.*

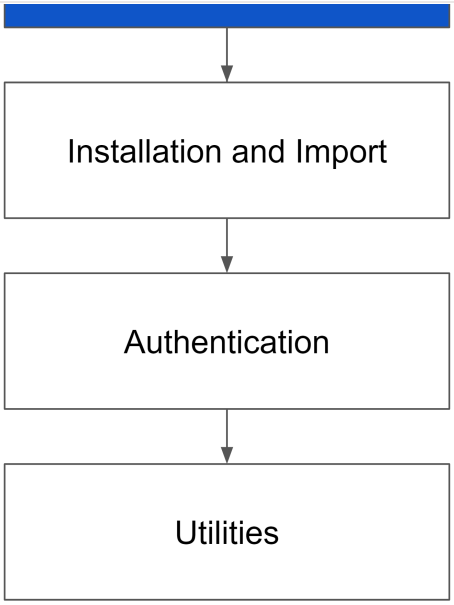


Figure 2

1.1 Installation and Import

Section Overview: *The provided code snippet sets up the Python environment for a classification task using the Azure OpenAI API and data processing libraries. It installs necessary packages, imports essential modules for data handling, tokenization, and model prediction, and displays session informatic for reference. This setup prepares the environment for loading datasets, preprocessing data, training models, and evaluating performance metrics effectively.*

Python----------

```
!pip install openai==1.2.0 tiktoken datasets session-info --quiet
```

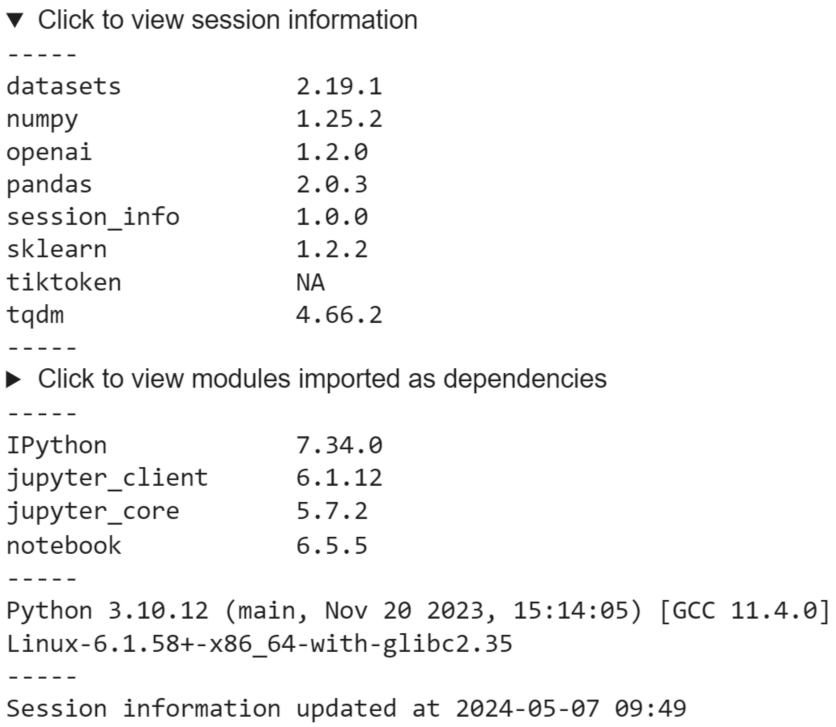
```
import json
import random
import tiktoken
import session_info

import pandas as pd
import numpy as np

from openai import AzureOpenAI
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
```

```
session_info.show()
```



▼ Click to view session information

```
-----
datasets            2.19.1
numpy               1.25.2
openai              1.2.0
pandas              2.0.3
session_info        1.0.0
sklearn             1.2.2
tiktoken            NA
tqdm                4.66.2
-----
```

▶ Click to view modules imported as dependencies

```
-----
IPython             7.34.0
jupyter_client      6.1.12
jupyter_core        5.7.2
notebook            6.5.5
-----
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
Linux-6.1.58+-x86_64-with-glibc2.35
-----
Session information updated at 2024-05-07 09:49
```

Python----------

- **Install Libraries:** We will install all the necessary libraries here.
  - `!pip install openai==1.2.0 tiktoken datasets session-info --quiet`: This lin is installing several libraries using pip, which is the package installer for Python.
  - This is the same step we did earlier for the prompt engineering week.

- **Import Libraries:** The code then imports the libraries that were installed. Most of the libraries imported here are already discussed previously. Let us discuss the libraries we are importing for the first time
  - `import random`: This line imports the random library, which provides a way to generate random numbers. We will need to select random sample of examples from large dataset, so the random library will be used.
  - `import tiktoken`: The tiktoken library is likely needed for tokenizing text data, which is a common task in natural language processing. Here we will count the number of tokens consumed in input and response generation by LLM.

perform numerical operations using it.

- `from datasets import load_dataset`: This line imports the `load_dataset` function from the `datasets` library. The `load_dataset` function is needed to load a dataset, which is a collection of data we will use as input to LLM for classifcation.
- `from collections import Counter`: This line imports the `Counter` class from the `collections` library.The `Counter` class is likely needed to count the frequency of certain elements in a dataset.
- `from tqdm import tqdm`: This line imports the `tqdm` function from the `tqdm` library.The `tqdm` function is needed to display a progress bar, which can be useful for tracking the progress of long-running tasks.This will help us display progress of predictions made for the dataset.
- `from sklearn.model_selection import train_test_split`: This line imports the `train_test_split` function from the `sklearn.model_selection` module.The `train_test_split` function is likely needed to split a dataset into training and testing sets, which is a common task in machine learning.
- `from sklearn.metrics import f1_score`: This line imports the `f1_score` function from the `sklearn.metrics` module. The `f1_score` function is needed to evaluate the performance of a LLM for the classification task, specifically by calculating the F1 score (a measure of precision and recall).

- **Display Session Details:**
  - `session_info.show()`: This line is used to display information about the current session, such as the session ID. Here 'show()' will call the function and we will be able to see the output with the session information.

1.2 Authentication

Section Overview: *This code snippet is used for OpenAI authentication we have already discussed previously. This code snippet is structured to read authentication credentials and configuration details from a JSON file, initialize an AzureOpenAI client with the provided credentials, and extract the deployment name for a classification task. By loading credentials from an external file, the code ensures secure handling of sensitive information and allows for easy configuration changes without modifying the code directly. The extracted information is then used to set up the AzureOpenAI client for interacting with the Azure OpenAI API, facilitating classification tasks with the specified model and endpoint.*

Python----------

```
creds = json.loads(data)

client = AzureOpenAI(
 azure_endpoint=creds["AZURE_OPENAI_ENDPOINT"],
 api_key=creds["AZURE_OPENAI_KEY"],
 api_version=creds["AZURE_OPENAI_APIVERSION"]
)

deployment_name = creds["CHATGPT_MODEL"]
```

Python----------

- **Upload and Read Configuration File:** Here we will read the config.json file. This is the same file we created in previous week of the course (Prompt Engineering at Scale) in the API Setup Hands-on Handbook.
  - `with open(...)`: This is a built-in Python function that opens a file and returns a file objec This will be used to read the file
- **Credentials Assignment:**
  - Assign the values from the `creds` dictionary to the corresponding attributes of the `openai module.These credentials are read by the openai function and act as authentication step.
  - `client = AzureOpenAI(...)`: This is the function that creates an instance of the AzureOpenAI class. This class is used to authenticate to the personalized Open AI model server and make requests to the API.
- **Deployment Name:**
  - `deployment_name = creds["CHATGPT_MODEL"]`: Assign the deployment name of the cha completion endpoint to the **deployment_name** variable.
  - This name is used to specify which chat model should be accessed when making requests to the Azure OpenAI API.

1.3 Utilities

Section Overview: *The purpose of this function is to provide a way to track token consumption by the OpenAI API. This is important because OpenAI's API charges based on the number of tokens processed*

message individually and adding the number of tokens in each message to the total number of tokens.

Python----------

```
def num_tokens_from_messages(messages):

    encoding = tiktoken.encoding_for_model("gpt-3.5-turbo")

    # Each message is sandwiched with <|start|>role and <|end|>
    # Hence, messages look like: <|start|>system or user or assistant{message}<|end|>
    tokens_per_message = 3
    # token1:<|start|>, token2:system(or user or assistant), token3:<|end|>

    num_tokens = 0

    for message in messages:
        num_tokens += tokens_per_message
        for key, value in message.items():
              num_tokens += len(encoding.encode(value))

    num_tokens += 3  # every reply is primed with <|start|>assistant<|message|>

    return num_tokens
```

Python----------

- **Define Token Counter Function:** This function will keep track of the total tokens consumed by inp and output message by the API.
    - `def num_tokens_from_messages(messages)`: This line defines a function named `num_tokens_from_messages` that takes one argument, 'messages', which is expected to be a list of messages. This function is used to track token consumption by the OpenAI API, whic is a critical aspect of using the API as it charges based on the number of tokens processed.
    - `encoding = tiktoken.encoding_for_model("gpt-3.5-turbo")`: This line loads the encoding for the "gpt-3.5-turbo" model. The 'tiktoken' library is used to encode and decode text into tokens that can be used with OpenAI's models. The encoding is specific to the mod being used, and different models may have different encodings. This is necessary because OpenAI's API uses tokens as the basic unit of work when processing requests.

used to keep track of the total number of tokens in all messages.

- `for message in messages`: This line starts a loop that will iterate over each message in the `messages` list. This loop is necessary because the function needs to process each message individually to calculate the total number of tokens.
- `num_tokens += tokens_per_message`: This line adds the number of tokens per message to the total number of tokens. Since each message has three tokens, this line adds three to the total number of tokens for each message.
- `for key, value in message.items()`: This line starts a nested loop that will iterate ove each key-value pair in the `message` dictionary. This loop is necessary because the functio needs to process each key-value pair in the message to calculate the total number of tokens
- `num_tokens += len(encoding.encode(value))`: This line adds the number of tokens ir the `value` to the total number of tokens. The `encoding.encode(value)` function is used to encode the `value` into tokens, and the length of the resulting tokens is added to the total number of tokens. This is necessary because OpenAI's API charges based on the number of tokens processed, and this line ensures that the function accurately tracks token consumption.
- `num_tokens += 3`: This line adds three to the total number of tokens. This is because eac reply is primed with `<|start|>assistant<|message|>`, which is three tokens.
- `return num_tokens`: This line returns the total number of tokens in all messages. This is the final result of the function, and it is used to track token consumption by the OpenAI API.

2. Assemble Data

Section Overview: *With all the necessary requirements in place, we can now proceed to load and preprocess the data. We'll ensure it's formatted appropriately for input to the OpenAI API. Additionally, we'll set aside separate gold examples to be used for prediction later.*

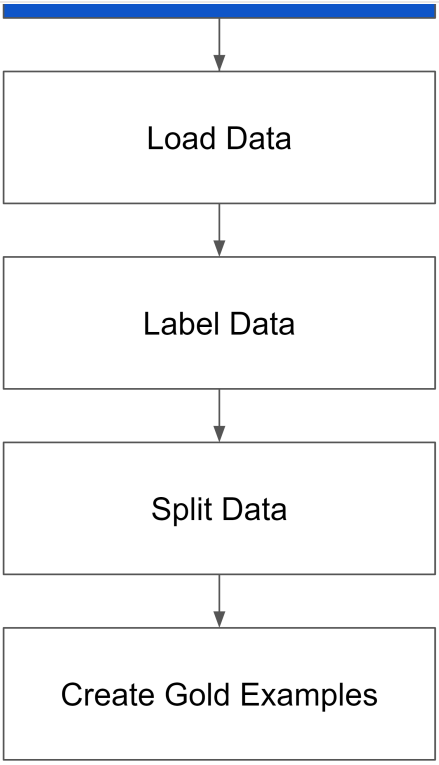


Figure 3

2.1 Load Data

Section Overview: *The provided code loads the dataset, which is a widely used benchmark for text classification tasks. It then converts the training split of the dataset to a Pandas DataFrame for easier data manipulation and analysis. Finally, it inspects the training data by printing the DataFrame information and the count of labels, which helps understand the characteristics of the dataset and prepare for further processing and model training.*

Note: *Here we are goint to use the IMDB Dataset for the classification tasks*

The IMDB movie review dataset is a popular benchmark for sentiment analysis tasks, consisting of 50,000 movie reviews labeled as either positive or negative. The reviews are highly polarized, with positive reviews scoring 7-10 out of 10 and negative reviews scoring 1-4 out of 10. The dataset is balanced, with an equal number of positive and negative reviews, and no more than 30 reviews per movie. You can read more about the dataset here

Python----------

```
# Load the dataset
```

```
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    unsupervised: Dataset({
        features: ['text', 'label'],
        num_rows: 50000
    })
})
```

```
#create dataframe for training data
imdb_reviews_train_df = imdb_reviews_corpus['train'].to_pandas()
```

```
imdb_reviews_train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    25000 non-null  object
 1   label   25000 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 390.8+ KB
```

```
imdb_reviews_train_df.label.value_counts()
```

```
label
0    12500
1    12500
Name: count, dtype: int64
```

Python----------

- **Load Dataset:**
  - `imdb_reviews_corpus = load_dataset("imdb")`: This line loads the IMDB Reviews dataset using the `load_dataset` function from the `datasets` library. The IMDB Reviews dataset is a popular benchmark dataset for text classification tasks, containing movie review labeled as either "positive" or "negative".
  - `imdb_reviews_corpus`: This line simply prints the loaded dataset object, which provides information about the dataset, such as the available splits (e.g., "train", "test", "validation") an the features (e.g., "text", "label".
  - `imdb_reviews_train_df = imdb_reviews_corpus['train'].to_pandas()`: This line

columns.

- `imdb_reviews_train_df.label.value_counts()`: This line counts the number of occurrences of each label (either "positive" or "negative") in the training data. This provides a overview of the class distribution, which is important for understanding the dataset and potential class imbalance issues.

2.2 Label Data

Section Overview: *The provided code adds a new "sentiment" column to the `imdb_reviews_train_df` DataFrame, which contains the textual labels "positive" and "negative" based on the numerical "label" values in the original dataset. This step is common in text classification tasks, as it makes the labels more intuitive and easier to work with.*

Python----------

```
# assign sentiment from number to label
imdb_reviews_train_df['sentiment'] = np.where(imdb_reviews_train_df.label == 1, "positive", "negati
ve")
imdb_reviews_train_df.sentiment.value_counts()
```

```
sentiment
negative    12500
positive    12500
Name: count, dtype: int64
```

Python----------

- **Assigning Sentiment Labels:**
  - `imdb_reviews_train_df['sentiment'] = np.where(imdb_reviews_train_df.labe == 1, "positive", "negative")`: This line creates a new column called "sentiment" in the `imdb_reviews_train_df` DataFrame.
  - `np.where()`: The `np.where()` function is used to assign the sentiment label based on th value in the "label" column.
  - If the "label" value is 1, the sentiment is set to "positive", otherwise, it is set to "negative".
  - This step is necessary because the original dataset uses a numerical label (0 for negative, 1 for positive), and it's often more intuitive to work with textual labels like "positive" and "negative".

- `value_counts()`: This method is used to get the count of unique values in the "sentiment" column.
- This provides an overview of the class distribution in the training data, which is important for understanding the dataset and potential class imbalance issues.

2.3 Split Data

Section Overview: *The provided code splits the `imdb_reviews_train_df` DataFrame into two separate DataFrames: `imdb_reviews_examples_df` and `imdb_reviews_gold_examples_df`. The `train_test_split` function is used to perform this split, with a `test_size` of 0.2, meaning tha 20% of the data is allocated to the "gold" set, and the remaining 80% is used for the "examples" set.*

This split is a common practice in machine learning and natural language processing tasks, as it allows you to have a dedicated "gold" set of examples that can be used for model prediction and evaluation, while the "examples" set is used for model training and development.

Python----------

```
imdb_reviews_examples_df, imdb_reviews_gold_examples_df = train_test_split(
  imdb_reviews_train_df, #<- the full dataset
  test_size=0.2, #<- 20% random sample selected for gold examples
  random_state=42 #<- ensures that the splits are the same for every session
)

(imdb_reviews_examples_df.shape, imdb_reviews_gold_examples_df.shape)
```

```
((20000, 3), (5000, 3))
```

Python----------

- **Splitting the Training Data**:
  - `imdb_reviews_examples_df, imdb_reviews_gold_examples_df = train_test_split(...)`: This line uses the `train_test_split` function from the `sklearn.model_selection` module to split the `imdb_reviews_train_df` DataFram into two separate DataFrame.
  - `imdb_reviews_examples_df`: This DataFrame contains the majority (80%) of the training

the training data, which will be used as a "gold" set for model prediction and evaluation.
- `test_size=0.2`: This parameter specifies that 20% of the data should be allocated to the "gold" set, and the remaining 80% should be used for the "examples" set.
- `random_state=42`: This parameter ensures that the same random split is generated every time the code is run, which is important for reproducibility.

- **Inspecting the Split Datasets**:
  - `(imdb_reviews_examples_df.shape, imdb_reviews_gold_examples_df.shape)`: This line prints the shapes (number of rows and columns) of the `imdb_reviews_examples_df` and `imdb_reviews_gold_examples_df` DataFrames.
  - This provides a quick sanity check to ensure that the split was performed correctly, and the "examples" and "gold" sets have the expected sizes.

2.4 Create Gold Examples

Section Overview: *The provided code snippet processes the `imdb_reviews_gold_examples_df` DataFrame to create a JSON representation of 50 gold examples containing the 'text' and 'sentiment' columns. These examples are randomly sampled and serialized into JSON format for further use.*
By selecting specific columns and sampling a subset of data, this code prepares a manageable set of gold examples for prediction and evaluation purposes. The conversion to JSON format allows for easy storage, sharing, and integration with other systems or tools. The final step of loading and displaying th first gold example provides a quick preview of the processed data for verification and further analysis.

Python----------

```
columns_to_select = ['text', 'sentiment']

# Store gold examples in json structure to be used for prediction later
gold_examples = (
imdb_reviews_gold_examples_df.loc[:, columns_to_select].sample(50, random_state=42).to_json(orient
='records'))

json.loads(gold_examples)[0]
```

Python----------

- **Selecting Specific Columns**:

- **Creating Gold Examples JSON**:
  - `gold_examples = (imdb_reviews_gold_examples_df.loc[:, columns_to_select].sample(50, random_state=42).to_json(orient='records'))`: This line selects the columns specified in `columns_to_select` ('text' and 'sentiment') from the `imdb_reviews_gold_examples_df` DataFrame.
  - `sample(50, random_state=42`: This method randomly selects 50 rows from the DataFrame for creating gold examples. The `random_state=42` ensures reproducibility of th random sampling.
  - `to_json(orient='records')` This method converts the selected DataFrame into a JSON format with each row represented as a JSON record. This format is commonly used for data interchange and serialization.

- **Loading and Displaying the First Gold Example**:
  - `json.loads(gold_examples)`: This line loads the JSON-formatted `gold_examples` string using `json.loads()` to convert it back into a Python object (list of dictionaries). `[0]` is used to access the first element (dictionary) in the list, representing the first gold example.

3. Devise Prompts

Section Overview: *Let's craft prompts for the OpenAI API utilizing the chain-of-thought technique. We' generate system message and user message along with the examples for input prompt. Then we will combine them to create the final prompt which we will use to do the classification task.*

Note: *In text to label tasks such as classification, we want the model to understand the expected outpu based on the input. As classification at industry level is a complex task, we want LLM to understand an think before responding.. Hence, we will be using the chain-of-thought prompting technique here out o the three techniques discussed(zero shot, few shot and chain-of-thought).We will also provide the examples in the input prompt for the LLM to understand the expected output. This technique is called chain-of-thought with examples.*

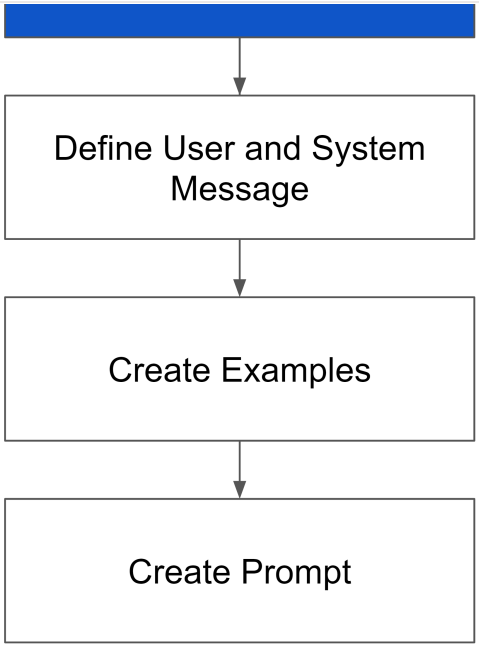


Figure 4

3.1 Define User and System Message

Section Overview: *Let's craft prompts for the OpenAI API utilizing the chain of thought technique. We'll start by creating system message and user message here. Let's start with each section one by one.*

Python----------

```
user_message_template = """```{movie_review}```"""

cot_system_message = """
Classify the sentiment of movie reviews presented in the input as 'positive' or 'negative'.
Movie reviews will be delimited by triple backticks in the input.
Answer only 'positive' or 'negative'. Do not explain your answer.
"""
```

Python----------

- **User Message**:
  - `user_message_template = """```{movie_review}```"""`: This defines a template f

- `cot_system_message` : This message will be used as an instruction to the LLM. The instructions are the asking the model to classify the sentiment of the movie reviews as either 'positive' or 'negative'.

3.2 Create Examples

Section Overview: *The provided code sets up examples for the chain of thought prompting approach f the sentiment classification task. We select a sample of positive and negative examples from the `imdb_reviews_examples_df` DataFrame.Then we are creating a function to create examples. Finally, we call this function and create examples.*

Python----------

```
positive_reviews = (imdb_reviews_examples_df.sentiment == 'positive')
negative_reviews = (imdb_reviews_examples_df.sentiment == 'negative')
(positive_reviews.shape, negative_reviews.shape)
```

```
((20000,), (20000,))
```

```
columns_to_select = ['text', 'sentiment']
positive_examples = imdb_reviews_examples_df.loc[positive_reviews, columns_to_select].sample(4)
negative_examples = imdb_reviews_examples_df.loc[negative_reviews, columns_to_select].sample(4)
```

```
def create_examples(dataset, n=4):
  positive_reviews = (dataset.sentiment == 'positive')
  negative_reviews = (dataset.sentiment == 'negative')
  columns_to_select = ['text', 'sentiment']

  positive_examples = dataset.loc[positive_reviews, columns_to_select].sample(n)
  negative_examples = dataset.loc[negative_reviews, columns_to_select].sample(n)

  examples = pd.concat([positive_examples, negative_examples])

  # sampling without replacement is equivalent to random shuffling
  randomized_examples = examples.sample(2*n, replace=False)
```

```
examples = create_examples(imdb_reviews_examples_df, 2)


json.loads(examples)
```

Python----------

- **Selecting Positive and Negative Examples:**
  - positive_reviews = (dataset.sentiment == 'positive') , negative_reviews = (imdb_reviews_examples_df.sentiment == 'negative'): These lines identify the rows in the `imdb_reviews_examples_df` DataFrame that have a 'positive' or 'negative' sentiment, and store them in the `positive_reviews` and `negative_reviews` variables, respectively
  - (positive_reviews.shape, negative_reviews.shape): This line prints the number of positive and negative examples, providing a quick sanity check on the data..

- **Selecting Sample Examples**:
  - `positive_examples`: This line select a sample of 4 positive examples from the imdb_reviews_examples_df DataFrame, using the previously identified positive_reviews mask
  - `negative_examples`: This line select a sample of 4 negative examples from the imdb_reviews_examples_df DataFrame, using the previously identified and negative_reviews mask.
  - The `columns_to_select` variable specifies that only the 'text' and 'sentiment' columns should be included in the sample.

- **Creating a Reusable Example Generation Function:**
  - `def create_examples(dataset, n=4)`: This function takes a dataset (in this case, `imdb_reviews_examples_df`) and an optional number of examples per class (`n`, default is 4) as input.
  - We will sample out the positive and negative examples here. The code lines follow the same logic as done earlier for selecting `n` positive examples and n negative examples from the input dataset.
  - `randomized_examples = examples.sample(2*n, replace=False)`: These lines concatenate the positive and negative examples, and then randomly shuffle the examples without replacement (equivalent to random shuffling).
  - `return randomized_examples.to_json(orient='records')`: The function returns the randomized examples in JSON format.

- **Generating Chain of Thought Examples**:

- `json.loads(examples)`: This line loads the JSON-formatted examples and displays the first example.

3.3 Create Prompt

Section Overview: *Now that we have created the examples for the prompt, let us create the final prom by combining all the components. We will create a function for the same so that we can call it again when required.W*

Python----------

```
def create_prompt(system_message, examples, user_message_template):

    final_prompt = [{'role':'system', 'content': system_message}]
    for example in json.loads(examples):
        example_review = example['text']
        example_sentiment = example['sentiment']
        final_prompt.append(
            {
                'role': 'user',
                'content': user_message_template.format(
                    movie_review=example_review
                )
            }
        )
        final_prompt.append(
            {'role': 'assistant', 'content': f"{example_sentiment}"}
        )
    return final_prompt
```

```
cot_few_shot_prompt = create_prompt(cot_system_message, examples, user_message_template)
num_tokens_from_messages(cot_few_shot_prompt)
```

1056

Python----------

prompt that includes system messages, user messages with example reviews, and assistant responses.

- `final_prompt = [{'role':'system', 'content': system_message}]`: Initializes the `final_prompt` list with the system message for the task.
- `for example in json.loads(examples)`: Iterates over each example in the JSON-formatted `examples`.
- `example_review = example['text']`: Extracts the review text label from each example.
- `example_sentiment = example['sentiment]`: Extracts the review sentiment label from each example.
- `final_prompt.append(...)`: Appends a user message, assistant message and examples to the `final_prompt.`

- **Calculating the Number of Tokens in the Prompt**:
  - `cot_few_shot_prompt = create_prompt(cot_system_message, examples, user_message_template)`: Calls the `create_prompt` function to generate the final prompt based on the system message, examples, and user message template.
  - `num_tokens_from_messages(cot_few_shot_prompt)`: Calculates the total number of tokens in the constructed final prompt using the `num_tokens_from_messages` function.

4. Extract Outputs

Section Overview: *Now that we have the data prepared and prompts defined, it's time to predict and assess the performance of the prompt created. Let's proceed.*

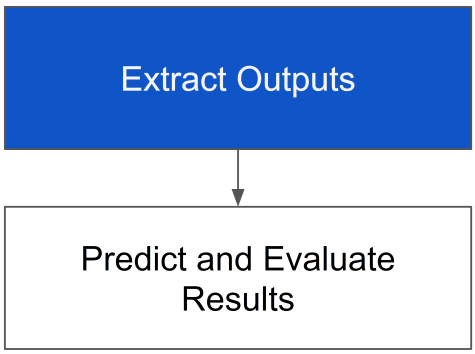


Figure 5

4.1 Predict sentiment with fixed prompt examples

Section Overview: *The purpose of this code is to predict the results and compute the performance of*

Python----------

```
def predict_and_compute_performance(prompt, gold_examples, user_message_template):

    model_predictions, ground_truths = [], []
    for example in json.loads(gold_examples):
        gold_input = example['text']
        user_input = [
            {
                'role':'user',
                'content': user_message_template.format(movie_review=gold_input)
            }
        ]
        try:
            response = client.chat.completions.create(
                model=deployment_name,
                messages=prompt+user_input,
                temperature=0, # <- Note the low temperature
                max_tokens=2 # <- Note how we restrict the output to not more than 2 tokens
            )
            prediction = response.choices[0].message.content
            model_predictions.append(prediction.strip().lower()) # <- removes extraneous white spac
e and lowercases output
            ground_truths.append(example['sentiment'])
        except Exception as e:
            continue
    micro_f1_score = f1_score(ground_truths, model_predictions, average="micro")
    return micro_f1_score
```

```
predict_and_compute_performance(cot_few_shot_prompt, gold_examples, user_message_template)
```

0.91666666666666

Python----------

and user_message_template. The prompt to be evaluated

- **Iterating Over Gold Examples:**
  - `for example in json.loads(gold_examples):` The function iterates over each example in the `gold_examples` JSON-formatted data.
  - `gold_input = example['text']:` The review text from the current gold example is extracted and stored in the `gold_input` variable.
  - `user_input = [{'role':'user', 'content': user_message_template.format(movie_review=gold_input)}]:` A user input messag is constructed using the `user_message_template` and the current `gold_input`.

- **Generating Predictions and Collecting Ground Truths**:
  - `try: ... except Exception as e: continue:` This block of code attempts to generate a prediction from the Azure OpenAI API using the provided `prompt` and