QuestionAnswering

March 3, 2026

1 Question Answering (QA) on semi-structured data using LLMs

One of the most fundamental capabilities of LLMs is that they can serve as a **semantic bridge** connecting textual descriptions that have few words in common, if any. For example, if a restaurant R advertizes that they serve seafood and a customer C expresses interest in a restaurant that serves shrimp-based dishes, an LLM can determine that R is a good match for C, even though the restaurant does not explicitly advertises they serve shrimp-based dishes.

In this assignment, you will use the Gemini API to help users find restaurants by answering queries that express constraints on location, cuisine or food type, and amosphere. The assignment is designed as a sequence of steps as follows:

1. Load and preprocess restaurant and customer review data that is stored in JSON files.
2. Use the Gemini API (gAPI) to process the user query and extract intended *location* and *cuisine*.
3. Find the restaurant(s) that satisfy the location and cuisine constraints.
    - Use the LLM to determine if a restaurant data indicates they offer the required cuisine.
    - Use the LLM to also provide explanations for its response above.
4. If more than one restaurant found, output the one that has the highest star rating.
5. Alternatively, if more than one restaurant found, give preference to the ones that have a *fun ambiance* or atmosphere.
    - If multiple restaurants have a fun ambiance, output the one with the highest star rating.
    - If no restaurant has a fun ambience, output the one with the highest star rating.

For bonus points, manually label the reviews with a binary "fun ambience" *label* and use this together with the ccAPI and customer reviews to compute the accuracy of ccAPI on the same labeling task.

1.1 Write Your Name Here:

2 Submission Instructions

1. Click the Save button at the top of the Jupyter Notebook.
2. Please make sure to have entered your name above.
3. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of ll cells).
4. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.
5. Once you've rerun everything, select File -> Download as -> PDF via LaTeX and download a PDF version *QuestionAnswering.pdf* showing the code and the output of all cells, and save

it in the same folder that contains the notebook file *QuestionAnswering.ipynb*.

6. Look at the PDF file and make sure all your solutions are there, displayed correctly. The PDF is the only thing we will see when grading!
7. Submit **both** your PDF and notebook on Canvas.
8. Make sure your your Canvas submission contains the correct files by downloading it after posting it on Canvas.

2.1 Preprocessing of restaurant and review data (15 points)

Load and preprocess restaurant and customer review data that is stored in JSON files. The files store information regarding the restaurants and customer reviews for a very small subset of the Yelp dataset. Look at the format of the data in the two files to understand its structure or read the dataset documentation here.

The Python JSON API is documented here.

[ ]:
```
import json

business = json.load(open('../data/business.json'))
reviews = json.load(open('../data/review.json'))

# Create the `id2business` table that maps the "business_id" to the␣
 ↪corresponding restaurant object.
id2business = {}

# YOUR CODE HERE (5 points)




# Create the `location2business` table that maps a tuple (city, state) to a␣
 ↪list of restaurants at that location.
location2business = {}

# YOUR CODE HERE (5 points)




# For each location show the number of restaurants. For example, the largest␣
 ↪number (8) should be in Philadelphia.
for location in location2business:
    print(location, len(location2business[location]))
```

```
# The "categories" field of a restaurant lists a number of categories, usually related to food served or cuisine.
# For each unique category that is in the data, show the total number of restaurants.
# For example, there are 5 restaurants that are categorized as Mexican.
cat2freq = {}

# YOUR CODE HERE (5 points)









# Print all category --> count mappings. For example, cat2freq['Mexican] should be 5.
print(cat2freq)
```

2.2 Gemini API setup

[2]:
```
import os
from google import genai
from dotenv import load_dotenv, find_dotenv

# Read the local .env file, containing the Gemini secret API key.
_ = load_dotenv(find_dotenv())

client = genai.Client(api_key = os.environ["GEMINI_API_KEY"])
```

2.3 Extract location and cuisine from user query using the Gemini API (25 points)

Process the user query and extract intended *location* and *cuisine*.

The user wants to find a restaurant. Given a query in natural language, use the API to output the following information: * **city**: if city is not mentioned, assume Philadelphia. * **state**: if state is not mentioned, assume most likely state for that city. * **cuisine**: if no cuisine or type of food is mentioned, assume Any.

Note that obtaining the structured data above can be done in multiple ways, such as:

1. Instruct the LLM in the prompt how you want the response to be formatted, e.g. as a Python dictionary (specify what the keys should be) or a JSON format (specify what the properties should be).
2. Supply the JSON schema in the API call, leveraging the Structured Outputs capability.

3. Instruct the LLM to output Python objects through a Pydantic specification, leveraging again the Structured Outputs capability.

Submitting multiple alternative implementations (as the 3 above) can lead to bonus points.

[ ]:
```python
def get_city_state_cuisine(user_message):
    city, state, cuisine = '', '', ''
    # YOUR CODE HERE

    return city, state, cuisine


# Example: user_message = "I am visiting Miami. Can you help me find a␣
 ↪restaurant that serves seafood?"
#    should result in city, state, cuisine = 'Miami', 'FL', 'seafood'
# Example: user_message = "I'm in the mood for a cheeseburger, can you help me␣
 ↪find a restaurant?"
#    should result in city, state, cuisine = 'Philadelphia', 'PA',␣
 ↪'cheeseburger'
# Example: user_message = "Can you help me find a restaurant that serves crab?"
#    should result in city, state, cuisine = 'Philadelphia', 'PA', 'crab'

user_message = "Can you help me find a restaurant that serves crab?"
city, state, cuisine = get_city_state_cuisine(user_message)
```

```
# This should print 'Philadelphia PA crab'.
print(city, state, cuisine)
```

2.4 Find restaurants that satisfy the location and cuisine constraint (40 + 5 points)

We will do this in two steps: 1. Use the `location2business` dictionary to satisfy the location constraint. 2. Find restaurants whose categories match the cuisine. - Use the gAPI to determine if a restaurant data indicates they offer the required cuisine.
- Use the gAPI to also provide explanations for its response above.

2.4.1 Prompt engineering

Calling the LLM's gAPI in a loop to find which restaurants satisfy the cuisine constraint can be expensive in terms of input and output tokens, depending on how many restaurants are at the desired location. Therefore, it is useful to stress-test the prompt first on some examples, before deploying it at scale. By the prompt, we meen the *developer* message and the *user* isntructions that are sent to the LLM.

You can do this in at least two ways:

1. Design the prompt and try it in the Google AI Studio, then once you are happy with the results on a few examples (see the one below) copy the corresponding code from the playground and paste it in the cell below.
2. Design the prompt and try it directly with the LLM's API in the cell below.

[ ]:
```
# An example to try your prompt with.
cuisine = "crab" # cheeseburger
categories = "Burgers, Vegetarian, Restaurants, Vegan, Seafood"

# Play with the rAPI here to figure out how to make the LLM determine if a␣
 ↪restaurant with some 'categories'
# is likely to offer 'cuisine'. Furthermore, in a second interaction, elicit an␣
 ↪explanation from the LM.

# YOUR CODE HERE
```

2.4.2 Deployment at scale

Now that you are happy with the quality of your prompt, let's use it do define a function that can be deployed at scale.

[ ]:
```
# Use the LM to find which restaurants offer that cuisine or food type.
# For each restaurant, store the rationale for the Yes or No answer.
def find_restaurants(city, state, cuisine):
    rlist, rationales = [], []
    # YOUR CODE HERE (40 points)
```

```
    return rlist, rationales
```

[ ]:
```
city = 'Philadelphia'
state = 'PA'
cuisine = 'crab'
results, rationales = find_restaurants(city, state, cuisine)

print()
for r in results:
    print(r['business_id'] + ': ' + r['categories'])

print()

for reason in rationales:
    print('Answer: ' + reason['response'])
    print('Rationale: ' + reason['explanation'])
    print()
```

[ ]:
```
# Given a list of restaurants that were found to satisfy location and cuisine␣
 ↪constraints,
# return the one with the highest star rating.
def argmax_rating(restaurants):
    R = None
    M = 0.0
```

```
    # YOUR CODE HERE (5 points)




    return R, M


R, M = argmax_rating(results)
print('Restaurant ' + R['name'] + ' is likely to offer ' + cuisine + '.')
print('It has a star rating of ' + str(R['stars']) + ' and the following␣
 ↪categories: ' + R['categories'] + '.')
```

2.5 Keep only restaurants that have a fun atmosphere (5 + 40 points)

We will do this by using the Gemini API to determine if a restaurant review indicates there is a fun atmosphere.

[ ]:
```
# Write a function that finds the text of the restaurant review.
def find_review_for_restaurant(rid):
    # YOUR CODE HERE (5 points)




# Use the LM to find which restaurants from 'rlist' have a fun atmosphere.
# For each restaurant, store the rationale for the Yes or No answer.
def find_fun_restaurants(rlist):
    fun_list, explanations = [], []
    # YOUR CODE HERE
```

```
    return fun_list, explanations


fun_list, explanations = find_fun_restaurants(results)
for e in explanations:
    print('Review: ' + e['review'])
    print('Response: ' + e['response'])
    print('Explanations: ' + e['explanation'])
    print()
```

[ ]:
```
# If restaurants with fun ambience were found, return the one with the highest␣
 ↪star rating.
if fun_list:
    R, M = argmax_rating(fun_list)
    print('Restaurant ' + R['name'] + ' is likely to offer ' + cuisine + ' and␣
 ↪has a fun ambience.')
    print('It has a star rating of ' + str(R['stars']) + ' and the review below.
 ↪')
    print(find_review_for_restaurant(R['business_id']))
```

2.6 [Bonus] Manual vs. System annotation of reviews (15 + 15 + 10 points)

Manually label the reviews with a binary fun ambience label and use this to compute the accuracy of LLM on the same labeling task.

1. [15p] For each review in `reviews`, manually add a new key named *label* that is mapped to a value of 1 if you determine that the review indicates a fun atmosphere, otherwise 0. Save this into a new JSON file named `reviews_manual.json`.
2. [15p] For each review in `reviews`, use the LLM API to determine if the review indicates a fun atmosphere. If it does, then add a new key named 'label' that is mapped to a value of 1, otherwise 0. Save this into a new JSON file named `reviews_system.json`.
3. [10p] Write a function `accuracy(manual, system)` that computes the accuracy of the system labels with respect to the manual labels.
    - Look at the cases where the system label is different from yoru manual label and try to explain why.

```
[ ]: # YOUR CODE HERE
```

2.7 Bonus points

Any non-trivial task that is relevant for this assignment will be considered for bonus points. For example:

1. Using the LLM to determine the sentiment of restaurant reviews.
   - Additionally, manually label sentiment and compute the LLM accuracy for sentiment classification.
2. Using the LLM to determine if the review indicates that the restaurant is kid-friendly.
   - Additionally, manually label sentiment and compute the LLM accuracy for this taks as well.
3. Use the much larger Yelp dataset where restaurant have multiple reviews and build a full conversational AI for restaurant search.
   - Incorporate a component that takes as input a user profile listing their preferences and interests, and finds restaurant reviews that contain `atypical aspects` that are likely to surprise the user in a positive way (talk to me or Erfan about this).
   - Run an empirical compairson of GTP models vs. Gemini models vs. Llama on various NLP tasks using this dataset.
   - Investigate methods for making the LLM outputs for the same input more stable.
4. Obtain an estimate of the LM's confidence in its responses and use this confidence estimate to select restaurants that are most likely to satisfy the user's cosntraints:
   - You can use "Verbalized confidence", "Self-consistency confidence", and "Induced Consistency Confidence" as described in the paper Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs.
5. Issuing an API call for each restaurant in `find-restaurants()` can be expensive. Can you think of a cheaper alternative and implement it, where the total number of tokens processed is much smaller?

```
[ ]:
```