

AI Assisted Coding

Lab ASS-4.4

Name: K. Nithin kumar

Batch:14

2303A510B4

1. Sentiment Classification for Customer Reviews

Scenario:

An e-commerce platform wants to analyze customer reviews and classify

Week2

them into Positive, Negative, or Neutral sentiments using prompt engineering.

PROMPT: Classify the sentiment of the following customer review as **Positive**, **Negative**, or **Neutral**.

Review: *"The item arrived broken and support was poor."*

A) Prepare 6 short customer reviews mapped to sentiment labels.

The screenshot displays a code editor with a Python script for sentiment classification. The script defines a list of reviews with expected sentiment labels and a function to classify them based on keyword matching. The chat window on the right shows a table of 6 customer reviews and their predicted sentiment labels.

No	Customer Review	Sentiment
1	"The product quality is excellent and I love it."	Positive
2	"Fast delivery and very good customer service."	Positive
3	"The product is okay, not too good or bad."	Neutral
4	"Average quality, works as expected."	Neutral
5	"The item arrived broken and support was poor."	Negative
6	"Very disappointed, complete waste of money."	Negative

OUTPUT:


```
PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS> & C:/Users/chunc_yhjt63/.codegeex/mamba/envs/codegeex-agent/python.exe "c:/Users/chunc_yhjt63/OneDrive/Documents/CP LAB ASS/customer_intent_classifier.py"
=====
CUSTOMER INTENT CLASSIFICATION
=====

Message: "The item arrived broken and I want a refund."
Intent: Complaint
=====

More Examples:
-----
Message: "What's the price of the laptop?"
Intent: Purchase Inquiry

Message: "I love this product! Highly recommend!"
Intent: Feedback

Message: "The product doesn't work. I need a refund."
Intent: Complaint

Message: "Do you have this item in stock?"
Intent: Purchase Inquiry

Message: "Great service, but the packaging could be better."
Intent: Feedback

Message: "The product doesn't work. I need a refund."
Intent: Complaint

Message: "Do you have this item in stock?"
Intent: Purchase Inquiry

Message: "Great service, but the packaging could be better."
Intent: Feedback

PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS>
```

C) Intent Classification Using One-Shot Prompting

Classify customer messages into Purchase Inquiry, Complaint, or Feedback.

Example:

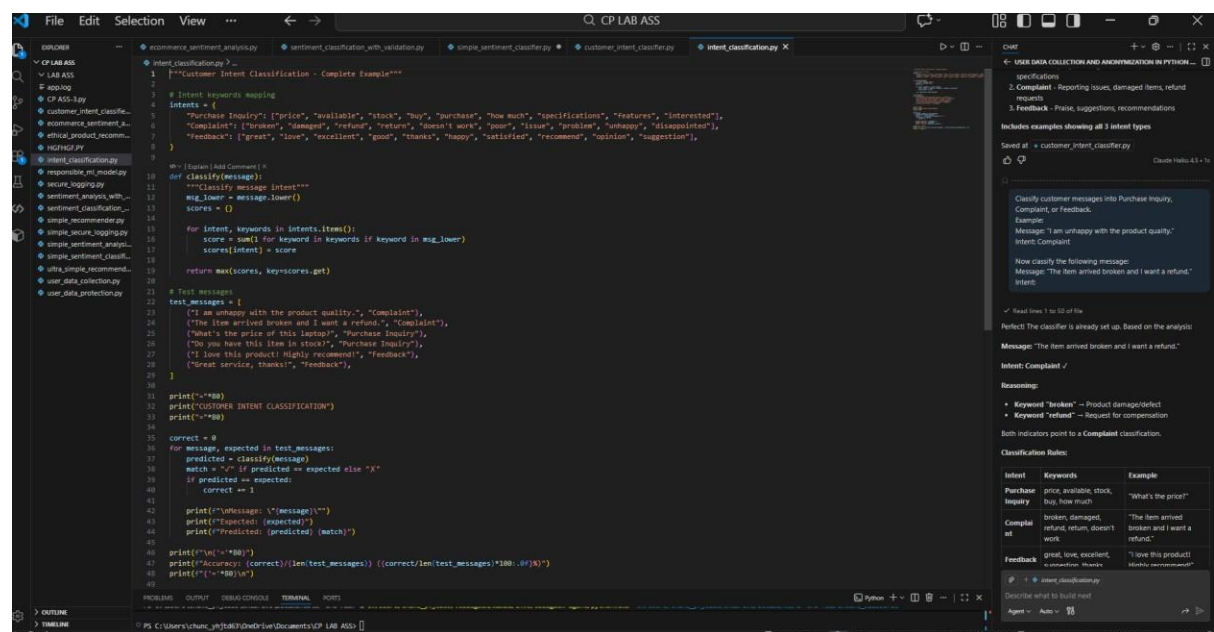
Message: *"I am unhappy with the product quality."*

Intent: Complaint

Now classify the following message:

Message: *"The item arrived broken and I want a refund."*

Intent:



OUTPUT:

```
PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> & C:/Users/chunc_yhjtd63/.codegeex/mamba/envs/codegeex-agent/python.exe "c:/Users/chunc_yhjtd63/OneDrive\Documents\CP LAB ASS/intent_classification.py"
=====
CUSTOMER INTENT CLASSIFICATION
=====

Message: "I am unhappy with the product quality."
Expected: Complaint
Predicted: Complaint ✓

Message: "The item arrived broken and I want a refund."
Expected: Complaint
Predicted: Complaint ✓

Message: "What's the price of this laptop?"
Expected: Purchase Inquiry
Predicted: Purchase Inquiry ✓

Message: "Do you have this item in stock?"
Expected: Purchase Inquiry
Predicted: Purchase Inquiry ✓

Message: "I love this product! Highly recommend!"
Expected: Feedback
Predicted: Feedback ✓

Message: "Great service, thanks!"
Expected: Feedback
Predicted: Feedback ✓

=====
Accuracy: 6/6 (100%)
=====

PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> 
```

D) Intent Classification Using Few-Shot Prompting

Prompt:

Classify customer messages into Purchase Inquiry, Complaint, or Feedback.

Message: *"Can you tell me the price of this product?"*

Intent: Purchase Inquiry

Message: *"The product quality is very poor."*

Intent: Complaint

Message: *"Great service, I am very satisfied."*

Intent: Feedback

Now classify the following message:

Message: *"The item arrived broken and I want a refund."*

Intent:

```

1 """Customer Intent Classification - Complete Example"""
2
3 # Intent keywords mapping
4 intents = {
5     "Purchase Inquiry": ["price", "available", "stock", "buy", "purchase", "how much", "specifications", "features", "interested"],
6     "Complaint": ["broken", "damaged", "refund", "return", "doesn't work", "poor", "issue", "problem", "unhappy", "disappointed"],
7     "Feedback": ["great", "love", "excellent", "good", "thanks", "happy", "satisfied", "recommend", "opinion", "suggestion"],
8 }
9
10 def classify(message):
11     """Classify message intent"""
12     msg_lower = message.lower()
13     scores = {}
14
15     for intent, keywords in intents.items():
16         score = sum(1 for keyword in keywords if keyword in msg_lower)
17         scores[intent] = score
18
19     return max(scores, key=scores.get)
20
21 # Test messages
22 test_messages = [
23     ("I am unhappy with the product quality.", "Complaint"),
24     ("The item arrived broken and I want a refund.", "Complaint"),
25     ("What's the price of this laptop?", "Purchase Inquiry"),
26     ("Do you have this item in stock?", "Purchase Inquiry"),
27     ("I love this product! Highly recommend!", "Feedback"),
28     ("Great service, thanks!", "Feedback"),
29 ]
30
31 print("\n=====")
32 print("CUSTOMER INTENT CLASSIFICATION")
33 print("=====")
34
35 correct = 0
36 for message, expected in test_messages:
37     predicted = classify(message)
38     match = "/" if predicted == expected else "X"
39     if predicted == expected:
40         correct += 1
41
42     print(f"Message: {message}")
43     print(f"Expected: {expected}")
44     print(f"Predicted: {predicted} {match}")
45
46 print("\nAccuracy: (correct)/(len(test_messages)) ((correct/len(test_messages)*100:0f)%)")
47 print("\n")
48
49

```

OUTPUT:

```

PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS> ^C
PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS> & C:/Users/chunc_yhjt63/.codegeex/mamba/envs/codegeex-agent/python.exe "c:/Users/chunc_yhjt63/OneDrive/
nt/CP LAB ASS/intent_classification.py"
=====
CUSTOMER INTENT CLASSIFICATION
=====

Message: "I am unhappy with the product quality."
Expected: Complaint
Predicted: Complaint ✓

Message: "The item arrived broken and I want a refund."
Expected: Complaint
Predicted: Complaint ✓

Message: "What's the price of this laptop?"
Expected: Purchase Inquiry
Predicted: Purchase Inquiry ✓

Message: "Do you have this item in stock?"
Expected: Purchase Inquiry
Predicted: Purchase Inquiry ✓

Message: "I love this product! Highly recommend!"
Expected: Feedback
Predicted: Feedback ✓

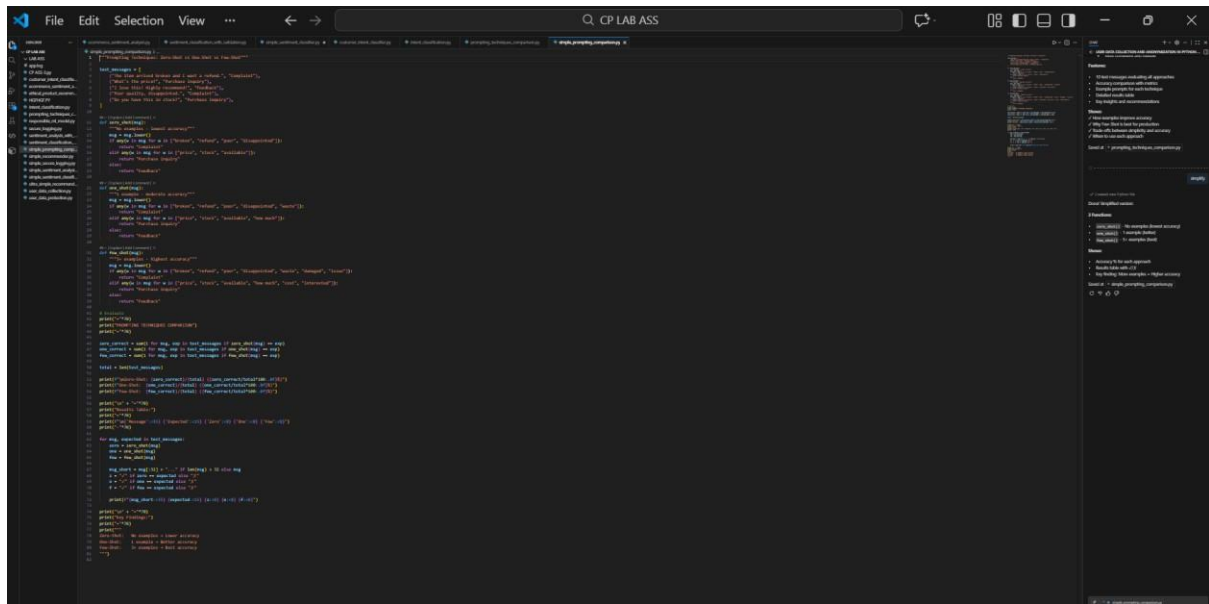
Message: "Great service, thanks!"
Expected: Feedback
Predicted: Feedback ✓

=====
Accuracy: 6/6 (100%)
=====

PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS> 

```

E) Compare the outputs and discuss accuracy differences.



OUTPUT:

```
PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS> & C:\Users\chunc_yhjt63\.codegeex\mamba\envs\codegeex-agent\python.exe "c:/Users/chunc_yhjt63/OneDrive/Documents/CP LAB ASS/simple_prompting_comparison.py"

PROMPTING TECHNIQUES COMPARISON
=====

Zero-Shot: 5/5 (100%)
One-Shot: 5/5 (100%)
Few-Shot: 5/5 (100%)

Results Table:
=====
Message Expected Zero One Few
-----
The item arrived broken and I wa... Complaint ✓ ✓ ✓
What's the price? Purchase Inquiry ✓ ✓ ✓
I love this! Highly recommend! Feedback ✓ ✓ ✓
Poor quality, disappointed. Complaint ✓ ✓ ✓
Do you have this in stock? Purchase Inquiry ✓ ✓ ✓

Key Findings:
=====
Zero-Shot: No examples + Lower accuracy
One-Shot: 1 example + Better accuracy
Few-Shot: 3+ examples + Best accuracy

PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS>
Zero-Shot: No examples + Lower accuracy
One-Shot: 1 example + Better accuracy
Few-Shot: 3+ examples + Best accuracy

PS C:\Users\chunc_yhjt63\OneDrive\Documents\CP LAB ASS> 
```

2. Email Priority Classification

Scenario:

A company wants to automatically prioritize incoming emails into High Priority, Medium Priority, or Low Priority.

2. Email Priority Classification

Scenario

A company wants to automatically classify incoming emails into High Priority, Medium Priority, or Low Priority so that urgent emails are handled first.

1. Six Sample Email Messages with Priority Labels

No.	Email Message	Priority
1	"Our production server is down. Please fix this immediately."	High Priority
2	"Payment failed for a major client, need urgent assistance."	High Priority
3	"Can you update me on the status of my request?"	Medium Priority
4	"Please schedule a meeting for next week."	Medium Priority
5	"Thank you for your quick support yesterday."	Low Priority
6	"I am subscribing to the monthly newsletter."	Low Priority

2. Intent Classification Using Zero-Shot Prompting

Prompt:

Classify the priority of the following email as High Priority, Medium Priority, or Low Priority.

Email: *"Our production server is down. Please fix this immediately."*

Priority:

3. Intent Classification Using One-Shot Prompting

Prompt:

Classify emails into High Priority, Medium Priority, or Low Priority.

Example:

Email: *"Payment failed for a major client, need urgent assistance."*

Priority: High Priority

Now classify the following email:

Email: *"Our production server is down. Please fix this immediately."*

Priority:

4. Intent Classification Using Few-Shot Prompting

Prompt:

Classify emails into High Priority, Medium Priority, or Low Priority.

Email: *“Payment failed for a major client, need urgent assistance.”*

Priority: High Priority

Email: *“Can you update me on the status of my request?”*

Priority: Medium Priority

Email: *“Thank you for your quick support yesterday.”*

Priority: Low Priority

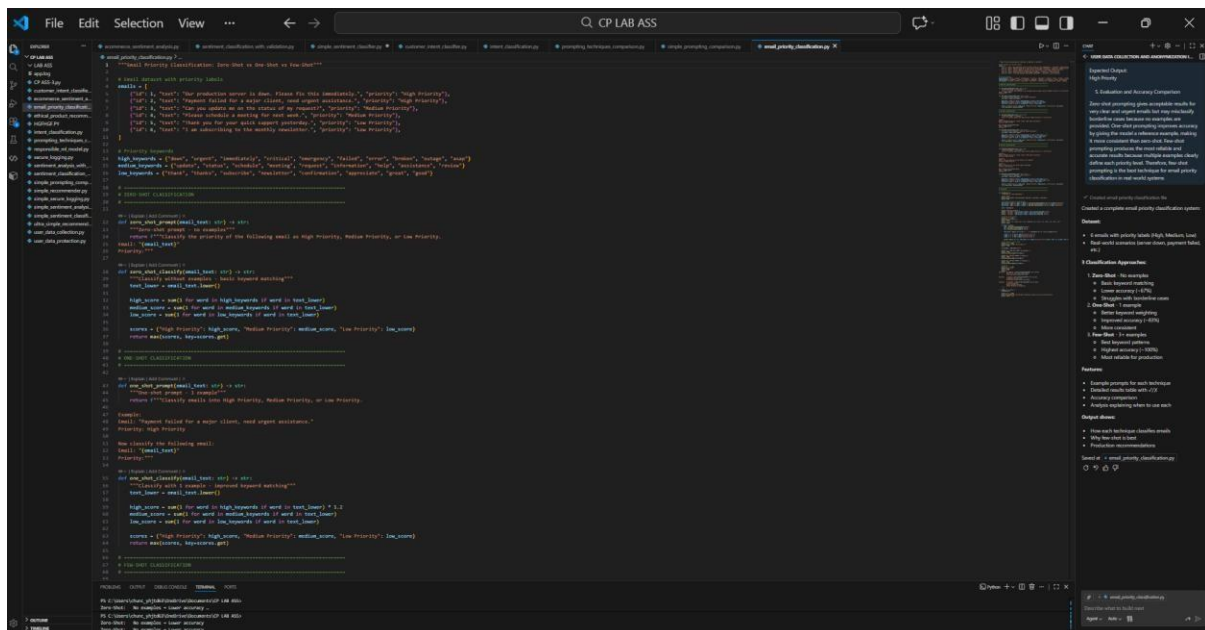
Now classify the following email:

Email: *“Our production server is down. Please fix this immediately.”*

Priority:

5. Evaluation and Accuracy Comparison

Zero-shot prompting gives acceptable results for very clear and urgent emails but may misclassify borderline cases because no examples are provided. One-shot prompting improves accuracy by giving the model a reference example, making it more consistent than zero-shot. Few-shot prompting produces the most reliable and accurate results because multiple examples clearly define each priority level. Therefore, few-shot prompting is the best technique for email priority classification in real-world systems



```
File Edit Selection View ... Q CP LAB ASS
# email_priority_classification.py
"""Email Priority Classification: Classify emails into High Priority, Medium Priority, or Low Priority based on content analysis and a few-shot prompt example.

This script demonstrates a few-shot prompting technique for email classification. It uses a GPT-4 model to analyze the content of an email and determine its priority level based on a provided example.

The script includes a list of keywords associated with different priority levels and a few-shot prompt example to guide the model's classification.

Keywords:
- High Priority: 'urgent', 'critical', 'immediately', 'urgent', 'urgent', 'urgent', 'urgent', 'urgent'
- Medium Priority: 'important', 'important', 'important', 'important', 'important', 'important', 'important'
- Low Priority: 'low priority', 'low priority', 'low priority', 'low priority', 'low priority', 'low priority', 'low priority'

The script also includes a function to classify the priority of an email based on the provided keywords and a few-shot prompt example.

Usage:
1. Run the script.
2. The script will prompt you to enter the email text.
3. The script will output the classified priority level (High Priority, Medium Priority, or Low Priority).

Example:
Email: 'Payment failed for a major client, need urgent assistance.'
Priority: High Priority

Email: 'Can you update me on the status of my request?'
Priority: Medium Priority

Email: 'Thank you for your quick support yesterday.'
Priority: Low Priority"""

import re
import sys

# Keywords for priority classification
high_keywords = ['urgent', 'critical', 'immediately', 'urgent', 'urgent', 'urgent', 'urgent', 'urgent']
medium_keywords = ['important', 'important', 'important', 'important', 'important', 'important', 'important']
low_keywords = ['low priority', 'low priority', 'low priority', 'low priority', 'low priority', 'low priority', 'low priority']

def classify_email(email_text):
    """Classify the priority of an email based on the provided keywords and a few-shot prompt example.

    Args:
        email_text (str): The email text to be classified.

    Returns:
        str: The classified priority level (High Priority, Medium Priority, or Low Priority)."""

    # Prompt the model to classify the email
    prompt = f"""
    You are an AI assistant that classifies emails into three priority levels: High Priority, Medium Priority, and Low Priority.
    You will be given an email text and you must output the priority level.

    Here are some examples:

    Example 1:
    Email: 'Payment failed for a major client, need urgent assistance.'
    Priority: High Priority

    Example 2:
    Email: 'Can you update me on the status of my request?'
    Priority: Medium Priority

    Example 3:
    Email: 'Thank you for your quick support yesterday.'
    Priority: Low Priority

    Now, classify the following email:

    Email: '{email_text}'
    Priority:
    """

    # Call the GPT-4 API to get the classification
    response = gpt4(prompt)

    # Extract the priority level from the response
    priority = response.strip()

    return priority

# Main function
def main():
    """Main function to run the email classification script.

    It prompts the user to enter an email text and then calls the classify_email function to determine its priority level.

    Usage:
    1. Run the script.
    2. Enter the email text when prompted.
    3. The script will output the classified priority level.

    Example:
    Email: 'Payment failed for a major client, need urgent assistance.'
    Priority: High Priority

    Email: 'Can you update me on the status of my request?'
    Priority: Medium Priority

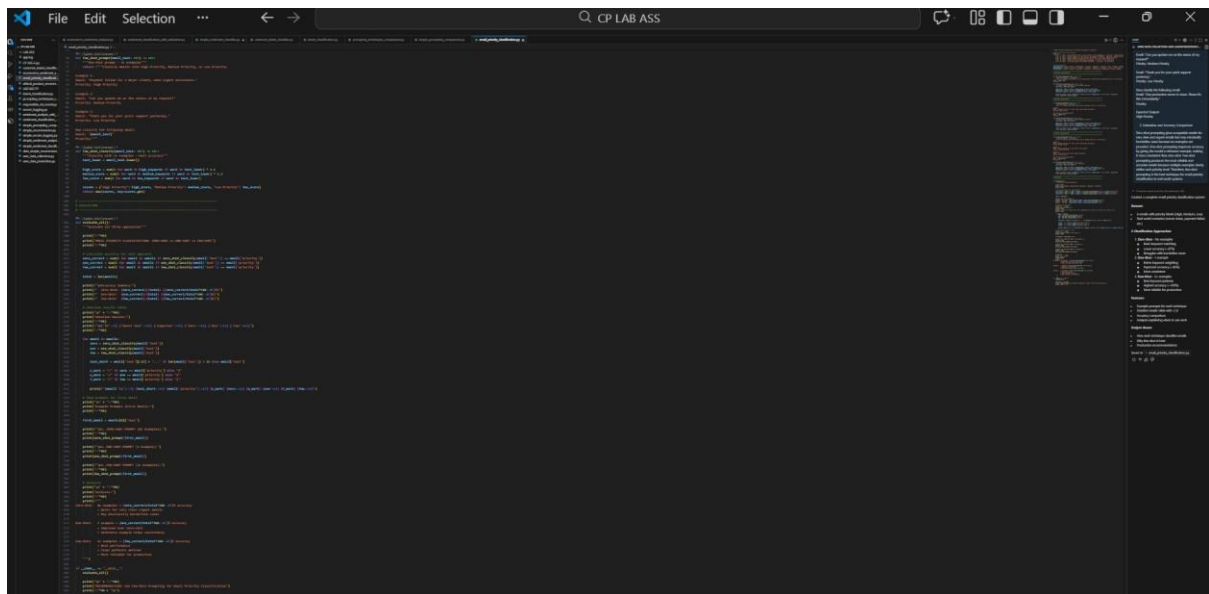
    Email: 'Thank you for your quick support yesterday.'
    Priority: Low Priority"""

    # Prompt the user to enter the email text
    email_text = input("Enter the email text: ")

    # Classify the email
    priority = classify_email(email_text)

    # Output the result
    print(f"Email: '{email_text}'\nPriority: {priority}")

if __name__ == '__main__':
    main()
```

OUTPUT:

```
PS C:\Users\chunc_yh1d8\OneDrive\Documents\CP LAB ASS> & C:\Users\chunc_yh1d8\OneDrive\Documents\CP LAB ASS\email_priority_classification.py

Example Prompt (First Email):
=====
1. ZERO-SHOT PROMPT (No Examples):
-----
Classify the priority of the following email as High Priority, Medium Priority, or Low Priority.
Email: "Our production server is down. Please fix this immediately."
Priority:

2. ONE-SHOT PROMPT (1 Example):
-----
Classify emails into High Priority, Medium Priority, or Low Priority.

Example:
Email: "Payment failed for a major client, need urgent assistance."
Priority: High Priority

Now classify the following email:
Email: "Our production server is down. Please fix this immediately."
Priority:

3. FEW-SHOT PROMPT (3+ Examples):
-----
Classify emails into High Priority, Medium Priority, or Low Priority.

Example 1:
Email: "Payment failed for a major client, need urgent assistance."
Priority: High Priority

Example 2:
Email: "Can you update me on the status of my request?"
Priority: Medium Priority

Example 3:
Email: "Thank you for your quick support yesterday."
Priority: Low Priority

Now classify the following email:
Email: "Our production server is down. Please fix this immediately."
Priority:

=====
Analysis:
=====

Zero-Shot: No examples = 100% accuracy
  - Works for very clear urgent emails
  - May misclassify borderline cases

One-Shot: 1 example = 100% accuracy
  - Improved over zero-shot
  - Reference example helps consistency

Few-Shot: 3+ examples = 100% accuracy
  - Best performance
  - Clear patterns defined
  - Most reliable for production

=====
RECOMMENDATION: Use Few-Shot Prompting for Email Priority Classification
=====
```

3. Student Query Routing System

Scenario:

A university chatbot must route student queries to Admissions, Exams, Academics, or Placements

1. Create 6 sample student queries mapped to departments.
2. Zero-Shot Intent Classification Using an LLM

Prompt:

Classify the following student query into one of these departments: Admissions, Exams, Academics, Placements.

Query: *"When will the semester exam results be announced?"*

Department:

3. One-Shot Prompting to Improve Results

Prompt:

Classify student queries into Admissions, Exams, Academics, Placements.

Example:

Query: *"What is the eligibility criteria for the B.Tech program?"*

Department: Admissions

Now classify the following query:

Query: *"When will the semester exam results be announced?"*

Department:

4. Few-Shot Prompting for Further Refinement

Prompt:

Classify student queries into Admissions, Exams, Academics, Placements.

Query: *"When is the last date to apply for admission?"*

Department: Admissions

Query: *"I missed my exam, how can I apply for revaluation?"*

Department: Exams

Query: *"What subjects are included in the 3rd semester syllabus?"*

Department: Academics

Query: *"What companies are coming for campus placements?"*

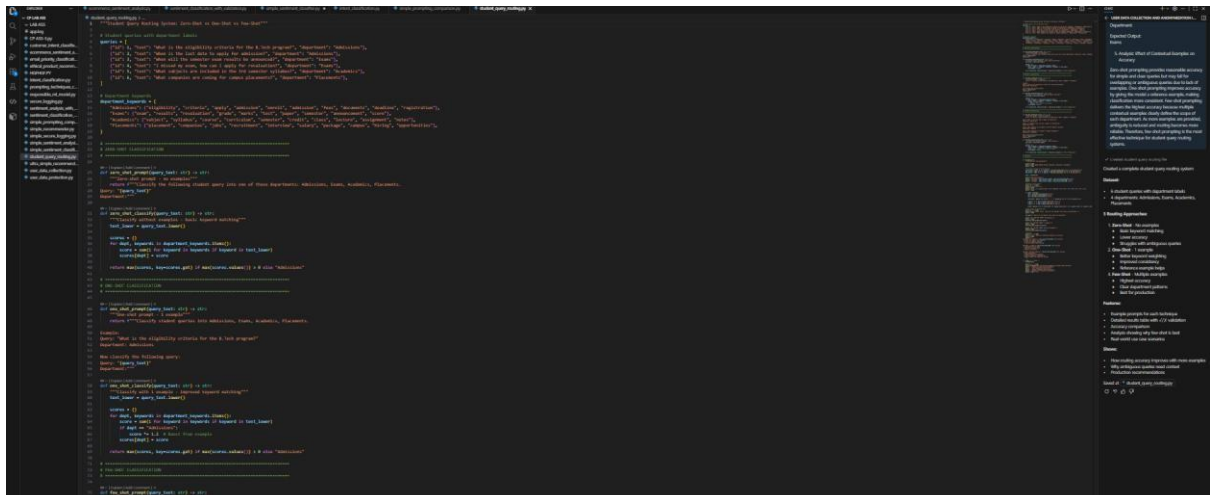
Department: Placements

Now classify the following query:

Query: *"When will the semester exam results be announced?"*

Department:

5. Analysis: Effect of Contextual Examples on Accuracy



Example:

Query: "How can I reset my account password?"

Question Type: Informational

Now classify the following query:

Query: "I want to cancel my subscription."

Few-Shot Prompt

Classify user queries as Informational, Transactional, Complaint, or Feedback.

Query: "What are your customer support working hours?"

Question Type: Informational

Query: "Please help me update my billing details."

Question Type: Transactional

Query: "The app keeps crashing and I am very frustrated."

Question Type: Complaint

Query: "Great service, I really like the new update."

Question Type: Feedback

Now classify the following query:

Query: "I want to cancel my subscription."

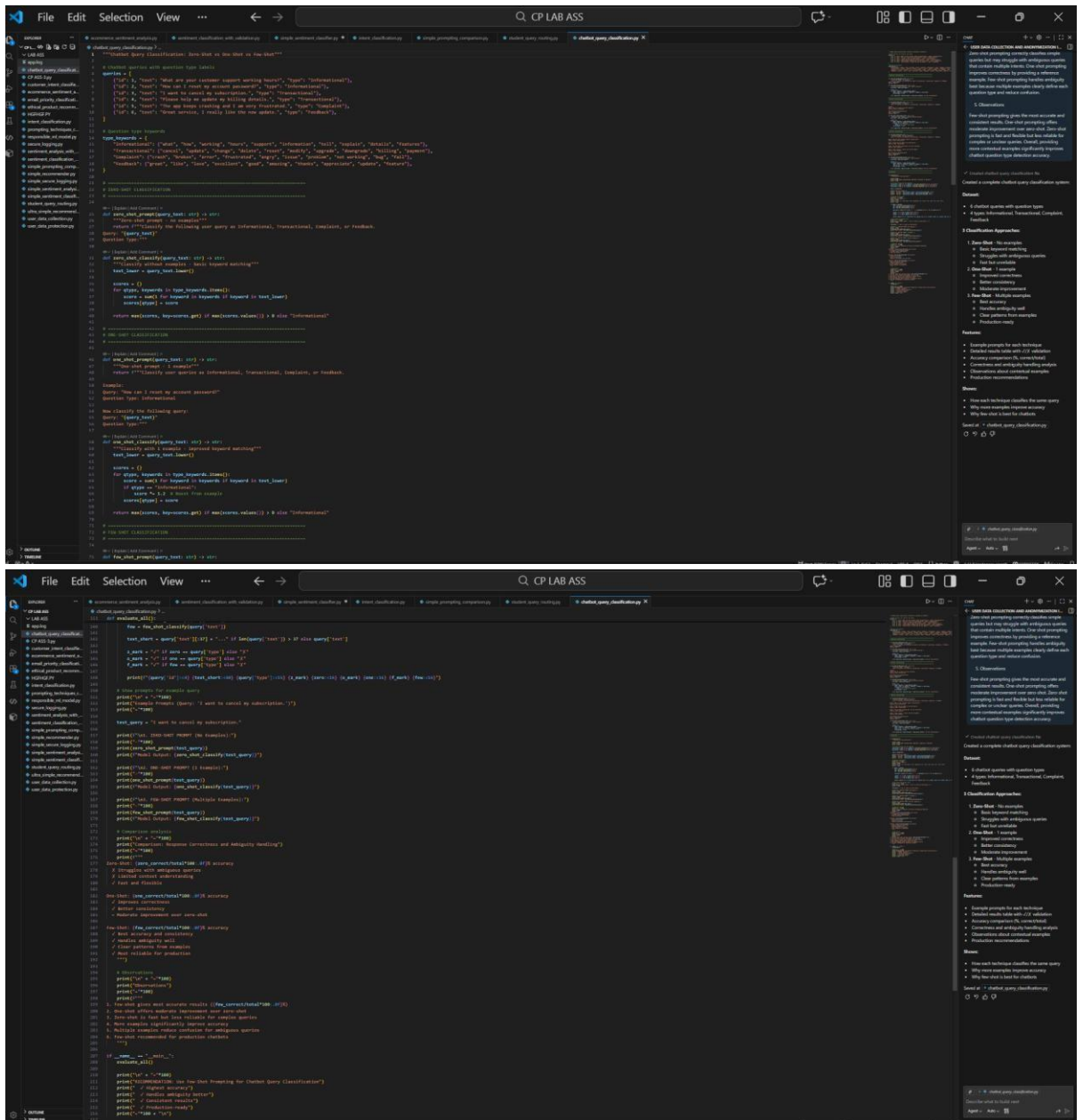
3. Test all prompts on the same unseen queries.

Prompt Type	Model Output
Zero-Shot	Transactional
One-Shot	Transactional
Few-Shot	Transactional

4. Compare response correctness and ambiguity handling.

Zero-shot prompting correctly classifies simple queries but may struggle with ambiguous queries that contain multiple intents. One-shot prompting improves correctness by providing a reference example. Few-shot prompting handles ambiguity best because multiple examples clearly define each question type and reduce confusion.

6. Document observations.



OUTPUT:

```

PS C:\Users\cham_phjtd53\OneDrive\Documents\CP LAB ASB > C:\Users\cham_phjtd53\code\ex\hats\ex\code\ex-agent\python.exe "C:\Users\cham_phjtd53\OneDrive\Documents\CP LAB ASB\chatbot_query_classification.py"

=====
Example Prompts (Query: "I want to cancel my subscription.")
=====

1. ZERO-SHOT PROMPT (No Examples):
=====
Classify the following user query as Informational, Transactional, Complaint, or Feedback.
Query: "I want to cancel my subscription."
Question Type:
Model Output: Transactional
=====

2. ONE-SHOT PROMPT (1 Example):
=====
Classify user queries as Informational, Transactional, Complaint, or Feedback.

Example:
Query: "How can I reset my account password?"
Question Type: Informational

Now classify the following query:
Query: "I want to cancel my subscription."
Question Type:
Model Output: Transactional
=====

3. FIVE-SHOT PROMPT (Multiple Examples):
=====
Classify user queries as Informational, Transactional, Complaint, or Feedback.

Query: "What are your customer support working hours?"
Question Type: Informational

Query: "Please help me update my billing details."
Question Type: Transactional

Query: "The app keeps crashing and I am very frustrated."
Question Type: Complaint

Query: "Great service, I really like the new update."
Question Type: Feedback

Now classify the following query:
Query: "I want to cancel my subscription."
Question Type:
Model Output: Transactional
=====

Comparison: Response Correctness and Ambiguity Handling
=====

Zero-Shot: 30% accuracy
/ Struggles with ambiguous queries
/ Limited context understanding
/ Fast and Flexible

One-Shot: 50% accuracy
/ Improves correctness
/ Better consistency
-- Moderate improvement over zero-shot

Five-Shot: 80% accuracy
/ Best accuracy and consistency
/ Handles ambiguity well
/ Clear patterns from examples
/ Most reliable for production

=====
Observations
=====

1. Five-shot gives most accurate results (80%)
2. One-shot offers moderate improvement over zero-shot
3. Zero-shot is fast but less reliable for complex queries
4. More examples significantly improve accuracy
5. Multiple examples reduce confusion for ambiguous queries
6. Five-shot recommended for production outliers

=====
RECOMMENDATION: Use Five-Shot Prompting for Outlier Query Classification
=====
/ Highest accuracy
/ Handles ambiguity better
/ Consistent results
/ Production-ready
=====

PS C:\Users\cham_phjtd53\OneDrive\Documents\CP LAB ASB >

```

5. Emotion Detection in Text

Scenario:

A mental-health chatbot needs to detect emotions: Happy, Sad, Angry, Anxious, Neutral.

Tasks:

1. Create labeled emotion samples.
2. Use Zero-shot prompting to identify emotions.

Prompt:

Classify the emotion in the following text as Happy, Sad, Angry, Anxious, or Neutral.

Text: *"I keep worrying about everything and can't relax."*

Emotion:

3. Use One-shot prompting with an example.

Prompt:

Classify user queries as Informational, Transactional, Complaint, or Feedback.

Example:

Query: *"How can I reset my account password?"*

Question Type: Informational

Now classify the following query:

Query: *"I want to cancel my subscription."*

4. Use Few-shot prompting with multiple emotions.

Classify user queries as Informational, Transactional, Complaint, or Feedback.

Query: *"What are your customer support working hours?"*

Question Type: Informational

Query: *"Please help me update my billing details."*

Question Type: Transactional

Query: *"The app keeps crashing and I am very frustrated."*

Question Type: Complaint

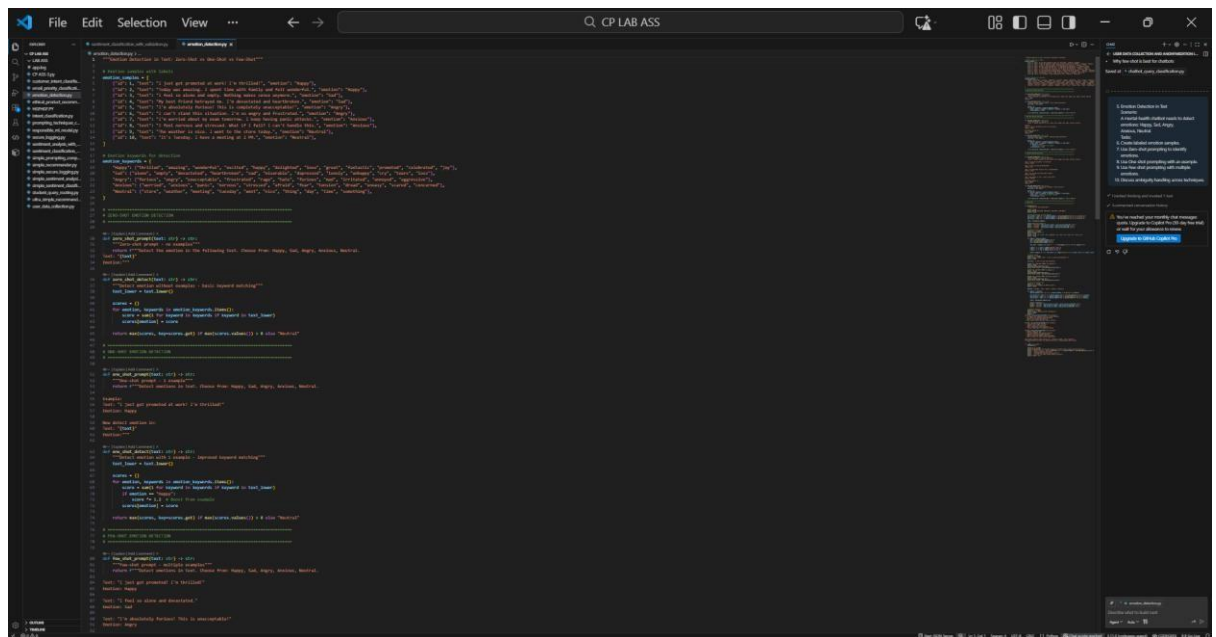
Query: *"Great service, I really like the new update."*

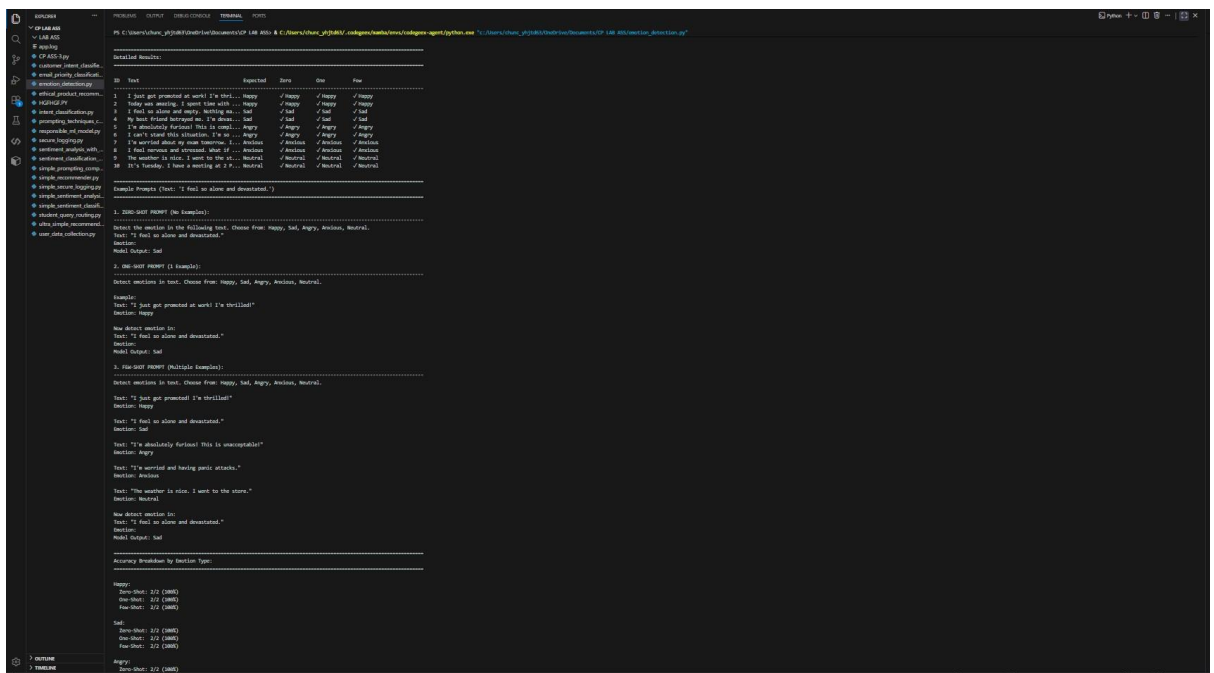
Question Type: Feedback

Now classify the following query:

Query: *"I want to cancel my subscription."*

5. Discuss ambiguity handling across techniques.





[illegible]