

# ASSIGNMENT-4.4

Name:R.Shashideepika

H.NO:2303A51871

B-13

## 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 shows a Jupyter Notebook environment with several files listed in the left sidebar, including `ecommerce_sentiment_analysis.py`, `sentiment_classification_with_validation.py`, and `simple_sentiment_classifier.py`. The main notebook cell contains the following Python code:

```
1  """Simple Sentiment Classification"""
2  reviews = [
3      {"id": 1, "text": "The product quality is excellent and I love it.", "expected": "Positive"},
4      {"id": 2, "text": "Fast delivery and very good customer service.", "expected": "Positive"},
5      {"id": 3, "text": "The product is okay, not too good or bad.", "expected": "Neutral"},
6      {"id": 4, "text": "Average quality, works as expected.", "expected": "Neutral"},
7      {"id": 5, "text": "The item arrived broken and support was poor.", "expected": "Negative"},
8      {"id": 6, "text": "Very disappointed, complete waste of money.", "expected": "Negative"}
9  ]
10 positive_words = ['excellent', 'love', 'great', 'good', 'fast', 'best', 'amazing', 'wonderful', 'perfect', 'quality']
11 negative_words = ['broken', 'poor', 'waste', 'disappointed', 'bad', 'terrible', 'awful', 'hate', 'worst']
12 neutral_words = ['okay', 'average', 'works', 'expected', 'fine', 'normal', 'adequate']
13
14 # Explain | Add Comment | X
15 def classify(review_text):
16     """Classify review sentiment"""
17     text_lower = review_text.lower()
18
19     pos = sum(1 for word in positive_words if word in text_lower)
20     neg = sum(1 for word in negative_words if word in text_lower)
21     neu = sum(1 for word in neutral_words if word in text_lower)
22
23     if pos > neg:
24         return "Positive"
25     elif neg > pos:
26         return "Negative"
27     else:
28         return "Neutral"
29
30 # Classify all reviews
31 print("ID | Expected | Predicted | Review")
32 print("- * 80")
33 correct = 0
34 for item in reviews:
35     predicted = classify(item['text'])
36     match = '+' if predicted == item['expected'] else '-'
37     if predicted == item['expected']:
38         correct += 1
39     review_short = item['text'][0:40] + "..."
40     print(f"\n{item['id']} | {item['expected']} | {predicted} | {review_short} {match}\n")
41 print(f"\nAccuracy: {correct}/{len(reviews)} ({correct/len(reviews)*100:.0f}%)")
42
```

To the right of the code, there is a table titled "Customer Review" with the following data:

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

Below the table, a message says "Created a complete sentiment classification system with your dataset!"

OUTPUT:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ⌂ ⌂ ⌂ ... | ⌂ x

4 | Neutral | Positive | Average quality, works as expected.... X
5 | Negative | Negative | The item arrived broken and support was ... ✓ ...
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/simple_sentiment_classifier.py"
● ID | Expected | Predicted | Review
-----
1 | Positive | Positive | The product quality is excellent and I l... ✓
2 | Positive | Positive | Fast delivery and very good customer ser... ✓
3 | Neutral | Neutral | The product is okay, not too good or bad... ✓
4 | Neutral | Positive | Average quality, works as expected.... X
5 | Negative | Negative | The item arrived broken and support was ... ✓
6 | Negative | Negative | Very disappointed, complete waste of mon... ✓

Accuracy: 5/6 (83%)
○ PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> ⌂
```

### B) Intent Classification Using Zero-Shot Prompting

**Prompt:** Classify the intent of the following customer message as Purchase Inquiry, Complaint, or Feedback.

**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/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_yhjtd63\OneDrive\Documents\CP LAB ASS>
```

### C) Intent Classification Using One-Shot Prompting

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

## Example:

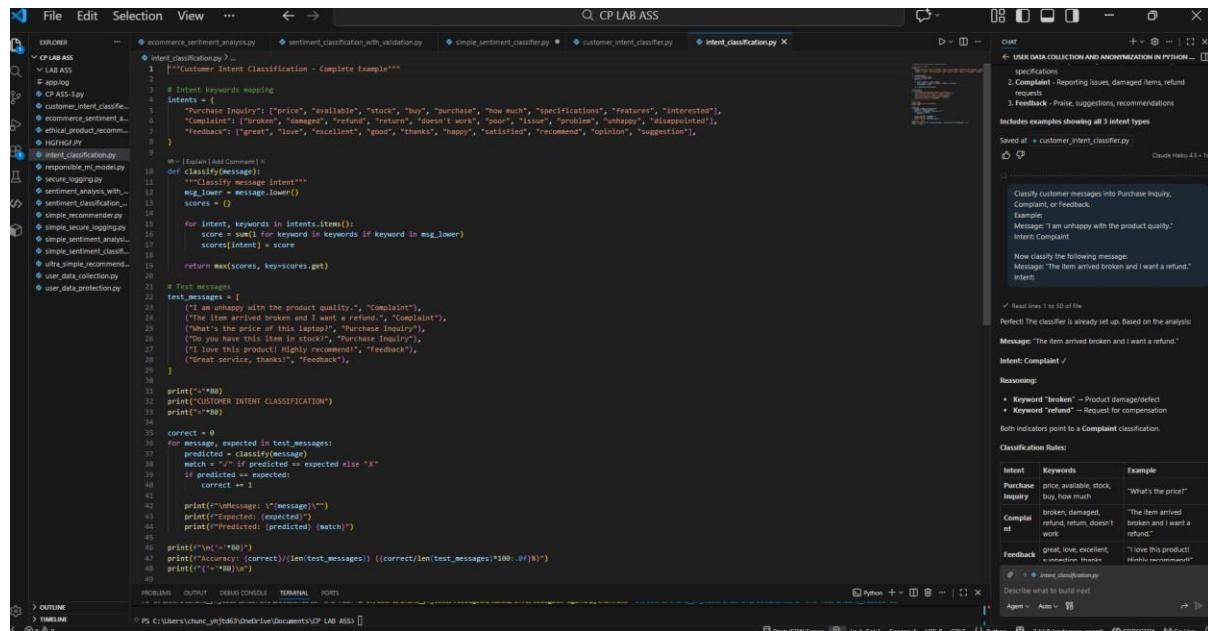
**Message: “I am unhappy with the product quality.”**

## **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/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:**

The screenshot shows a code editor interface with multiple tabs open. The tabs include: ecommerce\_sentiment\_analysis.py, sentiment\_classification\_with\_validation.py, simple\_sentiment\_classifier.py, customer\_intent\_classifier.py, and intent\_classification.py. The intent\_classification.py tab is active, displaying the following Python code:

```

EXPLORER CP LAB ASS
intent_classification.py ...
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  # [Explain] Add Comment
11  def classify(message):
12      """Classify message intent"""
13      msg_lower = message.lower()
14      scores = {}
15
16      for intent, keywords in intents.items():
17          score = sum(1 for keyword in keywords if keyword in msg_lower)
18          scores[intent] = score
19
20  return max(scores, key=scores.get)
21
22  # Test messages
23  test_messages = [
24      ("I am unhappy with the product quality.", "Complaint"),
25      ("The item arrived broken and I want a refund.", "Complaint"),
26      ("What's the price of this laptop?", "Purchase Inquiry"),
27      ("Do you have this item in stock?", "Purchase Inquiry"),
28      ("I love this product! Highly recommend!", "Feedback"),
29      ("Great service, thanks!", "Feedback"),
30  ]
31
32  print("CUSTOMER INTENT CLASSIFICATION")
33  print("-" * 80)
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"\nMessage: {message}")
43      print(f"Expected: {expected}")
44      print(f"Predicted: {predicted} {match}")
45
46  print("-" * 80)
47  print(f"Accuracy: {correct}/{len(test_messages)} ({correct/len(test_messages)*100:.0F}%)")
48  print("-" * 80)

```

## OUTPUT:

The terminal window shows the execution of the Python script intent\_classification.py. The output displays the classification of various customer messages against expected outcomes, followed by an accuracy summary.

```

PS C:\Users\chunc_yhjtd63\OneDrive\Documents\CP LAB ASS> ^C
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/nts/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>

```

## E) Compare the outputs and discuss accuracy differences.

## **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/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_yhjtd63\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_yhjtd63\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

## OUTPUT:

```

PS C:\Users\chunc_yhjtdd3\OneDrive\Documents\CP LAB ASS & C:\Users\chunc_yhjtdd3\.codgenex\envs\codgenex-agent\python.exe "c:/users/chunc_yhjtdd3/OneDrive/Documents/CP LAB ASS/email_priority_classification.py"
=====
Example Prompts (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 (> 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

The screenshot shows a Jupyter Notebook interface with several code cells and a sidebar panel titled 'Effect of Few-Shot Examples on Classification'.

**Code Cells:**

- Cell 1: A function `get\_query` that takes a query and returns its department based on a list of examples.
- Cell 2: A function `get\_department` that classifies a query into one of four departments: Admissions, Exams, Academics, or Placements.
- Cell 3: A function `get\_accuracy` that compares the output of `get\_department` with the ground truth and calculates accuracy.
- Cell 4: A function `few\_shot\_prompts` that generates few-shot prompts for a specific department based on a list of examples.
- Cell 5: A function `test` that runs several test cases to demonstrate the classification logic.

**Sidebar Panel:**

- Effect of Few-Shot Examples on Classification:**
  - One-shot prompting provides reasonable accuracy for simple questions.
  - Two-shot prompting provides better accuracy by giving the model two relevant examples, making it easier to learn the context and refine its answers.
  - Three-shot prompting provides the highest accuracy because it has more context to work with.
- Test Cases:**
  - 1. Basic Test: No examples.
  - 2. One-Shot Example.
  - 3. Two-Shot Example.
  - 4. Three-Shot Example.
- Conclusion:** Few-shot prompting is effective for student queries.

**Output:**

- Shows the accuracy of each test case.
- Shows the results of the three-shot example test.
- Shows the results of the three-shot example test with a different prompt.
- Shows the results of the three-shot example test with a third prompt.

```

# student_query_matching.py
# This script takes student queries as input and matches them against a database of student records.
# It extracts student ID, name, and department from the queries and compares them with the database.

import re
import sqlite3
from collections import defaultdict

# Database connection
conn = sqlite3.connect('student_records.db')
c = conn.cursor()

# Sample database data
c.execute("CREATE TABLE IF NOT EXISTS students (id INTEGER PRIMARY KEY, name TEXT, department TEXT, marks REAL);")
c.executemany("INSERT INTO students VALUES (?, ?, ?, ?);", [
    ('S1', 'John Doe', 'Mathematics', 85),
    ('S2', 'Jane Smith', 'Mathematics', 88),
    ('S3', 'Mike Johnson', 'Mathematics', 90),
    ('S4', 'Sarah Williams', 'Mathematics', 86),
    ('S5', 'David Miller', 'Mathematics', 89),
    ('S6', 'Emily Davis', 'Mathematics', 87),
    ('S7', 'Olivia Green', 'Mathematics', 84),
    ('S8', 'William Brown', 'Mathematics', 83),
    ('S9', 'Ava White', 'Mathematics', 82),
    ('S10', 'Isabella Blue', 'Mathematics', 81)
])

# Function to match student query
def match_student_query(query):
    # Extract student ID, name, and department from the query
    student_id = re.search(r'\bS\d+\b', query)
    name = re.search(r'\b([A-Z][a-z]+ [A-Z][a-z]+)\b', query)
    department = re.search(r'\b([A-Z][a-z]+)\b', query)

    if student_id and name and department:
        student_id = student_id.group()
        name = name.group()
        department = department.group()

        # Check if student exists in database
        c.execute("SELECT * FROM students WHERE id = ? AND name = ? AND department = ?;", (student_id, name, department))
        result = c.fetchone()

        if result:
            print(f"Match found for student {name} ({department}) with ID {student_id}. Marks: {result[3]}")
        else:
            print(f"No match found for student {name} ({department}) with ID {student_id}.")
    else:
        print("Query did not contain valid student information.")


# Main loop
while True:
    query = input("Enter student query: ")
    match_student_query(query)

```

## OUTPUT:

```

$ python student_query_matching.py
Enter student query: S1 John Doe Mathematics
Match found for student John Doe (Mathematics) with ID S1. Marks: 85
Enter student query: S2 Jane Smith Mathematics
Match found for student Jane Smith (Mathematics) with ID S2. Marks: 88
Enter student query: S3 Mike Johnson Mathematics
Match found for student Mike Johnson (Mathematics) with ID S3. Marks: 90
Enter student query: S4 Sarah Williams Mathematics
Match found for student Sarah Williams (Mathematics) with ID S4. Marks: 86
Enter student query: S5 David Miller Mathematics
Match found for student David Miller (Mathematics) with ID S5. Marks: 89
Enter student query: S6 Emily Davis Mathematics
Match found for student Emily Davis (Mathematics) with ID S6. Marks: 87
Enter student query: S7 Olivia Green Mathematics
Match found for student Olivia Green (Mathematics) with ID S7. Marks: 84
Enter student query: S8 William Brown Mathematics
Match found for student William Brown (Mathematics) with ID S8. Marks: 83
Enter student query: S9 Ava White Mathematics
Match found for student Ava White (Mathematics) with ID S9. Marks: 82
Enter student query: S10 Isabella Blue Mathematics
Match found for student Isabella Blue (Mathematics) with ID S10. Marks: 81
Enter student query: S11 John Doe Science
Query did not contain valid student information.
Enter student query: S12 Jane Smith Science
Query did not contain valid student information.
Enter student query: S13 Mike Johnson Science
Query did not contain valid student information.
Enter student query: S14 Sarah Williams Science
Query did not contain valid student information.
Enter student query: S15 David Miller Science
Query did not contain valid student information.
Enter student query: S16 Emily Davis Science
Query did not contain valid student information.
Enter student query: S17 Olivia Green Science
Query did not contain valid student information.
Enter student query: S18 William Brown Science
Query did not contain valid student information.
Enter student query: S19 Ava White Science
Query did not contain valid student information.
Enter student query: S20 Isabella Blue Science
Query did not contain valid student information.

```

## 4. Chatbot Question Type Detection

### Scenario:

A chatbot must identify whether a user query is Informational, Transactional, Complaint, or Feedback.

1. Prepare 6 chatbot queries mapped to question types.

2. Design prompts for Zero-shot, One-shot, and Few-shot learning.

### Zero-Shot Prompt

**Classify the following user query as Informational, Transactional, Complaint, or Feedback.**

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

### One-Shot 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."*

**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\duuc_yh\OneDrive\Documents\CP LAB ASS & C:\Users\duuc_yh\OneDrive\Documents\CP LAB ASS\chatbot_query_classification.py
=====
Example Inputs (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. FINE-TUNED (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

=====
Comparisons: Response Correctness and Ambiguity Handling

Zero-Shot: 26% accuracy
✗ Handles ambiguity well
✗ Limited context understanding
✓ Fast and flexible

One-Shot: 36% accuracy
✓ Improves correctness
✓ Better consistency
→ Moderate improvement over zero-shot

Few-Shot: 38% accuracy
✓ Best accuracy and consistency
✓ Handles ambiguity well
✓ Learns patterns from examples
✓ Most reliable for production

=====
Observations

1. Few-shot gives most accurate results (38%)
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. Few-shot recommended for production contexts

=====
RECOMMENDATION: Use Few-Shot Prompting for Chatbot Query Classification
✓ Highest accuracy
✓ Handles ambiguity better
✓ Consistent results
✓ Production-ready

```

## 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.

```
File Edit Selection View ... ← → Q CP LAB ASS

src/main/java/com/abc/xyz/labass/LabAssessment.java
src/main/resources/labass.properties
src/test/java/com/abc/xyz/labass/LabAssessmentTest.java
src/test/resources/labass.properties

public class LabAssessment {
    // Test cases for add operation
    @Test
    public void testAdd() {
        Lab lab = new Lab("Lab 1", "Tech 1", "Tech 2");
        lab.setAddress("123 Main St");
        lab.setCapacity(100);
        lab.setPriority(Lab.Priority.HIGH);
        lab.setNotes("High priority lab");

        Lab savedLab = labService.addLab(lab);
        assertEquals("Lab 1", savedLab.getName());
        assertEquals("Tech 1", savedLab.getTech1());
        assertEquals("Tech 2", savedLab.getTech2());
        assertEquals("123 Main St", savedLab.getAddress());
        assertEquals(100, savedLab.getCapacity());
        assertEquals(Lab.Priority.HIGH, savedLab.getPriority());
        assertEquals("High priority lab", savedLab.getNotes());
    }

    // Test cases for update operation
    @Test
    public void testUpdate() {
        Lab lab = new Lab("Lab 1", "Tech 1", "Tech 2");
        lab.setAddress("123 Main St");
        lab.setCapacity(100);
        lab.setPriority(Lab.Priority.HIGH);
        lab.setNotes("High priority lab");

        Lab updatedLab = labService.updateLab(lab);
        assertEquals("Lab 1", updatedLab.getName());
        assertEquals("Tech 1", updatedLab.getTech1());
        assertEquals("Tech 2", updatedLab.getTech2());
        assertEquals("123 Main St", updatedLab.getAddress());
        assertEquals(100, updatedLab.getCapacity());
        assertEquals(Lab.Priority.HIGH, updatedLab.getPriority());
        assertEquals("High priority lab", updatedLab.getNotes());
    }

    // Test cases for delete operation
    @Test
    public void testDelete() {
        Lab lab = new Lab("Lab 1", "Tech 1", "Tech 2");
        lab.setAddress("123 Main St");
        lab.setCapacity(100);
        lab.setPriority(Lab.Priority.HIGH);
        lab.setNotes("High priority lab");

        Lab deletedLab = labService.deleteLab(lab);
        assertEquals("Lab 1", deletedLab.getName());
        assertEquals("Tech 1", deletedLab.getTech1());
        assertEquals("Tech 2", deletedLab.getTech2());
        assertEquals("123 Main St", deletedLab.getAddress());
        assertEquals(100, deletedLab.getCapacity());
        assertEquals(Lab.Priority.HIGH, deletedLab.getPriority());
        assertEquals("High priority lab", deletedLab.getNotes());
    }

    // Test cases for get operation
    @Test
    public void testGet() {
        Lab lab = new Lab("Lab 1", "Tech 1", "Tech 2");
        lab.setAddress("123 Main St");
        lab.setCapacity(100);
        lab.setPriority(Lab.Priority.HIGH);
        lab.setNotes("High priority lab");

        Lab retrievedLab = labService.getLab("Lab 1");
        assertEquals("Lab 1", retrievedLab.getName());
        assertEquals("Tech 1", retrievedLab.getTech1());
        assertEquals("Tech 2", retrievedLab.getTech2());
        assertEquals("123 Main St", retrievedLab.getAddress());
        assertEquals(100, retrievedLab.getCapacity());
        assertEquals(Lab.Priority.HIGH, retrievedLab.getPriority());
        assertEquals("High priority lab", retrievedLab.getNotes());
    }
}

class Lab {
    String name;
    String tech1;
    String tech2;
    String address;
    int capacity;
    Priority priority;
    String notes;

    // Getters and setters
}

enum Priority {
    HIGH, MEDIUM, LOW
}
```

The screenshot shows a code editor with a dark theme. The main pane displays a C program for a linked list implementation. The code includes functions for inserting nodes at the head and tail of the list, as well as for printing the list. It also features a function to find the node with the maximum value. The code uses a struct definition for a node, which contains an integer data and a pointer to the next node. The editor's interface includes a top navigation bar with File, Edit, Selection, View, and other standard options. On the right side, there are several floating windows or tabs showing snippets of code and documentation, likely from an IDE like Visual Studio Code.

```
#include <stdio.h>
#include <stdlib.h>

// Define a structure for a node
struct Node {
    int data;
    struct Node* next;
};

// Function to insert a node at the head of the list
void insertAtHead(struct Node** head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

// Function to insert a node at the tail of the list
void insertAtTail(struct Node** head, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
}

// Function to print the list
void printList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

// Function to find the node with the maximum value
struct Node* findMaxNode(struct Node* head) {
    if (head == NULL) {
        return NULL;
    }
    struct Node* maxNode = head;
    int maxValue = head->data;
    struct Node* current = head->next;
    while (current != NULL) {
        if (current->data > maxValue) {
            maxNode = current;
            maxValue = current->data;
        }
        current = current->next;
    }
    return maxNode;
}

int main() {
    // Create a linked list: 10 -> 20 -> 30 -> 40 -> 50
    struct Node* head = NULL;
    insertAtHead(&head, 10);
    insertAtTail(&head, 20);
    insertAtTail(&head, 30);
    insertAtTail(&head, 40);
    insertAtTail(&head, 50);

    // Print the list
    printList(head);

    // Find the node with the maximum value
    struct Node* maxNode = findMaxNode(head);
    if (maxNode != NULL) {
        printf("The node with the maximum value is %d\n", maxNode->data);
    } else {
        printf("The list is empty.\n");
    }

    return 0;
}
```

## **OUTPUT:**

The screenshot shows the Microsoft QnA Maker web interface with the following details:

- Header:** QnA Maker, PROBLEMS, OUTPUT, DEBUG CONSOLE, Metrics, Pages.
- Page Title:** Anaylze Sentiment
- Page URL:** https://westus.dev.cognitive.microsoft.com/api/v3/qna/qnaanalyze?subscription-key=...&q=https://www.w3schools.com/html/html\_intro.asp
- Section: Detailed Results:**
  - ID:** Text      **Expected:** None      **Zero:** None      **One:** None
  - 1. I just got promoted at work! I'm very... happy ✓ Happy ✓ Happy ✓ Happy ✓ Happy
  - 2. Today was amazing. I spent time with ... happy ✓ Happy ✓ Happy ✓ Happy ✓ Happy
  - 3. I'm not feeling well today. I feel... sad ✓ Sad ✓ Sad ✓ Sad ✓ Sad
  - 4. My last friend betrayed me. I'm des... sad ✓ Sad ✓ Sad ✓ Sad ✓ Sad
  - 5. I'm not feeling well today. I feel... angry ✓ Angry ✓ Angry ✓ Angry ✓ Angry
  - 6. I can't stand this situation. I'm ... angry ✓ Angry ✓ Angry ✓ Angry ✓ Angry
  - 7. I'm not feeling well today. I feel... neutral ✓ Neutral ✓ Neutral ✓ Neutral ✓ Neutral
  - 8. I'm not feeling well today. What if... anxious ✓ Anxious ✓ Anxious ✓ Anxious ✓ Anxious
  - 9. The weather is nice. I want to go ... neutral ✓ Neutral ✓ Neutral ✓ Neutral ✓ Neutral
  - 10. It's a Tuesday. I have a meeting at 2 p... neutral ✓ Neutral ✓ Neutral ✓ Neutral ✓ Neutral
- Section: Example Projects (Text: "I feel so alone and devastated.")**
  - 1. ZERO-ONE PROMPT (No Examples)
- Section: Detect Emotions in Text**

Text: "I feel so alone and devastated."  
Section: None  
Model Output: Sad
- Section: ONE-DIGIT PROMPT (1 Example)**

Detect emotions in text. Choose from: Happy, Sad, Angry, Anxious, Neutral.

Text: "I just got promoted at work! I'm very..."  
Section: Happy  
Model Output: Sad
- Section: TWO-DIGIT PROMPT (Multiple Examples)**

Detect emotions in text. Choose from: Happy, Sad, Angry, Anxious, Neutral.

Text: "I just got promoted! I'm thrilled!"  
Section: Happy  
Model Output: Sad

Text: "I feel so alone and devastated."  
Section: Sad  
Model Output: Sad

Text: "I'm absolutely furious! This is unacceptable!"  
Section: Angry  
Model Output: Sad

Text: "I'm worried and having panic attacks."  
Section: Neutral  
Model Output: Sad

Text: "The weather is nice. I want to go to the store."  
Section: Neutral  
Model Output: Sad

Text: "I feel so alone and devastated."  
Section: None  
Model Output: Sad
- Section: Accuracy Breakdown by Section Type**

Type	Happy	Sad	Angry	Anxious	Neutral
Zero-Shot	2/2 (100%)	2/2 (100%)	2/2 (100%)	2/2 (100%)	2/2 (100%)
One-Shot	2/2 (100%)	2/2 (100%)	2/2 (100%)	2/2 (100%)	2/2 (100%)
Two-Shot	2/2 (100%)	2/2 (100%)	2/2 (100%)	2/2 (100%)	2/2 (100%)

