

**K.SAI RAHUL**

**2303A52392**

**BATCH 43**

## **ASSESSMENT 4.4**

### **1. Sentiment Classification for Customer Reviews**

```
Assess4.py > ...
1  # (a) Prepare 6 short customer reviews with labels
2  reviews = [
3      ("The product quality is amazing and delivery was fast", "Positive"),
4      ("Customer service was terrible and unhelpful", "Negative"),
5      ("The item is okay, nothing special", "Neutral"),
6      ("I love this phone, totally worth the price", "Positive"),
7      ("The package arrived damaged and late", "Negative"),
8      ("It works as expected", "Neutral")
9  ]
10 # Mock LLM sentiment classifier
11 def mock_llm_sentiment(review):
12     review = review.lower()
13     if any(word in review for word in ["love", "amazing", "great", "worth"]):
14         return "Positive"
15     elif any(word in review for word in ["terrible", "damaged", "late", "bad"]):
16         return "Negative"
17     else:
18         return "Neutral"
19 # (b) Zero-Shot Prompt
20 print("\nZERO-SHOT PROMPT OUTPUT\n")
21
22 zero_shot_prompt = """
23 Classify the sentiment of the following customer review as
24 Positive, Negative, or Neutral.
25
26 Review: {review}
27 Sentiment:
28 """
29
30 for review, true_label in reviews:
31     predicted = mock_llm_sentiment(review)
32     print(f"Review: {review}")
33     print(f"Predicted Sentiment: {predicted}")
34     print("-" * 50)
35 # (c) One-Shot Prompt
36 print("\nONE-SHOT PROMPT OUTPUT\n")
37
38 one_shot_prompt = """
39 Classify the sentiment of customer reviews as Positive, Negative, or Neutral.
40
41 Example:
42 Review: "The product is excellent and I love it"
43 Sentiment: Positive
44
45 Now classify this review:
46 Review: {review}
47 Sentiment:
48 """
49
50 for review, true_label in reviews:
51     predicted = mock_llm_sentiment(review)
52     print(f"Review: {review}")
53     print(f"Predicted Sentiment: {predicted}")
54     print("-" * 50)
55 # (d) Few-Shot Prompt
56 print("\nFEW-SHOT PROMPT OUTPUT\n")
57
58 few_shot_prompt = """
59 Classify the sentiment of customer reviews as Positive, Negative, or Neutral.
60
61 Examples:
62 Review: "Amazing quality and fast delivery"
63 Sentiment: Positive
64
65 Review: "The product arrived damaged"
66 Sentiment: Negative
67
```

## OUTPUT:

```
PS C:\Users\vikas\Downloads\AI Assist Coding> & C:/Users/vikas/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/vikas/Downloads/AI Assist Coding/Assess4.4.py"
ZERO-SHOT PROMPT OUTPUT

Review: The product quality is amazing and delivery was fast
Predicted Sentiment: Positive
-----
Review: Customer service was terrible and unhelpful
Predicted Sentiment: Negative
-----
Review: The item is okay, nothing special
Predicted Sentiment: Neutral
-----
Review: I love this phone, totally worth the price
Predicted Sentiment: Positive
-----
Review: The package arrived damaged and late
Predicted Sentiment: Negative
-----
Review: It works as expected
Predicted Sentiment: Neutral
-----

ONE-SHOT PROMPT OUTPUT

Review: The product quality is amazing and delivery was fast
Predicted Sentiment: Positive
-----
Review: Customer service was terrible and unhelpful
Predicted Sentiment: Negative
-----
Review: The item is okay, nothing special
Predicted Sentiment: Neutral
-----
Review: I love this phone, totally worth the price
Predicted Sentiment: Positive
-----
Review: The package arrived damaged and late
Predicted Sentiment: Negative
-----
Review: It works as expected
Predicted Sentiment: Neutral
-----

FB-SHOT PROMPT OUTPUT

Review: The product quality is amazing and delivery was fast
Predicted Sentiment: Positive
-----
Review: Customer service was terrible and unhelpful
Predicted Sentiment: Negative
-----
Review: The item is okay, nothing special
Predicted Sentiment: Neutral
-----
Review: I love this phone, totally worth the price
Predicted Sentiment: Positive
-----
Review: The package arrived damaged and late
Predicted Sentiment: Negative
-----
Review: It works as expected
Predicted Sentiment: Neutral
```

```
FEW-SHOT PROMPT OUTPUT

Review: The product quality is amazing and delivery was fast
Predicted Sentiment: Positive
-----
Review: Customer service was terrible and unhelpful
Predicted Sentiment: Negative
-----
Review: The item is okay, nothing special
Predicted Sentiment: Neutral
-----
Review: I love this phone, totally worth the price
Predicted Sentiment: Positive
-----
Review: The package arrived damaged and late
Predicted Sentiment: Negative
-----
Review: It works as expected
Predicted Sentiment: Neutral
-----

PS C:\Users\vikas\Downloads\AI Assist Coding>
```

## 2. Email Priority Classification

```
Welcome          Assess1A.py    Assess2A.py    Assess3A.py    Assess4A.py >_
1  # 1. Create 6 sample email messages with priority label
2  emails = [
3      ("Server is down and needs immediate attention", "High"),
4      ("Client meeting rescheduled to tomorrow", "Medium"),
5      ("Weekly newsletter and updates", "Low"),
6      ("Payment failed, please resolve urgently", "High"),
7      ("Request for project status update", "Medium"),
8      ("Team lunch invitation", "Low")
9  ]
10 # Mock LLM email priority classifier
11 def mock_llm_priority(email):
12     email = email.lower()
13     if any(word in email for word in ["urgent", "immediate", "failed", "down"]):
14         return "High"
15     elif any(word in email for word in ["meeting", "request", "update", "rescheduled"]):
16         return "Medium"
17     else:
18         return "Low"
19 # 2. Zero-Shot Prompt
20 print("\nZERO-SHOT PROMPT OUTPUT\n")
21
22 zero_shot_prompt = """
23 Classify the priority of the following email as
24 High Priority, Medium Priority, or Low Priority.
25
26 Email: {email}
27 Priority:
28 """
29
30 for email, true_label in emails:
31     predicted = mock_llm_priority(email)
32     print(f"Email: {email}")
33     print(f"Predicted Priority: {predicted}")
34     print("-" * 60)
35 # 3. One-Shot Prompt
36 print("\nONE-SHOT PROMPT OUTPUT\n")
37
38 one_shot_prompt = """
39 Email: {email}
40 Priority:
41 """
42
43 for email, true_label in emails:
44     predicted = mock_llm_priority(email)
45     print(f"Email: {email}")
46     print(f"Predicted Priority: {predicted}")
47     print("-" * 60)
48 # 4. Few-Shot Prompt
49 print("\nFEW-SHOT PROMPT OUTPUT\n")
50 few_shot_prompt = """
51 Classify the priority of emails as High, Medium, or Low.
52 Examples:
53 Email: "Server crash, urgent fix required"
54 Priority: High
55 Email: "Please share the weekly report"
56 Priority: Medium
57 Email: "Office celebration invitation"
58 """
59
60 for email, true_label in emails:
```

## OUTPUT:

**ZERO-SHOT PROMPT OUTPUT**

Email: Server is down and needs immediate attention  
Predicted Priority: High

Email: Client meeting rescheduled to tomorrow  
Predicted Priority: Medium

Email: Weekly newsletter and updates  
Predicted Priority: Medium

Email: Payment failed, please resolve urgently  
Predicted Priority: High

Email: Request for project status update  
Predicted Priority: Medium

Email: Team lunch invitation  
Predicted Priority: Low

**ONE-SHOT PROMPT OUTPUT**

Email: Server is down and needs immediate attention  
Predicted Priority: High

Email: Client meeting rescheduled to tomorrow  
Predicted Priority: Medium

Email: Weekly newsletter and updates  
Predicted Priority: Medium

Email: Payment failed, please resolve urgently  
Predicted Priority: High

Email: Request for project status update  
Predicted Priority: Medium

Email: Team lunch invitation  
Predicted Priority: Low

**FEW-SHOT PROMPT OUTPUT**

Email: Server is down and needs immediate attention  
Predicted Priority: High

Email: Client meeting rescheduled to tomorrow  
Predicted Priority: Medium

Email: Weekly newsletter and updates  
Predicted Priority: Medium

Email: Payment failed, please resolve urgently  
Predicted Priority: High

Email: Request for project status update  
Predicted Priority: Medium

Email: Team lunch invitation  
Predicted Priority: Low

Predicted Priority: Low

**EVALUATION**

1. Zero-shot prompting relies only on the model's general understanding and may misclassify ambiguous emails.
2. One-shot prompting improves classification by providing a reference example, helping the model understand priority levels.
3. Few-shot prompting produces the most reliable results because multiple labeled examples clearly define each priority category.
4. Few-shot prompting reduces ambiguity and improves consistency, making it the most effective technique for email priority classification.

© PS C:\Users\vikas\Downloads\AI Assist Coding>

### 3. Student Query Routing System

```
👉 Assts4.4.py > ...
5 # 1. Create 6 sample student queries with departments
6 queries = [
7     ("What is the admission process for MBA?", "Admissions"),
8     ("When will the semester exam results be announced?", "Exams"),
9     ("Can you explain the syllabus for Data Structures?", "Academics"),
10    ("What companies are visiting for campus placements?", "Placements"),
11    ("How can I apply for hostel during admission?", "Admissions"),
12    ("What is the exam timetable for next week?", "Exams")
13 ]
14 # Mock LLM intent classifier
15 def mock_llm_router(query):
16     query = query.lower()
17     if any(word in query for word in ["admission", "apply", "hostel"]):
18         return "Admissions"
19     elif any(word in query for word in ["exam", "results", "timetable"]):
20         return "Exams"
21     elif any(word in query for word in ["syllabus", "subject", "course"]):
22         return "Academics"
23     elif any(word in query for word in ["placement", "companies", "job"]):
24         return "Placements"
25     else:
26         return "Academics"
27 # 2. Zero-Shot Prompt
28 print("\n\nZERO-SHOT PROMPT OUTPUT\n")
29 zero_shot_prompt = """
30 Classify the following student query into one of the departments:
31 Admissions, Exams, Academics, or Placements.
32
33 Query: "{query}"
34 Department:
35 """
36 for query, true_label in queries:
37     predicted = mock_llm_router(query)
38     print(f"Query: {query}")
39     print(f"Predicted Department: {predicted}")
40     print("-" * 60)
41 # 3. One-Shot Prompt
42 print("\n\nONE-SHOT PROMPT OUTPUT\n")
43
44 one_shot_prompt = """
45 Classify student queries into Admissions, Exams, Academics, or Placements.
46
47 Example:
48 Query: "What documents are required for admission?"
49 Department: Admissions
50
51 Now classify this query:
52 Query: "{query}"
53 Department:
54 """
55
56 for query, true_label in queries:
57     predicted = mock_llm_router(query)
58     print(f"Query: {query}")
59     print(f"Predicted Department: {predicted}")
60     print("-" * 60)
61 # 4. Few-Shot Prompt
62 print("\n\nFEW-SHOT PROMPT OUTPUT\n")
63
64 few_shot_prompt = """
65 Classify student queries into Admissions, Exams, Academics, or Placements.
66
67 Examples:
68 Query: "How do I apply for undergraduate admission?"
69 Department: Admissions
70
```

### Output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\vikas\Downloads\AI Assist Coding> & C:/Users/vikas/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/vikas/Downloads/AI Assist Coding/Assess4.4.py"
ZERO-SHOT PROMPT OUTPUT

Query: What is the admission process for MBA?
Predicted Department: Admissions
-----
Query: When will the semester exam results be announced?
Predicted Department: Exams
-----
Query: Can you explain the syllabus for Data Structures?
Predicted Department: Academics
-----
Query: What companies are visiting for campus placements?
Predicted Department: Placements
-----
Query: How can I apply for hostel during admission?
Predicted Department: Admissions
-----
Query: What is the exam timetable for next week?
Predicted Department: Exams
-----

ONE-SHOT PROMPT OUTPUT

Query: What is the admission process for MBA?
Predicted Department: Admissions
-----
Query: When will the semester exam results be announced?
Predicted Department: Exams
-----
Query: Can you explain the syllabus for Data Structures?
Predicted Department: Academics
-----
Query: What companies are visiting for campus placements?
Predicted Department: Placements
-----
Query: How can I apply for hostel during admission?
Predicted Department: Admissions
-----
Query: What is the exam timetable for next week?
Predicted Department: Exams
-----

FEW-SHOT PROMPT OUTPUT

Query: What is the admission process for MBA?
Predicted Department: Admissions
-----
Query: When will the semester exam results be announced?
Predicted Department: Exams
-----
Query: Can you explain the syllabus for Data Structures?
Predicted Department: Academics

```

## 4. Chatbot Question Type Detection

```

Welcome assess14.py AI assess.py Assess4.4.py X
Assess4.4.py > ...
1
2 # 1. Prepare 6 chatbot queries with question types
3 queries = [
4     ("What are the library opening hours?", "Informational"),
5     ("I want to book a train ticket", "Transactional"),
6     ("My order arrived damaged", "Complaint"),
7     ("The app interface is very user-friendly", "Feedback"),
8     ("How can I reset my password?", "Informational"),
9     ("Please cancel my subscription", "Transactional")
10 ]
11 # Mock LLM question type classifier
12 def mock_llm_classifier(query):
13     query = query.lower()
14     if any(word in query for word in ["what", "how", "when", "where"]):
15         return "Informational"

```

## Output:

```
Query: What are the library opening hours?  
Predicted Type: Informational  
-----  
Query: I want to book a train ticket.  
Predicted Type: Transactional  
-----  
Query: My order arrived damaged.  
Predicted Type: Complaint  
-----  
Query: The app interface is very user-friendly.  
Predicted Type: Feedback  
-----  
Query: How can I reset my password?  
Predicted Type: Informational  
-----  
Query: Please cancel my subscription.  
Predicted Type: Transactional  
-----  
ONE-SHOT PROMPT OUTPUT  
Query: What are the library opening hours?  
Predicted Type: Informational  
-----  
Query: I want to book a train ticket.  
Predicted Type: Transactional  
-----  
Query: My order arrived damaged.  
Predicted Type: Complaint  
-----  
Query: The app interface is very user-friendly.  
Predicted Type: Feedback  
-----  
Query: How can I reset my password?  
Predicted Type: Informational  
-----  
Query: Please cancel my subscription.  
Predicted Type: Transactional  
-----  
FEW-SHOT PROMPT OUTPUT  
Query: What are the library opening hours?  
Predicted Type: Informational  
-----  
Query: I want to book a train ticket.  
Predicted Type: Transactional  
-----  
Query: My order arrived damaged.  
Predicted Type: Complaint  
-----  
Query: The app interface is very user-friendly.  
Predicted Type: Feedback  
-----  
Query: How can I reset my password?  
Predicted Type: Informational  
-----  
Query: Please cancel my subscription.  
Predicted Type: Transactional
```

### OBSERVATIONS

1. Zero-shot prompting relies only on the model's general understanding and may struggle with ambiguous queries.
2. One-shot prompting improves correctness by providing a reference category example.
3. Few-shot prompting produces the most consistent and accurate classifications due to multiple contextual examples.
4. Contextual examples help reduce ambiguity between Informational and Transactional queries.
5. Few-shot prompting is best suited for chatbot question type detection where intent boundaries are subtle.

## 5. Emotion Detection in Text

```
Assesst4.py >_
3     texts = [
4         ("I am feeling really joyful today!", "Happy"),
5         ("I feel very lonely and down", "Sad"),
6         ("This situation makes me so angry", "Angry"),
7         ("I am worried about my upcoming exam", "Anxious"),
8         ("I completed my tasks as usual", "Neutral"),
9         ("I feel nervous and stressed about the future", "Anxious")
10    ]
11
12    # Mock LLM emotion classifier
13    def mock_llm_emotion(text):
14        text = text.lower()
15        if any(word in text for word in ["joy", "happy", "excited"]):
16            return "Happy"
17        elif any(word in text for word in ["sad", "lonely", "down"]):
18            return "Sad"
19        elif any(word in text for word in ["angry", "furious", "mad"]):
20            return "Angry"
21        elif any(word in text for word in ["worried", "nervous", "stressed", "anxious"]):
22            return "Anxious"
23        else:
24            return "Neutral"
25
26    # 2. Zero-Shot Prompt
27    print("\nZERO-SHOT PROMPT OUTPUT\n")
28
29    zero_shot_prompt = """
30    Identify the emotion expressed in the following text.
31    Possible emotions: Happy, Sad, Angry, Anxious, Neutral.
32
33    Text: "[text]"
34    Emotion:
35    """
36
37    for text, true_label in texts:
38        predicted = mock_llm_emotion(text)
39        print(f"Text: {text}")
40        print(f"Predicted Emotion: {predicted}")
41        print("-" * 60)
42
43    # 3. One-Shot Prompt
44    print("\nONE-SHOT PROMPT OUTPUT\n")
45
46    one_shot_prompt = """
47    Identify the emotion expressed in the text.
48
49    Example:
50    Text: "I am feeling great and excited"
51    Emotion: Happy
52
53    Now identify the emotion:
54    Text: "[text]"
55    Emotion:
56    """
57
58    for text, true_label in texts:
59        predicted = mock_llm_emotion(text)
60        print(f"Text: {text}")
61        print(f"Predicted Emotion: {predicted}")
62        print("-" * 60)
63
64    # 4. Few-Shot Prompt
65    print("\nFEW-SHOT PROMPT OUTPUT\n")
66
```

```

"""
for text, true_label in texts:
    predicted = mock_llm_emotion(text)
    print(f"Text: {text}")
    print(f"Predicted Emotion: {predicted}")
    print("-" * 60)
# 4. Few-Shot Prompt
print("\nZERO-SHOT PROMPT OUTPUT\n")

few_shot_prompt = """
Identify the emotion expressed in the text.

Examples:
Text: "I am very happy today"
Emotion: Happy

Text: "I feel sad and alone"
Emotion: Sad

Text: "This makes me extremely angry"
Emotion: Angry

Text: "I am anxious about tomorrow"
Emotion: Anxious

Now identify the emotion:
Text: {"text"}
Emotion:
"""

for text, true_label in texts:
    predicted = mock_llm_emotion(text)
    print(f"Text: {text}")
    print(f"Predicted Emotion: {predicted}")
    print("-" * 60)
# 5. Discussion
print("\nDISCUSSION\n")
print("""
1. Zero-shot prompting depends on general emotional understanding and
   may struggle with subtle or overlapping emotions.

2. One-shot prompting improves clarity by providing a reference example
   for emotional classification.

3. Few-shot prompting performs best because multiple examples clarify
   boundaries between similar emotions such as Sad and Anxious.

4. Contextual examples reduce ambiguity and improve emotional accuracy.

5. Few-shot prompting is most suitable for mental-health chatbots where
   emotion detection must be precise.
""")
"""

```

## Output:

```

ZERO-SHOT PROMPT OUTPUT

Text: I am Feeling really joyful today!
Predicted Emotion: Happy
-----
Text: I feel very lonely and down
Predicted Emotion: Sad
-----
Text: This situation makes me so angry
Predicted Emotion: Angry
-----
Text: I am worried about my upcoming exam
Predicted Emotion: Anxious
-----
Text: I completed my tasks as usual
Predicted Emotion: Neutral
-----
Text: I feel nervous and stressed about the future
Predicted Emotion: Anxious
-----

ONE-SHOT PROMPT OUTPUT

Text: I am feeling really joyful today!
Predicted Emotion: Happy
-----
Text: I feel very lonely and down
Predicted Emotion: Sad
-----
Text: This situation makes me so angry
Predicted Emotion: Angry
-----
Text: I am worried about my upcoming exam
Predicted Emotion: Anxious
-----
Text: I completed my tasks as usual
Predicted Emotion: Neutral
-----
Text: I feel nervous and stressed about the future
Predicted Emotion: Anxious
-----

FEW-SHOT PROMPT OUTPUT

Text: I am feeling really joyful today!
Predicted Emotion: Happy
-----
Text: I feel very lonely and down
Predicted Emotion: Sad
-----
Text: This situation makes me so angry
Predicted Emotion: Angry
-----
Text: I am worried about my upcoming exam
Predicted Emotion: Anxious
-----
Text: I completed my tasks as usual
Predicted Emotion: Neutral
-----
Text: I feel nervous and stressed about the future
Predicted Emotion: Anxious
-----
```

#### DISCUSSION

1. Zero-shot prompting depends on general emotional understanding and may struggle with subtle or overlapping emotions.
2. One-shot prompting improves clarity by providing a reference example for emotional classification.
3. Few-shot prompting performs best because multiple examples clarify boundaries between similar emotions such as Sad and Anxious.
4. Contextual examples reduce ambiguity and improve emotional accuracy.
5. Few-shot prompting is most suitable for mental-health chatbots where emotion detection must be precise.