

Lab Assignment- 4.1

AI Assisted Coding
Sunny Kolluri
2303A52444

Problem Statement 1: Customer Email Classification

```
# Sample customer emails
sample_emails = [
    ("I was charged twice for my subscription this month.", "Billing"),
    ("My application crashes whenever I open it.", "Technical Support"),
    ("I really like the new update. Great job!", "Feedback"),
    ("What are your customer support working hours?", "Others"),
    ("I am unable to login to my account.", "Technical Support")
]

for email, category in sample_emails:
    print(f"Email: {email}")
    print(f"Category: {category}\n")

Email: I was charged twice for my subscription this month.
Category: Billing

Email: My application crashes whenever I open it.
Category: Technical Support

Email: I really like the new update. Great job!
Category: Feedback

Email: What are your customer support working hours?
Category: Others

Email: I am unable to login to my account.
Category: Technical Support

def run_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("MODEL OUTPUT:\n")
    print(output)
    print("-" * 60)

zero_shot_prompt = """
Classify the following customer email into one of these categories:
Billing, Technical Support, Feedback, Others.

Email: "I was charged twice for my subscription this month."
"""

Variables Terminal
```

```
Email: "I was charged twice for my subscription this month."
...
run_prompt(zero_shot_prompt, "Billing")
... PROMPT:

Classify the following customer email into one of these categories:
Billing, Technical Support, Feedback, Others.

Email: "I was charged twice for my subscription this month."
MODEL OUTPUT:
Billing
-----
one_shot_prompt = """
Example:
Email: "My application crashes frequently."
Category: Technical Support

Now classify the following email:
Email: "I really like the new update. Great job!"
"""

run_prompt(one_shot_prompt, "Feedback")
... PROMPT:

Example:
Email: "My application crashes frequently."
Category: Technical Support

Now classify the following email:
Email: "I really like the new update. Great job!"

MODEL OUTPUT:
Feedback
-----
```

```

[9] few_shot_prompt = """
Example 1:
Email: "I was charged twice for my subscription."
Category: Billing

Example 2:
Email: "My app crashes after the update."
Category: Technical Support

Example 3:
Email: "Great service and fast response."
Category: Feedback

Now classify:
Email: "What are your customer support working hours?"
"""

run_prompt(few_shot_prompt, "Others")

... PROMPT:

Example 1:
Email: "I was charged twice for my subscription."
Category: Billing

Example 2:
Email: "My app crashes after the update."
Category: Technical Support

Example 3:
Email: "Great service and fast response."
Category: Feedback

Now classify:
Email: "What are your customer support working hours?"

MODEL OUTPUT:
Others
"""

```

Final Observation

- Zero-shot works for simple cases
- One-shot improves clarity
- Few-shot provides the best accuracy

Problem Statement 2: Intent Classification for Chatbot Queries

```

[11] # Problem Statement-2
# Sample chatbot user queries with their intents

sample_queries = [
    ("I cannot access my account.", "Account Issue"),
    ("Where is my order now?", "Order Status"),
    ("Does this phone support 5G?", "Product Inquiry"),
    ("What are your working hours?", "General Question"),
    ("My password reset link is not working.", "Account Issue"),
    ("When will my package be delivered?", "Order Status")
]

for query, intent in sample_queries:
    print(f"Query: {query}")
    print(f"Intent: {intent}\n")

... Query: I cannot access my account.
Intent: Account Issue

Query: Where is my order now?
Intent: Order Status

Query: Does this phone support 5G?
Intent: Product Inquiry

Query: What are your working hours?
Intent: General Question

Query: My password reset link is not working.
Intent: Account Issue

Query: When will my package be delivered?
Intent: Order Status

[12] def run_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("\nMODEL OUTPUT:\n")
    print(output)
    print("-" * 60)

[13] zero_shot_prompt = """
Classify the following user query into one of these intents:
Account Issue, Order Status, Product Inquiry, General Question.

Query: "Where is my order now?"
"""

run_prompt(zero_shot_prompt, "Order status")

... PROMPT:

Classify the following user query into one of these intents:
Account Issue, Order Status, Product Inquiry, General Question.

Query: "Where is my order now?"

```

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```

Query: "Where is my order now?"
MODEL OUTPUT:
Order Status
-----
[14]  ✓ 0s
one_shot_prompt = """
Example:
Query: "I cannot log into my account."
Intent: Account Issue

Now classify the following query:
Query: "Does this phone support wireless charging?"
"""

run_prompt(one_shot_prompt, "Product Inquiry")

... PROMPT:

Example:
Query: "I cannot log into my account."
Intent: Account Issue

Now classify the following query:
Query: "Does this phone support wireless charging?"

MODEL OUTPUT:
Product Inquiry
-----
[15]  ✓ 0s
few_shot_prompt = """
Example 1:
Query: "I forgot my password."
Intent: Account Issue

Example 2:
Query: "When will my package be delivered?"
Intent: Order Status

Example 3:
Query: "Is this laptop good for gaming?"
Intent: Product Inquiry

Now classify the following query:
Query: "What time does customer support open?"
"""

run_prompt(few_shot_prompt, "General Question")

... PROMPT:

```

Variables Terminal 2:46 PM Python 3

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```

[15]  ✓ 0s
Example 2:
Query: "when will my package be delivered?"
Intent: Order Status

Example 3:
Query: "is this laptop good for gaming?"
Intent: Product Inquiry

Now classify the following query:
Query: "What time does customer support open?"
"""

run_prompt(few_shot_prompt, "General Question")

... PROMPT:

Example 1:
Query: "I forgot my password."
Intent: Account Issue

Example 2:
Query: "When will my package be delivered?"
Intent: Order Status

Example 3:
Query: "Is this laptop good for gaming?"
Intent: Product Inquiry

Now classify the following query:
Query: "What time does customer support open?"

MODEL OUTPUT:
General Question
-----
[16]  ✓ 0s
print("Evaluation Summary:")
print("Zero-shot Output : Order Status")
print("One-shot Output  : Product Inquiry")
print("Few-shot Output  : General Question")

... Evaluation Summary:
Zero-shot Output : Order Status
One-shot Output  : Product Inquiry
Few-shot Output  : General Question

Observation
Zero-shot prompting correctly identifies clear intents but may lack precision for ambiguous queries.
One-shot prompting improves intent clarity by providing a reference example.
Few-shot prompting gives the most accurate and reliable classification due to multiple contextual examples.

Start coding or generate with AI.

```

Variables Terminal 2:46 PM Python 3

Problem Statement 3: Student Feedback Analysis

A screenshot of a Jupyter Notebook interface titled "Lab_Assignment_4_1_2256.ipynb". The notebook contains several code cells and their corresponding outputs.

Cell [17] contains Python code to print sample student feedback with sentiment labels:

```
# Problem statement -3  
# Sample student feedback with sentiment labels  
  
sample_feedback = [  
    ("The course content was very informative.", "Positive"),  
    ("The lectures were boring and unclear.", "Negative"),  
    ("Classes were conducted regularly.", "Neutral"),  
    ("The instructor explained concepts clearly.", "Positive"),  
    ("The syllabus is outdated.", "Negative")  
]  
  
for feedback, sentiment in sample_feedback:  
    print(f"Feedback: {feedback}")  
    print(f"Sentiment: {sentiment}\n")
```

The output shows five entries of feedback and their sentiments:

```
Feedback: The course content was very informative.  
Sentiment: Positive  
  
Feedback: The lectures were boring and unclear.  
Sentiment: Negative  
  
Feedback: Classes were conducted regularly.  
Sentiment: Neutral  
  
Feedback: The instructor explained concepts clearly.  
Sentiment: Positive  
  
Feedback: The syllabus is outdated.  
Sentiment: Negative
```

Cell [18] contains a function definition for "run_prompt":

```
def run_prompt(prompt, output):  
    print("PROMPT:\n")  
    print(prompt)  
    print("\nMODEL OUTPUT:\n")  
    print(output)  
    print("-" * 60)
```

Cell [21] contains code to run the prompt with zero-shot learning:

```
zero_shot_prompt = """  
Classify the following student feedback as:  
Positive, Negative, or Neutral.  
  
Feedback: "The course content was very informative."  
"""  
  
run_prompt(zero_shot_prompt, "Positive")
```

Cell [22] contains the output of the zero-shot classification:

```
PROMPT:  
  
Classify the following student feedback as:  
Positive, Negative, or Neutral.  
  
Feedback: "The course content was very informative."  
  
MODEL OUTPUT:  
Positive
```

A screenshot of a Jupyter Notebook interface titled "Lab_Assignment_4_1_2256.ipynb". The notebook contains several code cells and their corresponding outputs.

Cell [22] contains code for one-shot learning:

```
one_shot_prompt = """  
Example:  
Feedback: "The lectures were boring."  
Sentiment: Negative  
  
Now classify the following feedback:  
  
Feedback: "The assignments were manageable."  
"""  
  
run_prompt(one_shot_prompt, "Neutral")
```

Cell [23] contains code for few-shot learning:

```
few_shot_prompt = """  
Example 1:  
Feedback: "Excellent teaching methods."  
Sentiment: Positive  
  
Example 2:  
Feedback: "The syllabus is outdated."  
Sentiment: Negative  
  
Example 3:  
Feedback: "Classes were conducted regularly."  
Sentiment: Neutral  
  
Now classify the following feedback:  
  
Feedback: "The instructor explained concepts clearly."  
"""  
  
run_prompt(few_shot_prompt, "Positive")
```

Cell [24] contains the output of the few-shot classification:

```
PROMPT:  
  
Example 1:  
Feedback: "Excellent teaching methods."  
Sentiment: Positive  
  
Example 2:  
Feedback: "The syllabus is outdated."  
Sentiment: Negative  
  
Example 3:  
Feedback: "Classes were conducted regularly."  
Sentiment: Neutral  
  
Now classify the following feedback:  
  
Feedback: "The instructor explained concepts clearly."  
"""  
  
run_prompt(few_shot_prompt, "Positive")
```

```

Lab_Assignment_4_1_2256.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text | Run all
[23] ✓ 0s
Example 2:
Feedback: "The syllabus is outdated."
Sentiment: Negative

Example 3:
Feedback: "Classes were conducted regularly."
Sentiment: Neutral

Now classify the following feedback:
Feedback: "The instructor explained concepts clearly."
"""

run_prompt(few_shot_prompt, "Positive")

PROMPT:

Example 1:
Feedback: "Excellent teaching methods."
Sentiment: Positive

Example 2:
Feedback: "The syllabus is outdated."
Sentiment: Negative

Example 3:
Feedback: "Classes were conducted regularly."
Sentiment: Neutral

Now classify the following feedback:
Feedback: "The instructor explained concepts clearly."

MODEL OUTPUT:
Positive
-----
[24] ✓ 0s
print("Evaluation Summary")
print("Zero-shot Output : Positive")
print("One-shot Output : Neutral")
print("Few-shot Output : Positive")

Evaluation Summary:
Zero-shot Output : Positive
One-shot Output : Neutral
Few-shot Output : Positive

Observation
Zero-shot prompting identifies sentiment correctly for clear feedback.
One-shot prompting improves understanding by providing sentiment reference.
Few-shot prompting yields the most accurate results by learning sentiment patterns from multiple examples.

Start coding or generate with AI.

```

The screenshot shows a Jupyter Notebook interface with two code cells. Cell [23] contains a few-shot learning prompt for sentiment classification, followed by three examples and their feedback. Cell [24] prints an evaluation summary comparing zero-shot, one-shot, and few-shot outputs. The notebook also includes an 'Observation' section and a code generation input field.

Problem Statement 4: Course Recommendation System

```

Lab_Assignment_4_1_2256.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text | Run all
[27] ✓ 0s
# Problem statement -4
# Sample learner queries with corresponding course levels

sample_queries = [
    ("I want to learn Python basics.", "Beginner"),
    ("I am new to programming.", "Beginner"),
    ("I know Python and want to learn data structures.", "Intermediate"),
    ("I want to build machine learning models.", "Intermediate"),
    ("I want to master deep learning and transformers.", "Advanced")
]

for query, level in sample_queries:
    print(f"Query: {query}")
    print(f"Level: {level}\n")

...
Query: I want to learn Python basics.
Level: Beginner
Query: I am new to programming.
Level: Beginner
Query: I know Python and want to learn data structures.
Level: Intermediate
Query: I want to build machine learning models.
Level: Intermediate
Query: I want to master deep learning and transformers.
Level: Advanced

[28] ✓ 0s
def run_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("\nMODEL OUTPUT:\n")
    print(output)
    print("." * 60)

zero_shot_prompt = """
Classify the learner query into:
Beginner, Intermediate, or Advanced.

Query: "I want to learn Python basics."
"""

run_prompt(zero_shot_prompt, "Beginner")

PROMPT:

Classify the learner query into:
Beginner, Intermediate, or Advanced.

Query: "I want to learn Python basics."

MODEL OUTPUT:
Beginner

```

The screenshot shows a Jupyter Notebook interface with four code cells. Cells [27] and [28] contain the logic for classifying learner queries into beginner, intermediate, or advanced levels. Cell [29] shows a zero-shot learning prompt for classifying a query. The notebook also includes a PROMPT section and a MODEL OUTPUT section.

File Edit View Insert Runtime Tools Help

Commands + Code + Text | Run all

```

MODEL OUTPUT:
Beginner
-----
[30] 0s
one_shot_prompt = """
Example:
Query: "I am new to programming."
Level: Beginner

Now classify the following query:
Query: "I know Python and want to learn data structures."
"""

run_prompt(one_shot_prompt, "intermediate")

... PROMPT:

Example:
Query: "I am new to programming."
Level: Beginner

Now classify the following query:
Query: "I know Python and want to learn data structures."

MODEL OUTPUT:
Intermediate
-----
[31] 0s
few_shot_prompt = """
Example 1:
Query: "I have no coding experience."
Level: beginner

Example 2:
Query: "I know Python fundamentals."
Level: intermediate

Example 3:
Query: "I want to master deep learning models."
Level: advanced

Now classify the following query:
Query: "I want to learn neural networks from scratch."
"""

run_prompt(few_shot_prompt, "intermediate")

... PROMPT:

Example 1:
Query: "I have no coding experience."
Level: beginner

Example 2:
Query: "I know Python fundamentals."
Level: intermediate

Example 3:
Query: "I want to master deep learning models."
Level: advanced

Now classify the following query:
Query: "I want to learn neural networks from scratch.

MODEL OUTPUT:
Intermediate
-----
```

Variables Terminal 2:46PM Python 3

File Edit View Insert Runtime Tools Help

Commands + Code + Text | Run all

```

[31] 0s
Example 2:
Query: "I know Python fundamentals."
Level: intermediate

Example 3:
Query: "I want to master deep learning models."
Level: advanced

Now classify the following query:
Query: "I want to learn neural networks from scratch."
"""

run_prompt(few_shot_prompt, "intermediate")

... PROMPT:

Example 1:
Query: "I have no coding experience."
Level: beginner

Example 2:
Query: "I know Python fundamentals."
Level: intermediate

Example 3:
Query: "I want to master deep learning models."
Level: advanced

Now classify the following query:
Query: "I want to learn neural networks from scratch.

MODEL OUTPUT:
Intermediate
-----
[32] 0s
print("Evaluation Summary:")
print("zero-shot Output : Beginner")
print("One-shot Output : Intermediate")
print("Few-shot Output : Intermediate")

Evaluation Summary:
zero-shot Output : Beginner
One-shot Output : Intermediate
Few-shot Output : Intermediate

Observation
Zero-shot prompting correctly classifies beginner-level queries.
One-shot prompting improves level detection by giving a reference example.
Few-shot prompting produces the most reliable classification due to multiple skill-level examples.

Start coding or generate with AI.
-----
```

Variables Terminal 2:46PM Python 3

Problem Statement 5: Social Media Post Moderation

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
# Problem statement - 5
# Sample social media posts with moderation categories

sample_posts = [
    ("Check out our new product launch!", "Acceptable"),
    ("You are useless.", "Offensive"),
    ("Click this link to win a free phone!", "Spam"),
    ("Happy to be part of this community.", "Acceptable"),
    ("Buy now and get 90% discount!", "Spam")
]

for post, category in sample_posts:
    print(f"Post: {post}")
    print(f"Category: {category}\n")

Post: Check out our new product launch!
Category: Acceptable

Post: You are useless.
Category: Offensive

Post: Click this link to win a free phone!
Category: Spam

Post: Happy to be part of this community.
Category: Acceptable

Post: Buy now and get 90% discount!
Category: Spam
```

```
[34]
def run_prompt(prompt, output):
    print("PROMPT:\n")
    print(prompt)
    print("MODEL OUTPUT:\n")
    print(output)
    print("-" * 60)

zero_shot_prompt = """
classify the following social media post as:
Acceptable, Offensive, or Spam.

Post: "Click this link to win a free phone!"
"""

run_prompt(zero_shot_prompt, "Spam")
```

```
PROMPT:

classify the following social media post as:
Acceptable, Offensive, or Spam.

Post: "click this link to win a free phone!"

MODEL OUTPUT:
---
```

Variables Terminal 2:46PM Python 3

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
MODEL OUTPUT:
Spam
```

```
[35]
one_shot_prompt = """
Example:
Post: "Buy now and get 90% discount!"
Category: Spam

Now classify the following post:

Post: "You are an idiot."
"""

run_prompt(one_shot_prompt, "Offensive")
```

```
PROMPT:

Example:
Post: "Buy now and get 90% discount!"
Category: Spam

Now classify the following post:
Post: "You are an idiot."

MODEL OUTPUT:
Offensive
```

```
[37]
few_shot_prompt = """
Example 1:
Post: "Check out our new product launch."
Category: Acceptable

Example 2:
Post: "You are useless."
Category: Offensive

Example 3:
Post: "Limited offer! Click now!"
Category: Spam

Now classify the following post:
Post: "Happy to be part of this community."
"""

run_prompt(few_shot_prompt, "Acceptable")
```

```
PROMPT:

Example 1:
Post: "Check out our new product launch."
Category: Acceptable
```

Variables Terminal 2:46PM Python 3

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text | Run all

[37] ✓ Os Post: "You are useless." Category: Offensive

Example 3:
Post: "Limited offer! Click now!"
Category: Spam

Now classify the following post:

Post: "Happy to be part of this community."
""

run_prompt(few_shot_prompt, "Acceptable")

*** PROMPT:

Example 1:
Post: "Check out our new product launch."
Category: Acceptable

Example 2:
Post: "you are useless."
Category: Offensive

Example 3:
Post: "Limited offer! click now!"
Category: Spam

Now classify the following post:

Post: "Happy to be part of this community."

MODEL OUTPUT:

Acceptable

[38] ✓ Os print("Evaluation Summary:")
print("Zero-shot Output : Spam")
print("One-shot Output : Offensive")
print("Few-shot Output : Acceptable")

Evaluation Summary:
Zero-shot Output : Spam
One-shot Output : Offensive
Few-shot Output : Acceptable

Observation

Zero-shot prompting works well for obvious spam content but may fail for subtle offensive language.

One-shot prompting improves classification by providing a single reference example.

Few-shot prompting produces the most accurate moderation results by learning from multiple examples.

[] Start coding or generate with AI.