AI Assisted Coding Lab 4

Shivani Pambidi

2303A52172

Batch-41

Question 1:

Task 1: Sentiment Classification for Customer Reviews

ZERO-SHOT PROMPT:

Read the customer review given below.Classify the sentiment as Positive, Negative, or Neutral.

CODE:

```
reviews = [
    ("The product quality is excellent.", "Positive"),
    ("I am very happy with this purchase.", "Positive"),
    ("Fast delivery and good packaging.", "Positive"),
    ("The item arrived damaged.", "Negative"),
    ("Very bad experience with customer service.", "Negative"),
    ("The product stopped working after two days.", "Negative"),
    ("The product is okay, nothing special.", "Neutral"),
    ("The item matches the description.", "Neutral"),
    ("Delivery was on time.", "Neutral"),
    ("The price is reasonable for this product.", "Neutral")
]

print("ZERO-SHOT SENTIMENT CLASSIFICATION\n")

for i, (review, sentiment) in enumerate[tuple[str, str]](reviews, start=1):
    print(f"{i}. Review: {review}")
    print(f"   Output: {sentiment}\n")
```

OUTPUT:

```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData
\Local\Programs\Python\Python313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/A
I-Assited-Coding/Lab 4.py"
ZERO-SHOT SENTIMENT CLASSIFICATION

1. Review: The product quality is excellent.
   Output: Positive

2. Review: I am very happy with this purchase.
   Output: Positive

3. Review: Fast delivery and good packaging.
   Output: Positive

4. Review: The item arrived damaged.
   Output: Negative

5. Review: Very bad experience with customer service.
   Output: Negative

6. Review: The product stopped working after two days.
```

JUSTIFICATION:

No examples are given to the model; it decides sentiment by understanding the review text. Works quickly because it doesn't need prior labelled data. Useful when you have new reviews and no examples ready. Accuracy can be lower because the model has no guidance. Suitable for simple, clear reviews with obvious sentiment. Easy to implement in assignments or quick analysis.

ONE-SHOT PROMPT:

Analyze each customer review and classify it as Positive, Negative, or Neutral. Example: Input - Customer Review:The product quality is excellent and I am very satisfied. Output - Sentiment: Positive. Now generate a efficient and readable code for 10 customer reviews classifying into sentiments.

CODE:

```
reviews = [
    ("The product quality is excellent.", "Positive"),
    ("I am very happy with this purchase.", "Positive"),
    ("Fast delivery and good packaging.", "Positive"),
    ("The item arrived damaged.", "Negative"),
    ("Very bad experience with customer service.", "Negative"),
    ("The product stopped working after two days.", "Negative"),
    ("The product is okay, nothing special.", "Neutral"),
    ("The item matches the description.", "Neutral"),
    ("Delivery was on time.", "Neutral"),
    ("The price is reasonable for this product.", "Neutral")
]

print("ONE-SHOT SENTIMENT CLASSIFICATION\n")

for i, (review, sentiment) in enumerate[tuple[str, str]](reviews, start=1):
    print(f"{i}. Review: {review}")
    print(f"   Output: {sentiment}\n")
```

OUTPUT:

```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData
\Local\Programs\Python\Python313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/A
I-Assited-Coding/Lab 4.py"
ONE-SHOT SENTIMENT CLASSIFICATION

1. Review: The product quality is excellent.
   Output: Positive

2. Review: I am very happy with this purchase.
   Output: Positive

3. Review: Fast delivery and good packaging.
   Output: Positive

4. Review: The item arrived damaged.
   Output: Negative
```

JUSTIFICATION:

One labeled example is provided to guide the model on how to classify sentiment.Helps the model understand the format of input and output.Improves accuracy compared to zero-shot because the model has a reference.Useful when you have a small number of examples.Still simple to implement but gives better results than zero-shot.Good for classroom or lab exercises to show how prompting works.

Question 2:

Task 2: Email Priority Classification

ONE-SHOT PROMPT:

Classify the email into High Priority, Medium Priority, or LowPriority.Example- "Server is down and needs immediate fix" → High Priority.Now classify the given email and return only the priority.

CODE:

```
emails = [
    "Server is down and needs immediate fix",
    "Meeting scheduled for tomorrow",
    "Monthly performance report attached",
    "Urgent payment issue from client",
    "Team lunch invitation",
    "Password reset not working",
    "Reminder to submit assignment",
    "Casual feedback about website",
    "Request for leave approval",
    "Newsletter from company"
]

def classify_email(email):
    email = email.lower()

    if "urgent" in email or "down" in email or "immediate" in email:
        return "High Priority"
    elif "meeting" in email or "report" in email or "reminder" in email:
        return "Medium Priority"
    else:
        return "Low Priority"

print("One-Shot Classification Output:\n")
for e in emails:
    print(f"Email: {e}")
    print(f"Priority: {classify_email(e)}\n")
```

OUTPUT:

```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\
Local\Programs\Python\Python313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-
Assited-Coding/Lab 4.py"
One-Shot Classification Output:

Email: Server is down and needs immediate fix
Priority: High Priority

Email: Meeting scheduled for tomorrow
Priority: Medium Priority

Email: Monthly performance report attached
Priority: Medium Priority

Email: Urgent payment issue from client
Priority: High Priority

Email: Team lunch invitation
Priority: Low Priority
```

Add to Chat Ctrl+L

JUSTIFICATION:

It uses only one example, so it is simple and easy to understand. The model learns the task by seeing a single sample, which helps it quickly understand what is expected. This approach saves time and uses less input text. It is useful when the task is very clear and small. It also helps the model understand the required output format and works well for basic email classification.

FEW-SHORT PROMPT:

Urgent or problem-related emails are High Priority, work-related emails like meetings or reports are Medium Priority, and general or casual emails are Low Priority.Example: “Server is down” → High Priority, “Meeting tomorrow” → Medium Priority, “Team lunch invite” → Low Priority.Now classify the given email and return only the priority.

CODE:

```python
emails = [
    "Server is down and needs immediate fix",
    "Meeting scheduled for tomorrow",
    "Monthly performance report attached",
    "Urgent payment issue from client",
    "Team lunch invitation",
    "Password reset not working",
    "Reminder to submit assignment",
    "Casual feedback about website",
    "Request for leave approval",
    "Newsletter from company"
]

def classify_email(email):
    email = email.lower()

    high_keywords = ["urgent", "down", "immediate", "payment", "not working"]
    medium_keywords = ["meeting", "report", "reminder", "approval"]
```

```
    for word in high_keywords:
        if word in email:
            return "High Priority"

    for word in medium_keywords:
        if word in email:
            return "Medium Priority"

    return "Low Priority"

print("Few-Shot Classification Output:\n")
for e in emails:
    print(f"Email: {e}")
    print(f"Priority: {classify_email(e)}\n")
```

OUTPUT:

```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\
Local\Programs\Python\Python313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-
Assited-Coding/Lab 4.py"
Few-Shot Classification Output:

Email: Server is down and needs immediate fix
Priority: High Priority

Email: Meeting scheduled for tomorrow
Priority: Medium Priority

Email: Monthly performance report attached
Priority: Medium Priority
```

JUSTIFICATION:

It gives multiple examples, so the model understands the task better. The model learns patterns from different cases, which helps it make better decisions. This method reduces mistakes compared to one-shot prompting. It is helpful when emails are different in nature and not all the same type. It improves the accuracy of classification and is suitable for real-world email systems.

Question 3:

Task 3: Student Query Routing System

ONE-SHOT PROMPT:

You are a university chatbot that routes student queries to the correct department. For example, if a student asks, “When is the last date to apply for admission?”, the correct department is Admissions. Now read the given student query and classify it into one of the following departments: Admissions, Exams, Academics, or Placements. Return only the department name.

CODE:

```
queries = [
    "What is the eligibility for CSE course?",
    "When will semester exams start?",
    "How can I apply for hostel admission?",
    "What is the syllabus for AI subject?",
    "When will placement training begin?",
    "Is there any campus recruitment drive?",
    "How to register for exams?",
    "Who is the faculty for DBMS?"
]
def route_query(query):
    q = query.lower()
    if "admission" in q or "apply" in q or "eligibility" in q:
        return "Admissions"
    elif "exam" in q or "register" in q:
        return "Exams"
    elif "syllabus" in q or "faculty" in q or "subject" in q:
        return "Academics"
    elif "placement" in q or "recruitment" in q:
        return "Placements"
    else:
        return "Academics"
print("One-Shot Classification Output:\n")
for query in queries:
    print(f"Student Query: {query}")
    print(f"Department: {route_query(query)}\n")
```

OUTPUT:

```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\
Local\Programs\Python\Python313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-
Assited-Coding/Lab 4.py"
One-Shot Classification Output:

Student Query: What is the eligibility for CSE course?
Department: Admissions

Student Query: When will semester exams start?
Department: Exams

Student Query: How can I apply for hostel admission?
Department: Admissions

Student Query: What is the syllabus for AI subject?
Department: Academics

Student Query: When will placement training begin?
Department: Placements
Department: Exams
```

JUSTIFICATION:

One-shot prompting uses only one example, so it is easy to understand. It helps the chatbot quickly learn how to route student queries to the correct department. This method saves time and uses less input text. It works well when the departments are clearly defined. It improves accuracy compared to giving no examples at all. Therefore, one-shot prompting is suitable for basic university chatbot systems.

Question 4:

Task 4: Chatbot Question Type Detection

FEW-SHOT PROMPT:

Identify the type of user query as Informational, Transactional, Complaint, or Feedback. Informational queries ask for details, Transactional queries request an action, Complaint queries report a problem, and Feedback queries share opinions. Example: “What are your working hours?” → Informational, “How can I reset my password?” → Transactional, “I was charged twice” → Complaint, “The app is easy to use” → Feedback. Now classify the given query and return only the question type.

CODE:

```
queries = [
    "What are your working hours?",
    "How can I reset my password?",
    "I was charged twice for my order.",
    "The app is very easy to use.",
    "Can I book a ticket online?",
    "Where is my order located?"
]
def classify_query(query):
    q = query.lower()
    if "how" in q or "book" in q or "reset" in q:
        return "Transactional"
    elif "charged" in q or "problem" in q or "not working" in q:
        return "Complaint"
    elif "good" in q or "easy" in q or "nice" in q:
        return "Feedback"
    else:
        return "Informational"
print("Chatbot Question Type Detection:\n")
for query in queries:
    print(f"Query: {query}")
    print(f"Type: {classify_query(query)}\n")
```

OUTPUT:

```
Local\Programs\Python\Python313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-
Assited-Coding/Lab 4.py"
Chatbot Question Type Detection:

Query: What are your working hours?
Type: Informational

Query: How can I reset my password?
Type: Transactional

Query: I was charged twice for my order.
Type: Complaint

Query: The app is very easy to use.
Type: Feedback

Query: Can I book a ticket online?
Type: Transactional

Query: Where is my order located?
Type: Informational
```

JUSTIFICATION:
Few-shot prompting gives multiple examples, so it is easy to understand. The chatbot learns different types of user queries from these examples. This helps the chatbot identify patterns correctly. It reduces mistakes when compared to one-shot prompting. It also improves the accuracy of question type detection. Therefore, few-shot prompting is suitable for real chatbot systems.

QUESTION 5
Task 5: Emotion Detection in Text
ONE-SHOT PROMPT:
You are a mental-health chatbot that detects emotions in user text. The possible emotions are Happy, Sad, Angry, Anxious, and Neutral. For example, if the text is "I feel very joyful and excited today," the emotion is Happy. Now read the given text and identify the emotion. Return only the emotion.
CODE:

```
    "I feel very happy today",
    "I am feeling very lonely",
    "This makes me so angry",
    "I am worried about my exams",
    "I am just doing my daily work"
]
def detect_emotion(text):
    t = text.lower()
    if "happy" in t or "joy" in t or "excited" in t:
        return "Happy"
    elif "sad" in t or "lonely" in t or "cry" in t:
        return "Sad"
    elif "angry" in t or "mad" in t or "furious" in t:
        return "Angry"
    elif "worried" in t or "anxious" in t or "nervous" in t:
        return "Anxious"
    else:
        return "Neutral"
print("One-Shot Emotion Detection:\n")
for text in texts:
    print(f"Text: {text}")
    print(f"Emotion: {detect_emotion(text)}\n")
```

OUTPUT:

```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\
Local\Programs\Python\Python313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-
Assited-Coding/Lab 4.py"
One-Shot Emotion Detection:

Text: I feel very happy today
Emotion: Happy

Text: I am feeling very lonely
Emotion: Sad

Text: This makes me so angry
Emotion: Angry

Text: I am worried about my exams
Emotion: Anxious

Text: I am just doing my daily work
Emotion: Neutral
```

JUSTIFICATION:

One-shot prompting uses only one example, so it is simple and easy to understand. The model learns the emotion detection task quickly by seeing a single sample. This method saves time and uses less input text. It works well when emotions in the text are clear. Therefore, one-shot prompting is suitable for basic emotion detection in a mental-health chatbot.

FEW-SHOT PROMPT:

You are a mental-health chatbot that detects emotions in text: Happy, Sad, Angry, Anxious, or Neutral. For example, “I feel very happy today” → Happy, “I am sad and lonely” → Sad, “This makes me angry” → Angry, “I am anxious about my exams” → Anxious, “I am sitting at home” → Neutral. Now read the given text and return only the emotion.

CODE:

```
texts = [
    "I am feeling excited about my results",
    "I feel very sad today",
    "This problem makes me angry",
    "I am anxious about my interview",
    "I am just sitting at home"
]
def detect_emotion(text):
    t = text.lower()
    happy_words = ["happy", "excited", "joy"]
    sad_words = ["sad", "lonely", "cry"]
    angry_words = ["angry", "mad", "frustrated"]
    anxious_words = ["worried", "anxious", "nervous"]
    for word in happy_words:
        if word in t:
            return "Happy"
    for word in sad_words:
        if word in t:
            return "Sad"
    for word in angry_words:
        if word in t:
            return "Angry"
    for word in anxious_words:
        if word in t:
            return "Anxious"
    return "Neutral"
print("Few-Shot Emotion Detection:\n")
for text in texts:
    print(f"Text: {text}")
    print(f"Emotion: {detect_emotion(text)}\n")
```

OUTPUT:

```
PS C:\Users\pambi\OneDrive\Documents\Desktop\AI-Assited-Coding> & C:\Users\pambi\AppData\
Local\Programs\Python\Python313\python.exe "c:/Users/pambi/OneDrive/Documents/Desktop/AI-
Assited-Coding/Lab 4.py"
Few-Shot Emotion Detection:

Text: I am feeling excited about my results
Emotion: Happy

Text: I feel very sad today
Emotion: Sad

Text: This problem makes me angry
Emotion: Angry

Text: I am anxious about my interview
Emotion: Anxious

Text: I am just sitting at home
Emotion: Neutral
```

JUSTIFICATION:

Few-shot prompting provides multiple examples, which helps the model understand different emotions better. The model learns patterns from many cases, such as happy, sad, angry, anxious, and neutral texts. This approach reduces mistakes compared to one-shot prompting. It also improves the accuracy of emotion detection. Therefore, few-shot prompting is useful for real-world mental-health chatbots.