AIAC- 4

Batch 32, 2303A52065

Customer Email Classification

A company receives a large number of customer emails every day and wants to automatically classify them into the following categories:

• Billing • Technical Support • Feedback • Others

Instead of training a new machine learning model, the company decides to use prompt engineering techniques with an existing large language model.

Tasks

1. Prepare five short sample emails, each belonging to one of the above categories.

2. Write a zero-shot prompt to classify a given email into one of the categories without providing any examples.

3. Write a one-shot prompt by including one labelled email example and ask the model to classify a new email.

4. Write a few-shot prompt by including two or three labelled email examples and ask the model to classify a new email.

5. Compare the outputs obtained using zero-shot, one-shot, and few-shot prompting techniques and briefly comment on their effectiveness

Prompt:

Zero Shot

```
Email1="I was charged twice for my electricity bill this month. "
Email2="My internet connection is not working since yesterday.  "
Email3="The new app update looks great and works smoothly.  "
Email4="Please explain the charges shown in my last invoice."
Email5="I just wanted to thank your team for quick service."

"""Classify the following customer email into one of these categories:
Billing, Technical Support, Feedback, or Others.
Return only the category name."""
```

Code:

```
def classify_email(email):
    email = email.lower()
    if "charged" in email or "invoice" in email or "bill" in email:
        return "Billing"
    elif "connection" in email or "not working" in email or "technical" in email:
        return "Technical Support"
    elif "thank" in email or "great" in email or "feedback" in email:
        return "Feedback"
    else:
        return "Others"
print(classify_email(Email1))
print(classify_email(Email2))
```

Output:

```
Billing
Technical Support
```

Prompt:

One shot

```
"""Example: Email: I was charged twice for my electricity bill.
Category: Billing
Now classify the following email into Billing, Technical Support, Feedback, or Others."""
```

Code:

```
def classify_email(email):
    email = email.strip()
    if "charged" in email or "invoice" in email or "bill" in email:
        return "Billing"
    elif "connection" in email or "not working" in email or "technical" in email:
        return "Technical Support"
    elif "thank" in email or "great" in email or "feedback" in email:
        return "Feedback"
    else:
        return "Others"
print("Email Category:", classify_email(Email3))
print("Email Category:", classify_email(Email4))
```

Output:

```
Email Category: Feedback
Email Category: Billing
```

Prompt:

Few Shots

```
"""Email: I was charged twice for my electricity bill. → Billing
Email: My internet connection is not working since yesterday. → Technical Support
Email: The new app update looks great. → Feedback
Now classify the following email into Billing, Technical Support, Feedback, or Others by writing a python code appropriate for it
Email: Please explain the charges shown in my last invoice."""
```

Code:

```
def classify_email(email):
    billing_keywords = ["charged", "bill", "invoice", "charges", "payment"]
    technical_keywords = ["connection", "not working", "error", "issue", "technical"]
    feedback_keywords = ["great", "thank", "app update", "service", "smoothly"]

    email_lower = email.lower()

    if any(keyword in email_lower for keyword in billing_keywords):
        return "Billing"
    elif any(keyword in email_lower for keyword in technical_keywords):
        return "Technical Support"
    elif any(keyword in email_lower for keyword in feedback_keywords):
        return "Feedback"
    else:
        return "Others"
# Classify the given email
email_to_classify = "Please explain the charges shown in my last invoice."
category = classify_email(email_to_classify)
print(f"The email is classified under: {category}")
```

Output:

```
The email is classified under: Billing
```

Justification:

Zero-shot prompting worked but relied only on general understanding. One-shot prompting improved accuracy by showing one clear example to guide the classification. Few-shot prompting gave the most reliable results because multiple examples clearly defined category patterns. Overall, increasing the number of examples improved clarity and correctness without using any machine learning model.

Intent Classification for Chatbot Queries

A company wants to deploy a chatbot to handle customer queries. Each query must be classified

into one of the following intents:

Account Issue, Order Status, Product Inquiry, or General Question using prompt engineering techniques.

Tasks to be Completed

1. Prepare Sample Data

Create 6 short chatbot user queries, each mapped to one of the four intents.

2. Zero-shot Prompting

Design a prompt that asks the LLM to classify a user query into the given intent categories without examples.

3. One-shot Prompting

Provide one labeled query in the prompt before classifying a new query.

4. Few-shot Prompting

Include 3–5 labeled intent examples to guide the LLM before classifying a new query.

5. Evaluation

Apply all three techniques to the same set of test queries and document differences in performance.

Prompt:

Zero shot

```
s1 = "I am unable to log in to my account."
s2 = "My account password is not working."
s3 = "Where is my order? It has not arrived yet."
s4 = "When will my package be delivered?"
s5 = "Does this product support fast charging?"
s6 = "What are your working hours?"

"""Classify the following chatbot query into one of these intents:
Account Issue, Order Status, Product Inquiry, or General Question.
Return only the intent name"""
```

Code:

```
def classify_intent(user_query):
    account_issue_keywords = ["log in", "password", "account"]
    order_status_keywords = ["order", "package", "delivered", "arrived"]
    product_inquiry_keywords = ["product", "support", "fast charging"]
    general_question_keywords = ["working hours", "hours"]

    query_lower = user_query.lower()

    if any(keyword in query_lower for keyword in account_issue_keywords):
        return "Account Issue"
    elif any(keyword in query_lower for keyword in order_status_keywords):
        return "Order Status"
    elif any(keyword in query_lower for keyword in product_inquiry_keywords):
        return "Product Inquiry"
    elif any(keyword in query_lower for keyword in general_question_keywords):
        return "General Question"
    else:
        return "Unknown Intent"
print(classify_intent(s1))
print(classify_intent(s2))
```

Output:

```
Account Issue
Account Issue
```

Prompt:

One shot

```
"""Example:
Query: I am unable to log in to my account.
Intent: Account Issue
Now classify the query into one of these intents by writing an efficient code:
Account Issue, Order Status, Product Inquiry, or General Question."""
def classify_intent(query):
    account_issue_keywords = ["log in", "password", "account", "unable"]
    order_status_keywords = ["order", "arrived", "delivered", "package"]
    product_inquiry_keywords = ["product", "support", "features", "charging"]
    general_question_keywords = ["working hours", "when", "where", "how"]

    query_lower = query.lower()

    if any(keyword in query_lower for keyword in account_issue_keywords):
        return "Account Issue"
    elif any(keyword in query_lower for keyword in order_status_keywords):
        return "Order Status"
    elif any(keyword in query_lower for keyword in product_inquiry_keywords):
        return "Product Inquiry"
    elif any(keyword in query_lower for keyword in general_question_keywords):
        return "General Question"
    else:
        return "Unknown Intent"
print(classify_intent(s3))
print(classify_intent(s4))
```

Code:

Output:

```
Order Status
Order Status
```

Prompt:

Few shot

```
'''
Query: I am unable to log in to my account. → Account Issue
Query: Where is my order? → Order Status
Query: Does this product support fast charging? → Product Inquiry
Query: What are your working hours? → General Question
Now classify the following query into one of these intents by a python code:
Account Issue, Order Status, Product Inquiry, or General Question.'''
```

Code:

```
def classify_query(query):
    account_keywords = ["log in", "account", "password", "unable to access"]
    order_keywords = ["order", "package", "delivered", "shipping", "arrival"]
    product_keywords = ["product", "support", "feature", "specification", "fast charging"]
    general_keywords = ["working hours", "contact", "help", "information", "question"]
    query_lower = query.lower()
    if any(keyword in query_lower for keyword in account_keywords):
        return "Account Issue"
    elif any(keyword in query_lower for keyword in order_keywords):
        return "Order Status"
    elif any(keyword in query_lower for keyword in product_keywords):
        return "Product Inquiry"
    elif any(keyword in query_lower for keyword in general_keywords):
        return "General Question"
    else:
        return "Unknown Intent"
# Test the function with the provided queries
queries = [s1, s2, s3, s4, s5, s6]
for q in queries:
    intent = classify_query(q)
    print(f"Query: {q} → Intent: {intent}")
```

(variable) keyword: str

Output:

```
Query: I am unable to log in to my account. → Intent: Account Issue
Query: My account password is not working. → Intent: Account Issue
Query: Where is my order? It has not arrived yet. → Intent: Order Status
Query: When will my package be delivered? → Intent: Order Status
Query: Does this product support fast charging? → Intent: Product Inquiry
Query: What are your working hours? → Intent: General Question
```

Justification:

Zero-shot prompting was able to classify simple queries correctly but depended only on general understanding. One-shot prompting improved accuracy by showing one clear example before classification. Few-shot prompting produced the most consistent and reliable results because multiple examples clearly defined intent patterns. Overall, providing more examples helped the model better understand intent boundaries without using any machine learning model.

Student Feedback Analysis

A university collects student feedback and wants to categorize comments as Positive, Negative, or Neutral.

Questions:

a) Write a Zero-shot prompt to classify feedback sentiment.

b) Provide a One-shot prompt with one feedback example.

c) Create a Few-shot prompt using multiple labeled feedback samples.

d) Explain how examples improve sentiment classification accuracy.

Zero Shot Prompt:

Code:

```
"""Classify the following student feedback into one of these sentiments:
Positive, Negative, or Neutral.
Return only the sentiment label using python code
Feedback: The lectures were very informative and well explained."""
def classify_feedback(feedback):
    positive_keywords = ["informative", "well explained", "great", "excellent", "good", "helpful"]
    negative_keywords = ["boring", "confusing", "poor", "bad", "unhelpful", "difficult"]
    feedback_lower = feedback.lower()
    if any(word in feedback_lower for word in positive_keywords):
        return "Positive"
    elif any(word in feedback_lower for word in negative_keywords):
        return "Negative"
    else:
        return "Neutral"
feedback = "The lectures were very informative and well explained."
sentiment = classify_feedback(feedback)
print(sentiment)
```

Output:

```
Positive
```

One shot prompt:

Code:

```
"""Example:
Feedback: The teacher explains concepts clearly.
Sentiment: Positive
Now classify the following feedback into Positive, Negative, or Neutral using python code
Feedback: The classes are too fast and hard to understand.
"""
def classify_feedback(feedback):
    negative_keywords = ['too fast', 'hard to understand', 'difficult', 'confusing', 'boring']
    positive_keywords = ['clearly', 'helpful', 'engaging', 'interesting', 'informative']
    feedback_lower = feedback.lower()
    if any(kw in feedback_lower for kw in negative_keywords):
        return 'Negative'
    elif any(kw in feedback_lower for kw in positive_keywords):
        return 'Positive'
    else:
        return 'Neutral'
feedback = "The classes are too fast and hard to understand."
sentiment = classify_feedback(feedback)
print(f"Feedback: {feedback}\nSentiment: {sentiment}")
```

Output:

```
Feedback: The classes are too fast and hard to understand.
Sentiment: Negative
```

Few shot prompt:

```
"""Feedback: The lectures were engaging and helpful. → Positive
Feedback: The syllabus is confusing and poorly structured. → Negative
Feedback: The class timings are okay. → Neutral
Now classify the following feedback into Positive, Negative, or Neutral.
Feedback: The course material is very useful."""
```

Code:

```python
def classify_feedback(feedback):
    positive_keywords = [
        "engaging", "helpful", "useful", "excellent", "great", "fantastic",
        "informative", "well-structured", "clear", "comprehensive", "beneficial",
        "valuable", "insightful", "enjoyable", "effective"
    ]
    negative_keywords = [
        "confusing", "poorly structured", "difficult", "boring", "unhelpful",
        "ineffective", "frustrating", "disappointing", "lacking", "inadequate",
        "complicated", "overwhelming", "tedious"
    ]
    feedback_lower = feedback.lower()
    positive_score = sum(1 for word in positive_keywords if word in feedback_lower)
    negative_score = sum(1 for word in negative_keywords if word in feedback_lower)
    if positive_score > negative_score:
        return "Positive"
    elif negative_score > positive_score:
        return "Negative"
    else:
        return "Neutral"
feedback = "The course material is very useful."
result = classify_feedback(feedback)
print(f"Feedback: {feedback} → {result}")
```

Output:

```
Feedback: The course material is very useful. → Positive
```

Justification:

Providing examples helps the model understand how different sentiments are expressed in feedback. Zero-shot prompting relies only on general understanding and may be less accurate. One-shot prompting improves clarity by showing one labelled example. Few-shot prompting gives the best code because multiple examples clearly define sentiment patterns

---

Course Recommendation System An online learning platform wants to recommend courses by classifying learner queries into Beginner, Intermediate, or Advanced levels.

Questions: a) Write a Zero-shot prompt to classify learner queries.

b) Create a One-shot prompt with one example query.

c) Develop a Few-shot prompt with multiple labeled queries.

d) Discuss how Few-shot prompting improves recommendation quality.

Zero shot Prompt:

Code:

```python
"""Classify the following learner query into one of these levels:
Beginner, Intermediate, or Advanced.
Return only the level name using python.
Query: I am new to programming and want to learn Python from scratch."""
def classify_query(query):
    if "new to programming" in query.lower() or "beginner" in query.lower():
        return "Beginner"
    elif "intermediate" in query.lower() or "experienced" in query.lower():
        return "Intermediate"
    else:
        return "Advanced"
query = "I am new to programming and want to learn Python from scratch."
print(classify_query(query))
```

Output:

```
Beginner
```

One shot prompt:

Code:

```
"""Example:
Query: I have never coded before and want an introduction to Java.
Level: Beginner
Now classify the following learner query into Beginner, Intermediate, or Advanced using python code
Query: I know basic Python and want to improve my problem-solving skills"""
def classify_query(query):
    beginner_keywords = ['never coded', 'introduction', 'beginner', 'basic']
    intermediate_keywords = ['improve', 'problem-solving', 'intermediate', 'skills']
    advanced_keywords = ['expert', 'advanced', 'complex', 'master']
    query_lower = query.lower()
    if any(keyword in query_lower for keyword in beginner_keywords):
        return 'Beginner'
    elif any(keyword in query_lower for keyword in intermediate_keywords):
        return 'Intermediate'
    elif any(keyword in query_lower for keyword in advanced_keywords):
        return 'Advanced'
    else:
        return 'Unclassified'
query = "I know basic Python and want to improve my problem-solving skills"
level = classify_query(query)
print(f"Query: {query}\nLevel: {level}")
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE

PS C:\Users\ayesh\AIAC3-2>  & 'c:\Users\ayesh\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\ayesh
ebugpy\launcher' '60710' '--' 'c:\Users\ayesh\AIAC3-2\AIAC4-1.py'
Query: I know basic Python and want to improve my problem-solving skills
Level: Beginner
```

Few Shot:

```
"""Query: I am new to programming and want to learn Python basics. → Beginner
Query: I have some experience with Java and want to learn data structures. → Intermediate
Query: I want to build machine learning models and optimize performance. → Advanced
Now classify the following learner query into Beginner, Intermediate, or Advanced using python code
Query: I know C programming and want to learn algorithms in depth."""
```

Code:

```
def classify_learner_query(query):
    beginner_keywords = ["new to programming", "learn Python basics", "beginner", "getting started", "introduction"]
    intermediate_keywords = ["some experience", "data structures", "intermediate", "familiar with", "basic knowledge"]
    advanced_keywords = ["machine learning models", "optimize performance", "advanced", "in depth", "expert level"]
    query_lower = query.lower()
    if any(keyword in query_lower for keyword in beginner_keywords):
        return "Beginner"
    elif any(keyword in query_lower for keyword in intermediate_keywords):
        return "Intermediate"
    elif any(keyword in query_lower for keyword in advanced_keywords):
        return "Advanced"
    else:
        return "Unclassified"
# Example usage
learner_query = "I know C programming and want to learn algorithms in depth."
classification = classify_learner_query(learner_query)
print(f"The learner query is classified as: {classification}")
```

Output:

```
The learner query is classified as: Advanced
```

Justification:

Few-shot prompting improves recommendation quality by clearly showing how different learning levels are defined. Multiple examples help the model distinguish between beginner, intermediate, and advanced intent more accurately. As a result, course recommendations become more relevant to the learner's actual skill level.

A social media platform wants to classify posts into Acceptable, Offensive, or Spam.

Questions: a) Write a Zero-shot prompt for post moderation. b) Convert it into a One-shot prompt.

c) Design a Few-shot prompt using multiple examples. d) Explain the challenges of Zero-shot prompting in content moderation.

Zero shot prompt:

Code:

```
"""Classify the following social media post as Acceptable, Offensive, or Spam
Post: You won a free iPhone! Click the link now."""
def classify_post(post):
    offensive_keywords = ["hate", "stupid", "idiot", "dumb"]
    spam_keywords = ["free", "click", "win", "prize", "offer"]
    post_lower = post.lower()
    # Check for offensive content
    if any(word in post_lower for word in offensive_keywords):
        return "Offensive"
    # Check for spam content
    if any(word in post_lower for word in spam_keywords):
        return "Spam"
    return "Acceptable"
post = "You won a free iPhone! Click the link now."
classification = classify_post(post)
print(f"The post is classified as: {classification}")
```

Output:

```
The post is classified as: Spam
```

One shot prompt:

Code:

```
"""Example:
Post: This app is useless and stupid.
Category: Offensive
Now classify the following post as Acceptable, Offensive, or Spam.
Post: Get rich fast by joining our online group."""
def classify_post(post):
    offensive_keywords = ["useless", "stupid", "hate", "idiot"]
    spam_keywords = ["get rich", "online group", "click here", "buy now"]
    post_lower = post.lower()
    for keyword in offensive_keywords:
        if keyword in post_lower:
            return "Offensive"
    for keyword in spam_keywords:
        if keyword in post_lower:
            return "Spam"
    return "Acceptable"
# Example usage
post = "Get rich fast by joining our online group."
category = classify_post(post)
print(f"Post: {post}\nCategory: {category}")
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE
Show Command Actions
Command executed now and took 377ms
\Users\ayesh\AppData\Local\Programs\Python\Python313
\Users\ayesh\AIAC3-2\AIAC4-1.py'
Post: Get rich fast by joining our online group.
Category: Spam
```

Few shot prompt:

Code:

```
"""Post: I really like this app, it works well. → Acceptable
Post: This service is horrible and annoying. → Offensive
Post: Limited offer! Buy now and win prizes. → Spam
Now classify the following post as Acceptable, Offensive, or Spam.
Post: Follow this page for free giveaways."""
def classify_post(post):
    spam_keywords = ["limited offer", "buy now", "win prizes", "free giveaways"]
    offensive_keywords = ["horrible", "annoying", "hate", "stupid"]
    post_lower = post.lower()
    for keyword in spam_keywords:
        if keyword in post_lower:
            return "Spam"
    for keyword in offensive_keywords:
        if keyword in post_lower:
            return "Offensive"
    return "Acceptable"
# Example usage
post = "Follow this page for free giveaways."
classification = classify_post(post)
print(f'Post: "{post}" → {classification}')
```

Output:

```
Post: "Follow this page for free giveaways." → Spam
```

Justification:

In zero-shot prompting, no examples are given, so the model may get confused between similar categories like spam and offensive posts. Because of this, zero-shot prompting can give repetitive or incorrect results. Adding examples improves clarity and accuracy.